Hasselt University
and transnational University of Limburg
School of Information Technology

Service Quality Improvement and User Experience Optimization by Introducing Intelligence in the Network

---

Dissertation submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Computer Science
at the transnational University of Limburg
to be defended by

Maarten Wijnants

on May 26, 2010

Supervisor: Prof. dr. Wim Lamotte

2005 – 2010

Universiteit Hasselt
en transnationale Universiteit Limburg
School voor Informatietechnologie

Service Quality Improvement and User Experience Optimization by Introducing Intelligence in the Network

---

Proefschrift voorgelegd tot het behalen van de graad van

Doctor in de Wetenschappen, Informatica

aan de transnationale Universiteit Limburg

te verdedigen door

Maarten Wijnants

op 26 mei, 2010

Promotor: Prof. dr. Wim Lamotte

2005 – 2010

Acknowledgments

No PhD thesis is ever the product of only the candidate's efforts, and certainly this one is no different. A lot of people have contributed to my study and hence to the book you are currently holding in your hands. Mentioning all of them individually would lead to a far too extensive list. I would therefore like to begin by extending a general word of gratitude to everyone who has in some way influenced me and/or my research. Thank you! Some people are however worthy of explicit acknowledgment due to their significant and direct impact on the realization of this dissertation and this will be done in the remainder of this section. My sincere apologies if I have inadvertently omitted anyone to whom personal acknowledgment is due.

First and foremost, I would like to thank my promotor, prof. dr. Wim Lamotte, for his continuous supervision and expert guidance. Thank you Wim for allowing me to explore my research interests, for your support throughout the PhD period and, probably most importantly, for your pointers and suggestions which have guided my research in the right direction. Without you, this dissertation would definitely not be in the state it is in today.

I have had the privilege of conducting my research at the Expertise centre for Digital Media (EDM), the ICT research institute of Hasselt University. I would very much like to thank prof. dr. Frank Van Reeth and prof. dr. Eddy Flerackers, respectively Deputy Managing Director and Managing Director of the EDM, for this wonderful opportunity. Thanks also goes to all present and former colleagues at the EDM for creating such a pleasant working environment. As a member of the Multimedia & Communication Technology research group, a special acknowledgment is due to fellow (ex-)group members with whom I have worked in close collaboration throughout the years: Stijn Agten, Bart Cornelissen, Jeroen Dierckx, Bjorn Geuns, Panagiotis Issaris, dr. Tom Jehaes, dr. Pieter Jorissen, dr. Jori Liesenborgs, dr. Patrick Monsieurs and dr. Peter Quax. I also wish to express my gratitude to my friends and former colleagues dr. Tim Clerckx, Bert Creemers and Chris Vandervelpen for the many nice experiences that we have shared while working together. Maarten Cardinaels and Geert Vanderhulst are thanked for the valuable collaboration during the IBBT iConnect project. I am also grateful to the administrative staff of the EDM, Roger Claes and Ingrid Konings, for their everyday support and in particular for taking care of the formalities that are associated with the successful completion of a PhD. Lastly, I would like to acknowledge the rest of the senior research staff of the EDM which have not yet been mentioned thus far: prof. dr. Philippe Bekaert, prof. dr. Karin Coninx, prof. dr. Kris Luyten and prof. dr. Chris Raymaekers. The EDM and its researchers, including myself, have benefited considerably (and keep on benefiting) from their experience, insights and expertise.

In the course of my PhD study, I have had the honor of meeting and collaborating with researchers from other universities and research institutes. I would particularly like to extend my appreciation to prof. dr. Filip De Turck, dr. Bart De Vleeschauwer, dr. Peter Lambert and Dieter Van de Walle, whose constructive attitude and cooperation in the IBBT E2E QoE project was very much appreciated. Without their help, the work that will be presented in chapters 12 and 13 could not have been realized. I am also grateful to the fellow researches whom I have met at attended scientific conferences for their feedback on my work, the interesting discussions and simply for contributing to having a good time while being away from home.

I am also very much obliged to the members of my PhD jury for their comments and suggestions regarding this text. I hope that they will notice the improvements that have resulted from their most valuable input.

Finally, some words of gratitude on a more personal level are definitely also in order. A very special and warm thanks goes to my parents, Josée and Guy, who have always believed in me and have given me more than ample opportunities for pursuing my ambitions in life. Thank you mom and dad for your love, guidance, motivation and never-ending support! I am also grateful for the many friends I have. Thank you all for making my leisure activities so enjoyable and for staging the necessary diversions throughout the years! Last but certainly not least, I would like to thank my girlfriend, Saadia, for standing by me while I was writing this dissertation and for brightening up my life!

Diepenbeek, May 2010

Abstract

The majority of present-day distributed applications demand a certain level of service from the transportation network and impose a number of performance requirements on it. Failure to meet these requirements will typically degrade the efficiency of the application and, more importantly, will likely have a detrimental impact on the experience of the end-user. Over the years, a number of frameworks has therefore been proposed that enable transportation networks to guarantee a certain Quality of Service (QoS) and the positive implications of these frameworks on application performance have been established. Due to the growing attention to human factors however, the focus is increasingly shifting from pure QoS provision towards user experience optimization. Stated differently, in recent years, the goal of guaranteeing a certain level of network performance is evermore being replaced with the aspiration to ensure a high end-user satisfaction or so-called Quality of Experience (QoE). The current generation of telecommunications networks, and the omnipresent Internet in particular, unfortunately lack elaborate and effective constructs for QoE optimization. Mitigating this deficiency forms the subject of this thesis.

The overall contribution of this doctoral dissertation is the Network Intelligence Proxy (NIProxy), a network intermediary which has been developed to enable QoE manipulation and optimization in IPv4-based computer networks. As is hinted at by its name, the NIProxy's methodology involves the introduction of "intelligence" in the networking infrastructure. This is achieved by accumulating contextual knowledge regarding the transportation network itself, the distributed application and the end-user (and his terminal).

On a finer-grained level, this PhD research contributes to various domains, the first of which being network traffic engineering and application-layer QoS provision. The NIProxy provides two complementary techniques to improve the (multimedia) traffic handling capabilities of the transportation network in which it is deployed. The first technique, network traffic shaping (NTS), enables the in-network coordination and management of the bandwidth consumption of the network traffic that is induced by distributed applications. The second tool, service provision, allows the NIProxy to act as a substrate for the hosting of services that will be applied to the (multimedia) content that flows through the network. Both traffic engineering mechanisms are context-aware and context-adaptive since they have unbridled access to the NIProxy's context repository.

A second concrete research contribution is situated in the area of network intermediary design and implementation. The service provision platform is designed with extensibility in mind. Thanks to a pluggable implementation, the set of provided services can at run-time be modified and extended. This extensible design for instance enables the NIProxy to cope with heterogeneous conditions in dynamic networking environments without requiring a reboot. Secondly, instead of implementing the NTS and service provision frameworks as isolated entities, an integrated design is adopted so that both traffic engineering techniques are allowed to interface and collaborate. The outcome is a holistic solution in which the composing components supplement each other's strong points and improve their potential through cooperation.

Finally, this PhD dissertation presents significant results in the area of context-aware networking and user experience optimization. The ultimate objective of the NIProxy is not to improve the QoS features of the transportation network, but instead to improve the experience of the users which leverage the network to run distributed applications. The viability and the feasibility of the NIProxy as a QoE manipulation and optimization framework is investigated by analyzing the outcomes of numerous experimental evaluations. These studies confirm that the NIProxy is equipped with an elaborate toolset for user experience optimization which enables it to deliver on its objective in a multitude of distributed scenarios, in dynamic environments, under variable network load and amidst heterogeneous mixes of network traffic types. It is also corroborated experimentally that the NIProxy is amenable to cooperation with other QoS provision or QoE optimization solutions and as such can serve as a building block for the construction of larger frameworks.

To conclude, it is worth underscoring what this dissertation is *not* about. The NIProxy is *not* concerned with (methods for) QoE measurement or the QoE concept per se. This thesis also does *not* involve the collection of qualitative feedback by means of user studies. Lastly, the NIProxy is *not* a ready-made solution for QoE optimization; instead, it is a QoE manipulation framework whose actual behavior needs to be attuned to the current context of use. Simply integrating the NIProxy in a transportation network is hence by *no* means a guarantee for success.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem Statement and Motivation

Given the omnipresence of the Internet in today's society, exchanging information between computers that are interconnected through a transportation network might seem like a straightforward issue. This is however largely untrue. Efficiently and effectively disseminating data over a computer network and implementing distributed applications is anything but trivial. For example, as the network traffic volume rises, bandwidth restrictions might become a problem. How should bandwidth shortage be dealt with? Should a number of network flows be disabled or reduced in quality and, if so, exactly which ones? As another example, an increase in the number of involved network traffic types is likely to complicate network traffic management. A TCP flow carrying file data for instance exhibits significantly divergent characteristics and requirements compared to a real-time multimedia stream with stringent delivery constraints. As a result, a differential treatment of heterogeneous network traffic types is advocated. Further complicating matters is the growing level of heterogeneity that is invading many aspects related to computer networking. This evolution is for instance exemplified by the continuous proliferation of networking technologies and protocols and the high amount of diversity that is currently prevailing in the end-user device space.

So far, only technological factors regarding computer networking and distributed application development have been considered. Another important aspect in software and system design, and one that is receiving increasing attention in recent years, is customer or end-user satisfaction. The person who will eventually be using the distributed application should be provided with an experience that is as agreeable as possible. Users who are unsatisfied with the operation or performance of a distributed application are likely to discard it and will probably start looking for an alternative, which will in turn lead to revenue loss for the provider of the abdicated solution (in case we are dealing with a commercial product). This implies that data dissemination over transportation networks should take into account not only technically-oriented issues but also human factors. Note that in this context, the complexity of the distributed application is in many cases directly proportional to the user experience optimization potential. For simple distributed applications, only a limited number of options regarding user experience improvement will typically be available. As the application becomes more advanced, for instance in terms of the number of induced network traffic types, more elaborate possibilities are likely to be unlocked, however at the expense of rendering the user satisfaction optimization process increasingly complex.

The current generation of computer networks, and the Internet in particular, were devised to passively transfer moderate volumes of (textual or binary) data from source to destination. Also, IP-based networking technology and the Internet have been conceived in an era when human dimensions and user experience did not yet receive the attention which they do today. Consequently, traffic management and differentiation features are largely lacking in current network infrastructure since there was initially considered to be little need for them. In recent years however, the networked distribution of tremendous amounts of traffic of heterogeneous types has become a reality. As a result, the importance and even necessity of facilities for influencing and tweaking network data dissemination has risen drastically and continues to grow every day.

As an example, streaming real-time multimedia content imposes high and strict requirements on the networking substrate, for instance in terms of bandwidth guarantees and delay bounds. It is consequently not uncommon that trade-offs will need to be applied due to network resources being insufficiently available. Whenever such situations arise, the experience of the end-user should always be reckoned with. In other words, the detrimental impact of each trade-off decision on the user satisfaction should be minimized. The Internet is unfortunately not equipped to take, let alone to enforce, such trade-off decisions. The likely effect is a user experience that is sub-optimal at best.

1.2 Contributions

The overall contribution of this PhD research is the development of the Network Intelligence Proxy (NIProxy), a network intermediary which can be integrated in the Internet (as well as other IP-based networks) to enhance the network's capabilities with regard to traffic engineering. Based on knowledge accumulated from multiple sources, including the networking substrate and the distributed application, the NIProxy exerts its traffic engineering facilities in an attempt to beneficially influence the experience that is provided to end-users of distributed applications.

From a more fine-grained perspective, this PhD research has yielded multiple contributions in various domains. The first set of contributions is situated in the field of network traffic engineering and application-layer Quality of Service (QoS) provision:

- To enable meaningful traffic engineering coordination, the NIProxy introduces "intelligence" in the transportation network by maintaining a context repository. The NIProxy's contextual knowledge is three-fold and encompasses information regarding the networking infrastructure, the distributed application and the end-user (and his terminal).
- The NIProxy encompasses a network bandwidth brokering framework. The framework draws upon findings from link-sharing research, but applies it on a per application basis (instead of a per link basis) and hence allows for the bandwidth consumption of network traffic that is generated by individual distributed applications to be mediated.
- At the same time, the NIProxy acts as a platform for the provision as well as execution of services. As such, it enables data to be processed and adapted as it is being disseminated throughout the network. An important feature of the service provision platform is its flexibility: service delivery occurs on a per client basis, services can implement generic as well as application-specific functionality and the composition of multiple, independent services is supported to enable collaborative data processing.

A second research domain to which the NIProxy has contributed is network intermediary design and implementation:

- The NIProxy system is completely software-based. A modular and relatively generic software architecture was defined and developed to enable in-network traffic engineering and user experience optimization. On top of this solid foundation, the bandwidth brokering and service provision facilities were implemented.

- The service provision platform was designed with extensibility in mind. Thanks to a pluggable implementation, the set of services that is provided by a particular NIProxy instance can at run-time be modified and extended. This extensible design for instance enables the NIProxy to cope with heterogeneous conditions in dynamic networking environments without requiring a reboot.

- Instead of implementing the bandwidth brokering and service provision frameworks as isolated entities, an integrated design was adopted to enable both traffic engineering techniques to interface and collaborate. The outcome is a holistic solution in which the composing components supplement each other and improve their potential through cooperation.

- To achieve easy exploitation of the NIProxy functionality, the Network Intelligence Layer (NILayer) auxiliary library was developed. This client-side library largely abstracts the existence of the NIProxy for application developers by attending to all aspects concerning client-NIProxy communication.

Finally, this PhD dissertation will present significant results in the area of context-aware networking and user experience optimization:

- The NIProxy does not limit its attention to QoS provision but rather exerts its traffic engineering functionality to address the larger and higher-level issue of user experience optimization.

- To direct the operation of its user experience optimization engine, the NIProxy largely draws from its accumulated contextual intelligence. Care is taken that each traffic engineering decision is in tune with the currently prevailing conditions and that the requirements of the network, the distributed application and the end-user are reconciled in the best possibly way.

- Numerous experimental studies have been performed to investigate the feasibility and viability of the NIProxy. These practical tests have revealed that the NIProxy is equipped with an elaborate toolset for user experience optimization which enables it to deliver on its objective in a multitude of distributed scenarios, in dynamic environments, under variable network load and amidst heterogeneous mixes of network traffic types.

- The NIProxy has been shown to be amenable to cooperation with other QoS provision or user experience optimization solutions. This implies that the NIProxy can serve as a building block for the construction of larger frameworks.

1.3 Terminology

An overview of the terminology that will be used in this PhD dissertation is given below:

Quality of Service (QoS) Quality of Service is a measure of technological performance and excellence as it denotes the capability of systems to guarantee that a certain level of performance will be met. In the field of computer networking, for instance, QoS might involve guaranteeing an upper bound on the latency that will be experienced by network traffic or reserving the amount of bandwidth that is required for transporting a network stream.

Quality of Experience (QoE) Quality of Experience is the formal term that is typically employed in research context to denote the experience witnessed by users, in this case of distributed applications. Contrary to QoS, it is a rather subjective metric that encompasses human dimensions. QoE is a multi-disciplinary concept which for instance involves user expectation, satisfaction and overall experience.

Network flow In this thesis, a network flow or network stream refers to a collection of (related) data that is being transmitted over a computer network from a source to a destination. It is possible for multiple flows to simultaneously exist between the same source and destination. The way network flows are identified might differ depending on the context and particular situation. One possibility to identify a (unicast) network flow is via the network address and port pair of the flow end-points.

Distributed application An application whose operation requires the exchange of one or more types of data over a computer network. A distributed application hence incorporates at least one network flow.

Traffic engineering Traffic management or engineering is a collective term for techniques for optimizing the performance of a telecommunications network by (dynamically) regulating the transmission of data over that network.

Network Traffic Shaping (NTS) A particular type of traffic engineering where the objective is to manage the allocation of network bandwidth. Therefore often also referred to as bandwidth brokering.

Access network An access network is that part of a telecommunications network which connects subscribers to their immediate network provider (see Figure 1.1). Since it represents the final leg (in the downstream direction) of a connection between a remote source and the customer, it is often referred to as the "last mile". The access network typically represents the most problematic part of an end-to-end network connection due to its relatively limited resource capacity (e.g., in terms of bandwidth).

Core network The core network is the central part (i.e., the backbone) of a transportation network which interconnects multiple access networks (see Figure 1.1). Data that is exchanged between different access networks will always need to pass through the network core. Compared to access networks, the network backbone is typically much more provisioned and capacitated in terms of resources.

NIProxy client A (end-)host that is managed by a NIProxy instance and which will hence benefit from the NIProxy's QoE optimization potential.

1.4 Outline

This dissertation is divided into two major parts, which are preceded by two preparatory chapters. These initial chapters serve as introduction and hence aim to sketch the context in which the presented research is situated. Besides the current chapter, the introductory section encompasses the results of a literature study. In particular, chapter 2 will compare the NIProxy with the state of the art and related research that is described in the literature.

Following these preliminary chapters, part I is devoted to the fundamentals of the NIProxy network intermediary and will discuss topics which range all the way from its high-level methodology to its design and low-level implementation. More specifically, part I will be launched with an overview of the NIProxy's objectives and methodology in chapter 3. This chapter will address

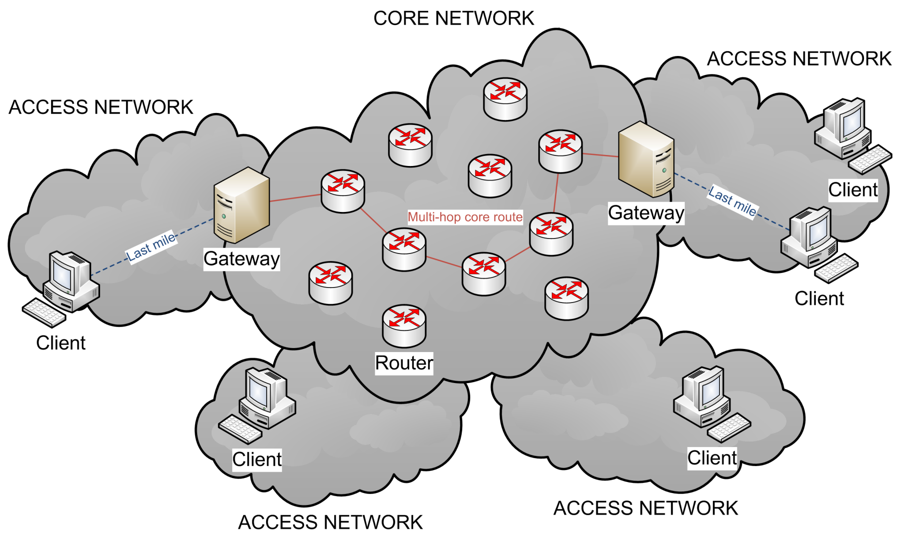


Figure 1.1: An end-to-end network connection between end-hosts located in different access networks.

topics such as context compilation, client-side requirements, auxiliary functionality which facilitates NIProxy exploitation, deployment prospects and the ability to employ the NIProxy as a constituting module in more extensive user experience optimization frameworks. The two subsequent chapters are dedicated to the NIProxy's traffic engineering techniques. In particular, chapter 4 will discuss the NIProxy's network traffic shaping support, whereas chapter 5 will dilate on its multimedia service provision and delivery facilities. The rationale behind both techniques will be presented, as well as crucial decisions regarding their design and implementation. Both chapters will also provide a representative example to further clarify the specific operation of the described traffic engineering mechanisms. The NIProxy's multimedia service provision support will for instance be exemplified through the discussion of a static video transcoding service which enables the NIProxy to adapt the fidelity and hence the bitrate of H.263-encoded video streams. Finally, the design and implementation of the NIProxy's software architecture will be described in chapter 6.

After this rather theoretical discussion of the NIProxy and its facilities, the outcomes and findings of a number of practical studies are clustered in part II. During this PhD research, the NIProxy has been evaluated experimentally numerous times. Every chapter in part II will dwell on a specific aspect of user experience optimization; the chapter order will hereby to a large extent match the chronology of the conducted evaluations. As a result, chapter 7 will first of all present a reference scenario to exemplify the basic functionality of the NIProxy. In particular, at the time of evaluation, only elementary network traffic shaping functionality was supported and the NIProxy's service set exclusively encompassed the static video transcoding service from chapter 5. Furthermore, the scenario was limited to the dissemination of real-time video data and hence did not consider any other type of network traffic. Chapter 7 therefore serves as a point of reference with regard to the NIProxy's QoE optimization potential against which subsequent results and achievements will be offset. The next chapter will for instance embellish the results from the reference scenario by discussing how the NIProxy's network traffic shaping functionality was extended with support for non-real-time network traffic. In particular, chapter 8 will present the architectural modifications that were required to enable the brokering of non-real-time network traffic (e.g., a file transfer). Experimental results will be provided which will comprehensively demonstrate that the NIProxy succeeds in performing network traffic shaping in case the traffic mix is composed of contending real-time and non-real-time network flows. In chapter 9, the NIProxy's real-time and non-real-time bandwidth brokering functionality will be applied to regulate the bandwidth consumption of a concrete, real-world distributed application. The defining characteristic of the considered application is its advanced rendering scheme, which relies on both geometrical and texture-based model representations. The application moreover supports real-time audiovisual communication between participants. Experimental evaluation will verify that the NIProxy is not only able to successfully fulfill the requirements which the application imposes in terms of the (non-real-time) distribution of rendering-related data, but in addition achieves an improvement of the experience that is witnessed by the application's users. Chapter 10 will prove that the NIProxy's traffic engineering techniques are not confined to network traffic that is destined for managed users, but that they instead can just as well be applied on the data which these users inject into the network themselves. It will be experimentally validated that regulating the network traffic which originates from users offers opportunities in terms of QoE optimization not only for the data sources, but indirectly also for hosts which are not explicitly managed by the NIProxy. Next, chapter 11 will explore the issue of data corruption and loss induced during network transmission and will demonstrate how Forward Error Correction (FEC) functionality was incorporated in the NIProxy to tackle its detrimental effect on user experience. Finally, the two subsequent chapters will exemplify that the NIProxy is not only usable as a stand-alone entity but instead is perfectly capable of collaborating with complementary user experience optimization frameworks so that a holistic solution is achieved. More specifically, chapter 12 will introduce the architecture of an integrated two-layer platform in which the NIProxy's functionality is combined with a resilient overlay routing service. The platform exploits its routing functionality to enhance data dissemination in the network backbone, whereas the incorporated NIProxy instances exert their traffic engineering mechanisms to control and direct the delivery of data over the last mile of the network connection. It will be shown that, by bundling the efforts of its constituting components, the two-tier platform achieves (near) end-to-end QoE optimization. In chapter 13, this platform will be further extended by introducing a NIProxy service which allows for H.264/AVC-encoded video flows to be dynamically transformed to an arbitrary bitrate in real-time. Practical findings will confirm the beneficial impact of this novel functionality on the user experience optimization features and effectiveness of the NIProxy and hence, through extrapolation, of the two-tier platform.

Once the results of the practical evaluations will have been presented, it will be time to look back at the problem statement and to determine whether the research objectives have been met. This will be done in chapter 14. Besides drawing overall conclusions, this chapter will enumerate possible directions for future research.

Finally, for reasons of completeness, a collection of superseded results will be presented in appendix A. In particular, this appendix will superficially survey QoE enhancement results that stem from the inceptive phase of this doctoral research and which have been rendered outmoded by later improvements to the NIProxy's implementation.

1.5 Publications

To conclude this introductory chapter, the articles that have been published in the course of this doctoral research are enumerated below. Publications that are not directly related to the work that will be presented in this thesis are marked with a † symbol.

1.5.1 Articles in International Journals

- Pieter Jorissen, *Maarten Wijnants*, and Wim Lamotte. Dynamic Interactions in Physically Realistic Collaborative Virtual Environments. IEEE Transactions on Visualization and Computer Graphics - Special issue on Haptics, Virtual and Augmented Reality, Volume 11, Number 6, p. 649-660, November/December 2005 †

- *Maarten Wijnants*, Wim Lamotte, Bart De Vleeschauwer, Filip De Turck, Bart Dhoedt, Piet Demeester, Peter Lambert, Dieter Van de Walle, Jan De Cock, Stijn Notebaert, and Rik Van de Walle. Optimizing User Quality of Experience Through Overlay Routing, Bandwidth Management and Dynamic Transcoding. Special Issue of the International Journal of Adaptive, Resilient and Autonomic Systems (IJARAS) on the Adaptive and Dependable Mobile Ubiquitous Systems (ADAMUS) Workshop (in press)

1.5.2 Articles in International Conference Proceedings

- Pieter Jorissen, *Maarten Wijnants*, and Wim Lamotte. Using Collaborative Interactive Objects and Animation to Enable Dynamic Interactions in Collaborative Virtual Environments. In: Proceedings of the Conference on Computer Animation and Social Agents (CASA 2004), p. 223-230, Geneva, Switzerland, July 7-9, 2004 †

- Patrick Monsieurs, *Maarten Wijnants*, and Wim Lamotte. Client-controlled QoS Management in Networked Virtual Environments. In: Proceedings of the 4th International Conference on Networking (ICN 2005), p. 268-276, Reunion Island, April 17-21, 2005

- *Maarten Wijnants*, Patrick Monsieurs, Peter Quax, and Wim Lamotte. Exploiting Proxy-Based Transcoding to Increase the User Quality of Experience in Networked Applications. In: Proceedings of the 1st International Workshop on Advanced Architectures and Algorithms for Internet DElivery and Applications (AAA-IDEA 2005), p. 73-80, Orlando, Florida, USA, June 15, 2005

- Peter Quax, *Maarten Wijnants*, Tom Jehaes, and Wim Lamotte, Bridging the Gap between Fixed and Mobile Access to a Large-Scale NVE Incorporating Both Audio and Video. In: Proceedings of the IASTED International Conference on Web Technologies, Applications, and Services (WTAS 2005), Calgary, Canada, July 4-6, 2005

- Peter Quax, Tom Jehaes, *Maarten Wijnants*, and Wim Lamotte. Mobile Adaptations for a Multi-User Framework Supporting Video-Based Avatars. In: Proceedings of the 9th IASTED International Conference on Internet and Multimedia Systems and Applications (IMSA 2005), p. 412-417, Honolulu, Hawaii, USA, August 15-17, 2005

- *Maarten Wijnants*, and Wim Lamotte. Audio and Video Communication in Multiplayer Games through Generic Networking Middleware. In:

Proceedings of the 7th International Conference on Computer Games (CGAMES 2005), p. 52-58, Angoulême, France, November 28-30, 2005

- Maarten Cardinaels, Geert Vanderhulst, *Maarten Wijnants*, Chris Raymaekers, Kris Luyten, and Karin Coninx. Seamless Interaction between Multiple Devices and Meeting Rooms. In: Proceedings of the CHI Workshop on Information Visualization and Interaction Techniques for Collaboration across Multiple Displays (IVITCMD 2006), Montreal, Canada, April 22-23, 2006 †

- *Maarten Wijnants*, Bart Cornelissen, Wim Lamotte, and Bart De Vleeschauwer. An Overlay Network Providing Application-Aware Multimedia Services. In: Proceedings of the 2nd International Workshop on Advanced Architectures and Algorithms for Internet DElivery and Applications (AAA-IDEA 2006), Pisa, Italy, October 10, 2006

- *Maarten Wijnants*, and Wim Lamotte. The NIProxy: a Flexible Proxy Server Supporting Client Bandwidth Management and Multimedia Service Provision. In: Proceedings of the 8th IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM 2007), Helsinki, Finland, June 18-21, 2007

- *Maarten Wijnants*, and Wim Lamotte. Managing Client Bandwidth in the Presence of Both Real-Time and non Real-Time Network Traffic. In: Proceedings of the 3rd IEEE International Conference on COMmunication System softWAre and MiddlewaRE (COMSWARE 2008), Bangalore, India, January 5-10, 2008

- Bart De Vleeschauwer, Filip De Turck, Bart Dhoedt, Piet Demeester, *Maarten Wijnants*, and Wim Lamotte. End-to-end QoE Optimization Through Overlay Network Deployment. In: Proceedings of the 22nd IEEE International Conference on Information Networking (ICOIN 2008), Busan, Korea, January 23-25, 2008

- *Maarten Wijnants*, Tom Jehaes, Peter Quax, and Wim Lamotte. Efficient Transmission of Rendering-Related Data Using the NIProxy. In: Proceedings of the IASTED International Conference on Internet and Multimedia Systems and Applications (EuroIMSA 2008), Innsbruck, Austria, March 17 19, 2008

- *Maarten Wijnants*, Wim Lamotte, Bart De Vleeschauwer, Filip De Turck, Bart Dhoedt, Piet Demeester, Peter Lambert, Dieter Van de Walle, Jan De Cock, Stijn Notebaert, and Rik Van de Walle. Optimizing

User QoE through Overlay Routing, Bandwidth Management and Dynamic Transcoding. In: Proceedings of the 2nd International Workshop on Adaptive and DependAble Mobile Ubiquitous Systems (ADAMUS 2008), Newport Beach, California, USA, June 23, 2008

- *Maarten Wijnants*, and Wim Lamotte. Effective and Resource-Efficient Multimedia Communication Using the NIProxy. In: Proceedings of the 8th IEEE International Conference on Networks (ICN 2009), p. 266-274, Cancun, Mexico, March 1-6, 2009

- *Maarten Wijnants*, and Wim Lamotte. FEC-Integrated Network Traffic Shaping Using the NIProxy. In: Proceedings of the 1st IEEE International Conference on Emerging Network Intelligence (EMERGING 2009), p. 51-60, Sliema, Malta, October 11-16, 2009

- *Maarten Wijnants*, Stijn Agten, Peter Quax, and Wim Lamotte. Investigating the Relationship Between QoS and QoE in a Mixed Desktop/Handheld Gaming Setting. In: Proceedings of the Student Workshop of the 5th ACM International Conference on emerging Networking EXperiments and Technologies (CoNEXT 2009) (poster presentation), p. 29-30, Rome, Italy, December 1, 2009

1.5.3 Internal Technical Reports

- *Maarten Wijnants*, Patrick Monsieurs, and Wim Lamotte. Improving the User Quality of Experience by Incorporating Intelligent Proxies in the Network, EDM Technical Report TR-UH-EDM-0502, April 2005, `http://research.edm.uhasselt.be/~mwijnants/pdf/wijnantsTRMSAN.pdf`

Chapter 2

Background and Related Work

The NIProxy is an intra-network solution for application-layer QoS provision and QoE optimization in IPv4-based transportation networks. This chapter will review the state of the art concerning QoS and QoE research. Since the NIProxy is a context-aware network intermediary whose primary operational tools consist of network traffic shaping and multimedia service provision, specific attention will be given to the subtopics of context-aware networking, bandwidth brokering and service provision and delivery platforms. A survey of Active Networking research will also be presented and its underlying philosophy will be related to the NIProxy's approach to QoE optimization. Finally, a (non-exhaustive) taxonomy of frameworks that are concerned with QoS and/or QoE provision will be developed via the identification of a number of interesting classification criteria and it will be discussed where exactly the NIProxy is situated on each of the identified axes. Throughout this chapter, the NIProxy's distinctive features will be highlighted by contrasting it with kindred systems and research to which it is most closely related.

2.1 Quality of Service Provision

QoS is a well-established notion in software and system engineering and is hence a concept that has been around for quite some time. It is a measure of technological performance and excellence which, broadly speaking, denotes the quality of systems, networks or applications. The QoS construct is hence very extensive and has potential applications in multiple domains. Given the focus and objectives of the NIProxy, the discussion will in this dissertation be constrained to QoS provision in transportation networks. In this context, the term QoS has two possible definitions which are often used interchangeably:

- The ability of the transportation network to *guarantee a minimal performance level*, for instance in terms of throughput, maximal delay, average delay variation, bit error rate, and so on
- The ability of the transportation network to support *heterogeneous treatment* of different applications, services, users or data flows; during its operation, the network could for instance favor (e.g., prioritize) particular types of network traffic or could discriminate between different types of users (e.g., standard, premium, . . .) in terms of network resource allocation

Present-day transportation networks adhere to a layered design; QoS provision can be implemented at any of these layers. The layered networking model is even to a certain degree responsible for the duality of the QoS definition: at the lower levels, QoS provision is typically considered from the guaranteed performance perspective, whereas the majority of the higher-level QoS frameworks swears by the differential treatment definition. Therefore, before actually diving into the related work on QoS mechanisms and architectures, this section will first briefly describe the layered networking model. During the subsequent discussion of the state of the art, it will be indicated for each QoS solution on which layer of the networking model it conceptually operates.

2.1.1 Layered Networking Model

The majority of modern telecommunications networks are designed according to a layered model [Tanenbaum 02]. In this approach, the network is organized as a series of so-called layers or levels which each encompass a collection of semantically similar functions. The purpose of every layer is to offer services to the layer directly above it in the model without exposing the latter to the technicalities which effectively enable the provided services. In other words, layers define a clear interface via which higher layers can easily access and leverage their functionality. Through these interfaces, layers can draw upon the services that are provided by their underlying layer to implement their own functionality. An important advantage of the interfaces-based design is that each layer is rendered functionally independent of the others.

Layered network development has emerged out of practical concerns. Designing complete computer networks is a hard and extremely comprehensive task. By adhering to a layered approach, this task can be broken into smaller, less complex and hence more manageable components (i.e., the design of individual tiers and their corresponding protocols). Another advantage of this approach is that it isolates the implementation of the different layers from each other: as long as its interface remains consistent, the implementation and internals of an individual tier can be updated independently of the others. As such, the layered networking model shows conceptual similarities to the modular programming paradigm.

Conceptually, data exchange between end-hosts occurs horizontally in the layered networking model. In particular, an entity which is located in a certain layer on the source host appears to communicate directly with its peer in the same layer on the destination host. The communication between the peer entities will hereby always follow a particular protocol that has been defined for the network layer to which they belong. In practice however, data is not transferred straightly between corresponding layers of communicating parties. Instead, on the source host the data that needs to be transmitted is passed down iteratively from the originating layer to the layer immediately below it. This process is repeated until the lowest layer in the networking model is reached, after which the data is put on the physical medium, which implements the actual communication. As the data moves down the series of layers on the sending machine, each layer typically will add some extra information to it (e.g., by prefixing a header). This process is usually referred to as *data encapsulation*. After the destination host has extracted the data from the physical medium, the data is passed upward, from layer to layer, until the layer from which the data originated at sender-side is encountered. As the received data climbs up the layer hierarchy at the destination host, *decapsulation* occurs: each layer strips off the additional information which was added by its peer at the source host before passing the data to the next layer.

Presumably the most elaborate model for layered networking is the Open System Interconnection Reference Model (OSI Reference Model or OSI Model). This model is based on a proposal by the International Standards Organization (ISO) and was introduced in an attempt to internationally standardize the

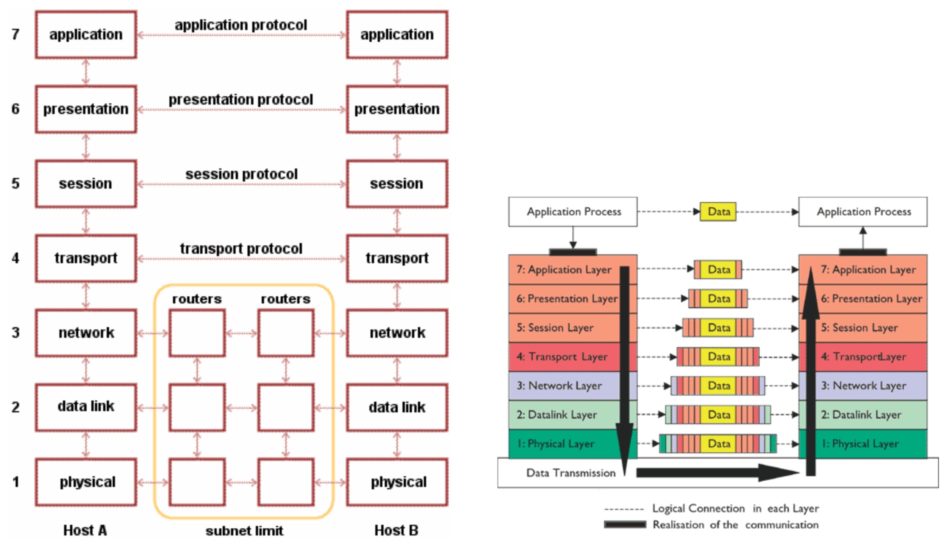


Figure 2.1: Two alternative representations of the OSI reference model.

protocols that are used in the various layers [Tanenbaum 02]. As is illustrated in Figure 2.1, it encompasses seven layers. From top to bottom, these are the application, presentation, session, transport, network, datalink and physical layer. The principal responsibilities of each layer are summarized below:

- The **application layer** is the topmost layer of the model and hence represents the ingress point for distributed applications and their users; in other words, it is the level at which applications access network services and it therefore provides a number of functions which directly support applications (e.g., file transfer functionality)

- The **presentation layer** performs data encoding and furthermore manages security issues by providing services such as data encryption and compression

- The **session layer** allows applications on different computers to establish and terminate sessions; services provided by this layer include dialog control and synchronization

- The **transport layer** segments large data chunks into smaller units (if necessary) and is responsible for delivering each piece to the host for which the data is destined

- The major objective of the **network layer** is to allow heterogeneous networks to be interconnected; it deals with issues such as addressing, routing and congestion control; the network layer is the realm of the ubiquitous IPv4 protocol

- The **datalink layer** breaks input data up into frames and is responsible for the reliable transfer of these frames to the next hop; it consequently deals with error detection and recovery, for instance by relying on acknowledgment-based procedures

- The **physical layer** transmits bits from one computer to another and hence regulates the transmission of a stream of bits over a physical communication channel

QoS provision mechanisms can theoretically be implemented at any echelon of layered network communication models. In reality however, the majority of the QoS solutions concentrate their efforts on either the datalink layer (L2), the network layer (L3) or the application layer (L7) of the OSI model (or the corresponding levels in comparable models). As a rule of thumb, the deeper one descends in the layered network model, the increasingly "technical" QoS provision becomes. Also, evermore specific constructs will be involved and the solution will become less broadly applicable as QoS provision is implemented in lower layers. As an example, network-layer QoS provision might entail the modification of the header of IP packets to achieve prioritized treatment, to influence queue management at intermediate routers, to achieve packet route optimization and so on. In contrast, L7 approaches will generally attempt to improve the performance by leveraging purely application-layer constructs and high-level knowledge of the distributed application. As will become clear later on, the NIProxy operates exclusively at the application layer.

2.1.2 Service Level Agreement

QoS provision typically involves the definition of contracts between the network operator and the customer. These contracts are often denoted by Service Level Agreements (SLAs). Note that a customer in this context not necessarily needs to correspond to an end-user; it might for instance just as well be a Value Added Service (VAS) provider which aims to offer (commercial) products on top of the basic networking service that is provided by the network operator. The objective of QoS-enabled networks is to ensure that agreed upon SLAs will be satisfied, since a SLA breach is likely to result in financial repercussions for the network operator. Note that in the absence of network congestion (i.e., under ideal conditions in which network capacity is sufficiently available), QoS mechanisms will typically not be required to ensure SLA satisfaction. QoS techniques are hence primarily applied at times when a shortage in network resources is noticed. This implies that an alternative solution to QoS support is to over-provision the capacity of the transportation network so that network resources are at all times abundantly available and so that the network will always be able to sustain the injected traffic load. It is apparent however that this approach suffers from a lack of efficiency from an economical and financial perspective.

2.1.3 QoS Provision in IPv4-Based Networks

The most popular network-layer protocol to date is irrefutably the Internet Protocol version 4 (IPv4). It is the "driving force" behind the omnipresent Internet. IPv4 is a relatively lightweight data distribution protocol which does not guarantee packet delivery, nor does it assure proper packet sequencing or avoidance of duplicate delivery. Instead, these responsibilities are left to protocols higher up in the network stack (i.e., transport- or even application-layer protocols). This was a deliberate decision of the IP standardization committee [Postel 81a] as it confers the advantage that complexity is eliminated from the network core and instead is pushed to the end-points of the network connection, where the higher layers of the protocol stack are implemented. The outcome is a highly scalable network design which is considered to be one of the key factors in the success and explosive growth of the Internet [Saltzer 84].

A direct consequence of the IPv4 policy is that each IPv4-based network is limited to providing *best effort data delivery* and by default does not support any form of QoS provision. In particular, each request will be treated identically by the network, no guarantees are given that data will actually be delivered and each network flow will experience an unspecified and unpredictably varying throughput and delivery time, depending on the current traffic load. The best effort behavior of the Internet suffices to host the services and applications for which it was originally devised (i.e., non-real-time services of low complexity such as e-mail, file transfer and information browsing). Recently, the use of the Internet is however increasingly shifting towards real-time services which require the dissemination of one or more types of multimedia data. Compared to the traditional services, the latter have a much higher sensitivity to network anomalies such as packet loss, data corruption, congestion, delay and jitter variation, bandwidth fluctuations, etcetera. Stated differently, modern Internet applications require the transportation network to deliver a certain level of service. This evolution inspired the Internet Engineering Task

Force (IETF), the standardization organization of the IP protocol suite and the Internet in particular [IETF 10], to develop two fundamentally divergent IP-layer QoS solutions: the Integrated Services (IntServ) and the Differentiated Services (DiffServ) architectures.

IntServ

The IntServ architecture, proposed by the IETF in RFC 1633 [Braden 94], is intended to emulate circuit switching (and in particular the QoS provision options which it entails) on packet-switched IPv4 networks[1]. IntServ is a fine-grained QoS solution as it operates on the level of individual network flows [White 97][Clark 92]. Stated differently, applications can request a particular level of service from the transportation network on a per flow basis (e.g., minimum required throughput, maximally tolerable end-to-end delay, etcetera). In case a flow is admitted to the network, the Resource reSerVation Protocol (RSVP) [Braden 97] is employed to actually reserve resources in the network routers which form the end-to-end path between the end-points of the flow, this way assuring that the flow's QoS demands will at all times be satisfied. While its ability to provide deterministic and absolute end-to-end performance guarantees makes IntServ a very powerful approach, it suffers considerably from scalability and manageability issues [Mankin 97]. Each router is namely required to maintain per flow control and forwarding state (e.g., the resources that are reserved for each flow) and every flow needs to be processed individually (e.g., packet scheduling as well as queue and buffer management need to be performed for each flow separately). It is clear that these requirements might pose serious problems in network backbones due to the enormous traffic loads to which they are subjected.

DiffServ

Being faced with the drawbacks of IntServ (and the resulting reluctance of network operators and router vendors to adopt it), the IETF decided to develop a completely new model with the prerequisite that it should be a scalable and readily manageable solution. The outcome was DiffServ [Blake 98, Ramanathan 01], a framework which manages QoS on the level of traffic classes, aggregates of flows with comparable service requirements, rather than considering network flows individually. This implies that all packets which are transported on any flow that belongs to a particular class will be processed

[1]Please see [Tanenbaum 02] for more information regarding circuit switching and its differences with packet switching.

identically by routers. In particular, DiffServ-enabled routers will typically resort to queuing strategies and priority scheduling to achieve a differentiation in the service that is provided to the various traffic classes. As such, the need for maintaining per flow state is eliminated; instead, it suffices for routers to distinguish between a limited number of flow aggregates. At the end-host or the first-hop router, packets are conditioned and classified as belonging to a particular class by filling in the Differentiated Service CodePoint (DSCP) field of the packet's IPv4 header. From that point on, the DSCP mark specifies the per hop processing that the packet will experience at each of the routers along the network path to the remote flow end-point.

In contrast to IntServ, DiffServ is compatible with one of the Internet's fundamental design principles, namely of placing complexity at the network edge while keeping the processing behavior in core routers to a minimum. As a result, DiffServ clearly outperforms IntServ in terms of scalability. On the downside, a direct consequence of its class-based approach is that DiffServ is innately less suited for the provision of absolute QoS guarantees. Instead, DiffServ will typically only provide relative performance assurances (e.g., traffic belonging to the class with the highest QoS guarantees will receive a treatment that is at least as good as the service that is provided to traffic from other classes). An excellent comparison of the IntServ and DiffServ models is presented by Dovrolis and Ramanathan in [Dovrolis 99].

Despite its relatively mature status, its scalability advantages and its behavioral conformity with the infrastructural philosophy of the Internet, DiffServ, like IntServ, thus far has seen far from universal adoption. This can be largely attributed to the reserved attitude of both network operators and service providers. Both IntServ and DiffServ require support from each single router in the IP-based network, support which is typically not available in older routers which at the moment still constitute the bulk of the Internet infrastructure. As a result, huge financial commitments are required from network operators if they want to make their network compliant with the IntServ and/or DiffServ QoS model. Service providers on the other hand would need to modify their services to make optimal use of the unlocked QoS provision possibilities. In summary, due to the hesitating stance of network and service providers alike and due to the reluctance of either party to take the initiative [Huston 00], the Internet to date still largely lacks IntServ and/or DiffServ support and it is very questionable whether this situation will turn around in the future.

Non-Standardized Solutions

Complementary to the standardized IntServ and DiffServ architectures, a plethora of QoS provision solutions has been proposed throughout the years by the research community. Discussing each one in detail is virtually impossible. Moreover, it would exceed the scope of this dissertation since the NIProxy is not concerned with QoS provision per se but instead leverages it as a means to achieve the more high-level objective of user QoE optimization. Therefore, only a subset of the most relevant and noticeable approaches that have been described in the literature will be presented. For the same reason, the NIProxy will not be explicitly related to each of the discussed systems; this section is intended to provide background information and to familiarize laymen with the QoS research domain rather than to discriminate the NIProxy from existing QoS solutions.

De Silva et al. have proposed TOMTEN (TOtal Management of Transmissions for the ENd-user), an application-layer framework for network resource management which aims to maximize user satisfaction [De Silva 99]. The distinctive feature of TOMTEN is its reactive nature: TOMTEN in itself does not make decisions regarding the acceptability of user sessions; instead, it provides the necessary controls to react to end-user decisions. In particular, TOMTEN provides the infrastructure to monitor the state of the system and to present aggregated information to the end-user so that the latter can make informed decisions regarding the actions that need to be taken in case the current session quality is considered to be unsatisfactory. These decisions are translated into resource management actions, which will subsequently be enforced.

But et al. present in [But 08] the Automated Network Games Enhancement Layer (ANGEL) prototype which supports prioritized transmission of network game traffic over bandwidth-constrained links. They in particular propose to decouple traffic classification from the actual prioritization in residential settings by outsourcing the former task to the Internet Service Provider (ISP) gateway. The advantage of this approach is that it allows residential network infrastructure to remain simple (i.e., cheap and easily manageable for home users which typically lack network administration expertise).

An application-layer QoS framework that is targeted specifically at videoconferencing and videotelephony is presented in [Mulroy 06]. Depending on the condition of the communication channel (e.g., the current packet loss rate), the video encoding process at the source is adapted. The system for example exploits a recently drafted profile extension for RTP which enables receivers to provide early feedback to sources as well as the reference picture selection feature of the H.264/AVC video coding standard.

The Mobility And Service Adaptation (MASA) project integrates multiple types of so-called QoS Broker entities which interoperate and interact to achieve adaptive, end-to-end QoS provision in heterogeneous multi-domain and multi-operator environments [Niedermeier 03]. The MASA architecture hereby specifically concentrates on content delivery and adaptation services. In [Kassler 01] and [Kassler 03], two examples of potential QoS Broker functionality are presented. In particular, respectively a filtering scheme for wavelet-encoded video traffic and a QoS-aware media adaptation and transcoding unit are described. While the MASA framework claims to allow underlying network-layer QoS constructs to be leveraged, the two example QoS Brokers operate exclusively at the application level.

Li and Mohapatra advocate in [Li 04] the use of overlay networking technology to achieve QoS provision support in the Internet. Their work is specifically concentrated on the design of QoS-aware overlay routing protocols. The vision of Li and Mohapatra is shared by Subramanian et al., who have proposed the OverQoS system, an overlay-based architecture for enhancing Internet QoS [Subramanian 04]. An important merit of both solutions is their compatibility with prevailing Internet hardware: contrary to for instance IntServ- or DiffServ-based architectures, neither requires non-standard functionality from the underlying Internet infrastructure.

The impact of the use of content transcoding on QoS provision for multimedia traffic has been studied in [Kumar 03]. In particular, the authors propose a protocol which amalgamates components and features of both the IntServ and DiffServ standards to improve multimedia communication over the Internet. Within this protocol, transcoding serves a dual purpose. First of all, it is exploited to avoid network congestion by making amends for bandwidth constraints. Secondly, the transcoding layer adapts multimedia traffic so that it matches end-user requirements (e.g., the characteristics of his terminal).

Another attempt at combining the granularity of IntServ with the scalability benefits of DiffServ is presented in [Kusmierek 02]. An integrated network resource and QoS management framework is described which is structured around the concept of decoupling the QoS control plane from the data plane. The control plane performs network management functions and resource reservation accounting, whereas the data plane is populated by core routers that execute stateless packet scheduling and forwarding (i.e., no per flow state needs to be maintained by the core routers).

All QoS systems that have been discussed thus far can be regarded as general-purpose solutions. An increasing fraction of the QoS research is however targeted at specific environments and domains. These frameworks typically limit the problem scope by focusing on the unique QoS requirements and characteristics of the considered setup. For mobile scenarios for example, notable QoS solutions include the wireless video streaming framework proposed in [Cai 08] and Chen and Heinzelman's multi-layer QoS platform for mobile ad hoc networks (MANETs) [Chen 04]. Another emerging research subdomain for QoS provision is pervasive computing. Significant achievements in this setting include the QoS-aware middleware for ubiquitous environments presented in [Nahrstedt 01], the framework for QoS provision in ontology-centered pervasive systems [Arora 09] and the QoS architecture proposed by Bolla et al. for smart homes [Bolla 08]. As the NIProxy is not particularly targeted at a specific setting but instead is intended as a universally applicable solution for QoS provision (and, more correctly, user QoE improvement), a full-fledged survey of domain-unique QoS frameworks is not in order. Important to note however is that the NIProxy, thanks to its generic methodology, could very well be exploited in many of the fields that are targeted by these focused research efforts. As an example, section A.2 will illustrate that the NIProxy has already been leveraged and (superficially) evaluated in a pervasive meeting room context.

As a final remark, QoS provision is not only being studied in academic contexts or as network backbone technology, it is also slowly penetrating the market of IP consumer devices. As an example, D-Link, a global leader in consumer network connectivity, has developed GameFuel, an intelligent packet processing engine which features dynamic packet fragmentation as well as packet prioritization [D-Link 10]. Routers in the GameFuel line-up promise lag-free gaming experiences by giving game-related traffic precedence for network bandwidth over traffic that is induced by other Internet applications like e-mail or FTP file transfer.

2.1.4 QoS provision in non-IPv4-based networks

Thus far, QoS provision has been exclusively described in the context of IPv4-based networks. Of course, technologies other than IPv4 exist for implementing network communication. Contrary to IPv4, some of these technologies by default even include prominent mechanisms and constructs for QoS provision and hence do not require explicit additional procedures to achieve it.

The connection-oriented Asynchronous Transfer Mode (ATM) standard, for instance, defines a number of QoS parameters (e.g., cell transfer delay, cell loss ratio and cell delay variation) that are negotiated between the customer and the network carrier prior to connection setup. In case the level of service that is requested by the customer is sustainable by the network, the connection will be set up and the ATM network will provide the required QoS for it; on the other hand, in case both parties fail to agree on terms, the connection will not be created at all. This implies that, in case the connection is approved, ATM will provide hard guarantees that the desired QoS will actually be delivered during the complete lifecycle of the connection.

Another protocol with (modest) built-in QoS features is IPv6, the successor of IPv4. QoS support is not the only area in which IPv6 outclasses its precursor; additional examples include an increased IP address space, better mobility support and multihoming options. However, despite the numerous improvements which it introduces and the relative maturity of the standard, IPv6 is still in its infancy in terms of infiltration in operational networks (see, for instance, the survey results presented in [ARIN 08]). Like the IntServ and DiffServ standards, IPv6 appears to be the victim of the success of its predecessor: over the years, IPv4 has become the de facto networking technology and the benefits that are afforded by newer standards seem to be insufficient to justify the high financial repercussions that are associated with a full-scale technology transition.

As a final example that will be described here, the MultiProtocol Label Switching (MPLS) architecture, which was standardized by the IETF in RFC 3031 [Rosen 01], can also be employed to achieve QoS provision. In essence, MPLS is a protocol-agnostic, label-based mechanism for data transmission. Each data packet is assigned a label and routing decisions depend solely on the contents of the packet's label. MPLS conceptually operates at an OSI model layer that is considered to lie between the datalink and the network layer, and is therefore often referred to as a "layer 2.5" protocol. Besides its modest innate QoS support (in the form of fields in the MPLS header), the MPLS architecture can also be exploited to implement QoS on a higher level through packet label interpretation (e.g., divide network traffic into classes based on the labels of their constituting network packets).

As the NIProxy aims to be compliant with the current Internet, QoS provision in non-IPv4-based networks will not be further elaborated on in this thesis. The reader should however keep in mind that alternative network protocols and technologies exist which, contrary to standard IPv4, exhibit interesting assets in terms of QoS provision (and hence, indirectly, QoE optimization).

2.2 Quality of Experience

Attention for Quality of Experience (QoE) has emerged in the late 1990s and has since then increased exponentially. Compared to QoS, it has hence only recently become an active research topic. QoE can be considered the semantic variant of QoS since, broadly speaking, it denotes the overall experience that is witnessed by an end-user or, stated differently, the consumer's satisfaction when using a particular product or service. Somewhat paradoxically perhaps, the majority of research on user experience optimization is focused primarily on technical performance and QoS achievements. This is especially true for initial studies, but a strong attention for technological factors remains explicit in more recent research as well. The aim of these studies is to reveal the impact of the provided level of service on the user experience. For example, Eberle et al. have developed a cross-layer methodology for trading off performance versus energy consumption in wireless communication environments, but they largely neglect actual user satisfaction in the process [Eberle 05]. In [Siller 03b], Siller and Woods hypothesize that QoE is directly proportional to QoS and therefore evaluate and determine QoE by nearly exclusively considering network performance and QoS metrics. In particular, they state that the exploitation of QoS constructs to deliberately introduce service differentiation holds considerable potential for QoE improvement. Soldani has investigated QoE in a telecommunications setting (i.e., for cellular phone users) and claims that QoE performance is defined by service accessibility, retainability, availability and integrity [Soldani 06], which are all purely technical concepts. As a final example, Vanhaverbeke et al. quantify user experience in HDTV video streaming setups as the number of Visible Distortions in Twelve hours (VDTs) and propose Forward Error Correction (FEC) and retransmission schemes as means to improve video quality and hence user satisfaction [Vanhaverbeke 09].

As QoE research evolved, it became clear that user experience is not constricted to technical parameters. QoE researchers therefore increasingly started to consider aspects from non-technological domains, including sociology, cognitive sciences, psychology, usability research, user interface design, context-aware computing, expectation modeling and market studies. For instance, Nokia has investigated the influence of the usability of (mobile) services on user experience [Nokia 04]. As another example, based on the results of a comparative literature study as well as feedback from an expert panel, De Marez and De Moor propose in [De Marez 09] a 5-pillar conceptual model which attempts to cover the most significant QoE dimensions while at the same time integrating QoS parameters. Deryckere et al. have applied this model to assess the experience of users of a mobile wineguide application [Deryckere 08]. A similar multi-dimensional QoE model is presented in [Perkis 06]. Finally, Wu et al. define yet another conceptual QoE framework which allows causal relationships between objective, technical QoS parameters and subjective QoE constructs to be identified [Wu 09]. The mapping methodology is exemplified through a number of empirical studies in a Distributed Interactive Multimedia

Environments (DIME) application.

2.2.1 Definitions

Given its broad scope, definitions for the QoE concept abound in the literature. These definitions have followed the evolution of the QoE research in terms of elaborateness. In particular, the bulk of the initial definitions very narrowly interpreted user experience by concentrating on technical parameters and performance metrics. Kumar [Kumar 05] for example has proposed the following definition:

> Quality of Experience can be defined as the qualitative measure of the daily experience the customer gets when he uses the services he is subscribed to - including experiences such as outages, quality of picture, speed of the Internet service, latency and delay, customer service, etcetera. The better the consumer's experience, the higher his QoE. And that has an effect on customer loyalty.

Another completely technology-oriented interpretation is given in [Dey 08]. In particular, Dey states user satisfaction in a (mobile) video streaming context to be a function of session quality (e.g., initial buffering time, audio-video synchronization), audio-related attributes (e.g., mono versus stereo) and video-related parameters (e.g., image quality, stalling). As can be seen, subjective or contextual factors are not included in this definition. QoS is also the decisive factor in the formulation by Siller and Woods, who define QoE as "the user's perceived experience of what is being presented by the Application Layer, where the Application Layer acts as a user interface front-end that presents the overall result of the individual Quality of Services" [Siller 03a].

More recent formulations acknowledge the multi-dimensional and the interdisciplinary nature of the QoE concept. For instance, Beauregard and Corriveau consider QoE to be "the degree to which a system meets the target user's tacit and explicit expectations for experience" [Beauregard 07]. As another example, the conceptual model proposed in [De Marez 09] accounts for both technical and subjective metrics:

> The model was constructed with the intention to cover not only what the technology does (QoS, performance measurement), but also what people (can) do with the technology, what people want or think to do with it and expect from it, in what context people (intend to) use it, and up to what degree it is meeting their expectations and resulting in an "end-user happiness".

Probably one of the most elaborate and versatile definitions is provided by Wu et al. in [Wu 09]:

> QoE is a multi-dimensional construct of perceptions and behaviors of a user, which represents his/her emotional, cognitive, and behavioral responses, both subjective and objective, while using a system.

Finally, popular informal phrasings of QoE include "user perceived performance", "the degree of satisfaction of the user" and "the number of happy customers". In this dissertation, the term QoE will be wielded loosely to denote the experience or the satisfaction of the user, in this case when consuming a distributed application running over a transportation network.

2.2.2 Measurement

Given the fact that it deals exclusively with objectively measurable parameters, the task of determining QoS performance is relatively straightforward. In contrast, due to its multi-dimensional hallmark, the involvement of purely subjective aspects and its high sensitivity to the context of use, measuring QoE performance is a much more complicated issue. As a result, it has become an important subtopic in the field of QoE research. Soldani for example proposes the use of a Mobile QoS Agent (MQA) to measure service quality on standard mobile terminals (i.e., cellular phones) [Soldani 06]. The MQA software library is however restricted to technical service parameters such as service application setup time and Received Signal Code Power (RSCP). Deryckere et al. extend this approach by attempting to also register actual user experience [Deryckere 08]. In particular, they present a modular software tool which can be integrated in end-user terminals and which encompasses three different types of probes (contextual, QoS-related and experience-related). User experience probing is for instance supported by facilitating the post-usage presentation and completion of user surveys and questionnaires on the user's device.

User experience and satisfaction measurement can be considered as a special form of user research, a topic which has a relatively long history in the domain of Human Computer Interaction (HCI). For instance, techniques such as usability testing, user observation, user interviewing and self-reporting have proven to be very effective and efficient methods to evaluate the (graphical) user interfaces of software systems [Nielsen 94]. Whereas these techniques were previously purely paper-and-pencil-based, there have recently been some attempts to computerize them. Several advantages of computerized user research methods over their non-digital predecessors are enumerated in [Barrett 01]. Two notable achievements in the field of digital user research and, in particular, of computerized user experience sampling are the MyExperience and SocioXensor frameworks. The open-source MyExperience software tool for the Windows Mobile platform combines sensing technology and computerized self-report procedures to collect both quantitative and qualitative data regarding human behaviors, attitudes and activities [MyExperience 10]. An important characteristic of the MyExperience framework is its support for in-situ, context-triggered sampling [Froehlich 07]: it allows participants to report on their thoughts, feelings, behavior and so on as they are experienced, this way eliminating recall bias. Additionally, MyExperience takes advantage of the multimedia features of the device on which it is hosted (e.g., audio recording, image and video acquisition) to provide richer forms of participant response. Likewise, SocioXensor is also a software toolkit for in-situ data collection [Mulder 05]. Analogous to MyExperience, SocioXensor recognizes that personal mobile devices such as contemporary cellular phones are excellent means for capturing objective data regarding application usage, for accumulating contextual information and for measuring user experience. The main difference between both solutions is that SocioXensor has a larger focus on interpersonal relations and social phenomena. Note that neither MyExperience nor SocioXensor are intended to capture QoS-related information.

Part II of this thesis will evaluate the impact of the traffic engineering operations performed by the NIProxy on the experience of users of distributed applications. This will be accomplished by presenting the outcomes of numerous experimental studies. All results that will be provided were captured objectively, for instance by tracing the traffic that traversed the transportation network. For each test, the recorded results will be interpreted analytically, after which conclusions will be drawn regarding the implications of the achieved results on user perceived QoE. Notice that this implies that the conclusions will be rather speculative in nature, since no traditional usability research methods (e.g., user surveys) were applied to formally confirm them. Also, although they offer substantial opportunities with regard to low-intrusive user experience sampling, experience measurement software toolkits such as MyExperience or SocioXensor were not exploited. This latter negligence can in part be explained by the relatively recent emergence of these solutions. However, the absence of explicit user feedback and data from formal qualitative studies is justified by the fact that the produced experimental results are of such a magnitude that the beneficial influence on user QoE will be intuitively apparent.

2.3 Context-Aware Networking

The NIProxy enables user QoE optimization by introducing intelligence in the transportation network. This intelligence takes the form of contextual knowledge that is accumulated from a number of different sources. In particular, as will be discussed in section 3.1, the NIProxy's contextual awareness is threefold and encompasses information regarding the transportation network, the distributed application as well as the terminal and preferences of the end-user.

Context is a very extensive concept. This is for instance illustrated in the definition by Abowd and Dey, which state that context "includes any information that characterizes an entity's situation and environment" [Abowd 99]. The same authors define a system to be context-aware in case "it leverages context to provide relevant information or services to the user, where relevancy depends on the user's task". Since communication networks can be considered as systems (e.g., systems for data dissemination), context sensitivity also applies to them. Context-aware networking therefore refers to the exploitation of context during the execution of network-related operations such as routing.

Countless examples of context-sensitive systems are described in the literature. As an example, nearly all of the QoS architectures that have been described in section 2.1.3 as well as the traffic engineering frameworks that will be reviewed in section 2.4 consider at least modest forms of contextual knowledge. Some of these systems merely exploit context to improve the efficiency of their operation. Others leverage context to achieve higher-level objectives that surpass simple efficiency maximization. For instance, the MobiGATE system (see section section 2.4.2) provides an adaptation engine which is steered by accumulated context to guarantee that end-users are provided with content that is tailored to their particular consumption environment.

An exhaustive survey of context-aware systems is not in order in this dissertation. It suffices to know that the NIProxy exploits its contextual awareness as a means to achieve its overall objective of user QoE optimization. Indeed, it is apparent that contextual data plays a crucial role in improving the satisfaction of users of distributed applications. Therefore, all of the NIProxy's QoE optimization tools have access to its complete context repository and can leverage it to coordinate their actions. This approach guarantees that the NIProxy's user QoE optimization decisions will always be in tune with the current context of use.

As a final remark, of the NIProxy's three supported categories of context (i.e., network-, application- and terminal-related), its application awareness is the most unique (i.e., the one that is the least prevalent in related systems). Section 3.1.2 will explain that this type of context encompasses application-specific information and that it is hence highly variable. Generally speaking, by supporting this form of context, the NIProxy enables distributed applications to inform it of any application-related knowledge which they deem relevant. Despite its relatively limited adoption by related systems, numerous examples of the usefulness (and even essentiality) of the availability of application-dependent knowledge during QoE optimization will emerge in part II.

2.4 Network Traffic Engineering

The NIProxy enables IPv4-based networks to impact data dissemination and hence to influence the experience that is provided to users of distributed applications. In particular, the NIProxy's dual network traffic engineering toolset includes bandwidth brokering functionality as well as multimedia service provision and delivery support. This section will first review the related work on both research topics and will subsequently state the NIProxy's contributions in the field of network traffic engineering.

2.4.1 Bandwidth Management

The high-level objective of network traffic shaping (NTS) and, in particular, bandwidth brokering schemes consists of mediating the network bandwidth consumption of distributed applications. The need for such schemes is advocated by the fact that, despite the continuously growing bandwidth capacity of transportation networks, network bandwidth remains a scarce commodity and consequently requires judicious management. As such, bandwidth brokering schemes are typically concerned with distributing a certain amount of available bandwidth over a number of competing network flows. This section will review important recent achievements in this research domain.

Two of the initial explorers of the issue of bandwidth brokering were Floyd and Jacobson. In [Floyd 95], they have described the requirements and objectives of link-sharing in packet-switched networks and they have identified link-sharing enforced at gateway nodes to be an essential mechanism to satisfy the demands of emerging real-time distributed applications. Furthermore, formal guidelines for the implementation of hierarchical link-sharing are presented. In their approach, network traffic which arrives at a gateway is categorized and the resulting traffic classes are subsequently laid out in a hierarchical link-sharing structure. Every class in the hierarchy is allocated its entitled segment of the link bandwidth; in case classes fail to completely consume their assigned bandwidth volume, the proposed guidelines state that the excess bandwidth should be divided among the other classes in accordance with "some set of reasonable rules".

An architecture for the dynamic and fair distribution of network bandwidth called the eXact Bandwidth Distribution Scheme (X-BDS) is presented in [Hnatyshin 06]. The term "exact" refers to the fact that the scheme relies on measured instead of estimated values of flow requirements. The objectives of the X-BDS architecture are threefold: to guarantee that each flow receives at least its minimal requested rate, to distribute network resources (i.e., bandwidth) in a fair fashion among admitted flows and to support dynamic adjustment of flow resource allocation. To attain scalability, the X-BDS system pushes complexity towards the periphery of the network. In particular, whereas edge routers are required to maintain per flow information, core routers only need to deal with flow aggregates. Provided experimental results confirm that X-BDS succeeds in supporting fair per flow distribution of available bandwidth without compromising overall network link utilization.

Rakocevic et al. have presented in [Rakocevic 01] a dynamic bandwidth partitioning scheme in which network links are logically split up into a collection of bandwidth-configurable sublinks that each serve a particular network traffic class. In particular, analogous to the DiffServ standard (see section 2.1.3), the proposed solution defines a number of independent traffic classes such that network traffic belonging to different classes will not impact each other. The scheme dynamically adapts the portion of the link bandwidth that is allocated to each class to improve the overall utilization of the available bandwidth as well as to maximize the utility for the end-user (where utility can be interpreted as a simplified form of QoE). The authors therefore describe their solution as being both adaptive and user-oriented.

Network traffic shaping and bandwidth sharing has also been investigated in the context of the WebTP project. The WebTP research effort attempts to address the complete suite of issues that are introduced by the present-day content transportation model of the World Wide Web. In [Gupta 02], a transport-layer solution for Web transport (i.e., a replacement for HTTP over TCP/IP) is conceptualized which is exclusively receiver-driven in terms of connection setup, flow control, congestion control and retransmission initiation. Simulation results confirm that the proposed protocol achieves stable, efficient and fair bandwidth partitioning among concurrent WebTP flows. The protocol is in addition TCP-compatible, as WebTP traffic is shown to interact fairly with competing TCP flows.

Foster et al. describe in [Foster 04] GARA (General-purpose Architecture for Reservation and Allocation), an extensive resource management architecture which supports secure immediate as well as advance reservation of resources, resource co-reservation (i.e., simultaneously allocating various types of resources such as network bandwidth and CPU time) and online monitoring and control of both individual resources and heterogeneous resource ensembles. A prototype GARA implementation is presented which largely builds on DiffServ mechanisms. The authors prove that GARA can be exploited in in DiffServ-enabled networks to perform bandwidth management as well as to enforce bandwidth guarantees for heterogeneous flow types. In particular, two specific types of network traffic are considered, namely foreground (i.e., latency- and jitter-sensitive) media data and background (i.e., voluminous yet latency-insensitive) bulk transfers.

Another scheme for bandwidth brokering in DiffServ domains is the Active Resource Management (ARM) approach that is described in [Ramanathan 01]. ARM aims to optimize bandwidth utilization by actively and dynamically reallocating bandwidth in function of the current requirements of the applications as well as the status of the DiffServ-enabled network. The ARM approach is motivated by the observation that static resource allocations based on peak bitrate are likely to lead to network under-utilization. Further aggravating matters is the fact that applications which could make good use of the excess resources might suffer from starvation as they are by default unable to adopt the unexploited capacity. Through simulation, the authors demonstrate that ARM might conserve up to 75 percent of the bandwidth capacity of DiffServ networks while still honoring the QoS requirements of the admitted network traffic.

The issue of bandwidth utilization optimization has also been investigated by Furini and Towsley, this time however in the context of the Premium Service (PS) architecture [Furini 01]. The default Bandwidth Allocation Mechanism (BAM) of the PS architecture is based on the peak bitrate of real-time traffic. In contrast, the newly proposed BAM dynamically adjusts the amount of bandwidth that is reserved for real-time flows while preventing their QoS requirements from being violated. As such, the available network bandwidth is exploited more efficiently and completely and hence overall bandwidth utilization is improved.

The generic, application-independent Congestion Manager (CM) framework combines integrated flow management with a convenient programming interface which allows distributed applications to achieve adaptive behavior [Balakrishnan 99][Andersen 00]. As is alluded to by its name, the core focus of the CM system is on the prevention and management of network congestion. The framework however also adopts principles from the Application-Level Framing (ALF) methodology to permit the distributed application to orchestrate its data transmissions and to control, to a certain extent, the fraction of the available bandwidth that is allocated to each of the network flows which it induces.

Anjum and Tassiulas present in [Anjum 99] Balanced Random Early Detection (BRED), a queuing algorithm for Internet gateways which achieves fair bandwidth sharing in the presence of contending adaptive (e.g., TCP) and non-adaptive (e.g., UDP) network traffic. The algorithm penalizes non-adaptive network traffic through packet dropping as soon as it attempts to unfairly claim room in the gateway packet queue (i.e., queue space, and therefore bandwidth, that actually accrues to concurrent adaptive flows).

In [Amir 97], a scalable, lightweight and tunable feedback protocol for reflecting receiver interest back to media sources is presented. The proposed scheme was denominated SCUBA (Scalable ConsensUs-based Bandwidth Allocation) and focuses specifically on multicast-based media conferences. Similar to for instance RTCP, SCUBA embraces the announce/listen metaphor. In this case, periodic receiver reports are exploited to explicitly rank active media sources. By listening for such reports and by aggregating the information that is contained in them, sources are able to adjust their transmission rate. Stated differently, SCUBA enables sources in multicast-based conferencing sessions to account for receiver interest in their rate-adjustment algorithms. The expected outcome is an improved session effectiveness since the available session bandwidth will be allocated in a more efficient and intelligent manner.

An example bandwidth brokering scheme for 3D tele-immersive environments is described by Yang et al. in [Yang 06]. Since such environments incorporate multiple 3D cameras, they require the dissemination of large volumes of video data. The proposed framework leverages the semantic correlation among the generated 3D video flows as well as their exact contribution to the current view to guide stream selection, priority-based bandwidth allocation and content adaptation operations. Presented experimental results demonstrate that the framework yields good video rendering quality in case bandwidth is sufficiently available; more importantly, in the event of bandwidth shortage, the scheme achieves graceful quality degradation by judiciously enforcing bandwidth allocations and by dynamically adapting stream fidelity according to user preferences.

2.4.2 Multimedia Service Provision

The second pillar of the NIProxy's network traffic engineering functionality is multimedia service provision. The NIProxy can assume the role of service provision and delivery platform, which implies that it enables the intra-network application of services on transported (multimedia) flows. Stated differently, service provision and delivery platforms introduce the possibility to process multimedia traffic during its dissemination through the communication network. The type of processing that is offered by services can be very diverse and is in theory only limited by the policy and capabilities of the delivery platform. In particular, services might be absolutely generic, completely application-specific, or anything in between these extremes. This section will representatively sample research efforts with regard to multimedia service provision.

A popular application of the multimedia service provision concept is in-network content adaptation. The typical objective of such services is to improve content delivery to the destination. Initial achievements in this area include the on-demand dynamic distillation approach to cater to network and client heterogeneity [Fox 96][Fox 98] and the InfoPyramid proposal [Mohan 99]. A solution that is targeted specifically at pervasive geospatial content access is presented in [Lin 04a]. Schill et al. propose in [Schill 99] an adaptive multimedia (i.e., image) transfer service for mobile computing environments, whereas Lum and Lau present a context-aware decision engine for content negotiation and adaptation in such environments [Lum 02]. The ZUMA platform for smart home setups incorporates a Universal Content Router (UCR) component to take care of format conversion and content transcoding [Gruenen 06]. The TranSquid network intermediary combines transcoding functionality with a multi-level cache to improve and expedite Web object delivery in heterogeneous client spaces [Maheshwari 02]. Another example of a quality-adaptive proxy cache, this time for layered encoded multimedia data, is the Mocha system [Rejaie 01] in which the quality of cached content is adjusted according to its popularity as well as the bandwidth capacity of interested clients. As a final example, the recently introduced MPEG-21 standard is also concerned with the concept of Universal Multimedia Access (UMA) and therefore specifies tools to assist with the (in-network) adaptation of multimedia content [Vetro 05].

A specific subtopic in the field of intra-network content adaptation is proxy-based video transcoding. Notable examples in this research domain include the application-level video gateway proposed by Amir et al. [Amir 95], the video transcoding gateway for wireless video access presented in [Lei 03], the wireless video transcoding middleware by Aghera et al. [Aghera 03] and the solution that is based on MPEG-21 technology from [Rho 05].

All content transformation systems cited thus far are proxy-based solutions (i.e., the provided functionality is executed by an intermediate network node that is deployed somewhere along the network route from content provider to consumer). In contrast, Chandra et al. consider in [Chandra 00] the application of transcoding technology at the content origin. In particular, the authors propose to leverage transcoding functionality to allow Web servers to customize the size of the content (i.e., images) which constitute Web pages. Experimental results show that the suggested approach provides Web servers a certain amount of control over their upstream bandwidth consumption and allocation, without adding excessive latency and without them having to resort to ad hoc service denials. Furthermore, it is also confirmed that content transcoding support enables sources to implement differentiated service by facilitating heterogeneous treatment of incoming content request.

In the Server-Directed Transcoding (SDT) approach, content adaptation is still performed at a network intermediary but in this case the transcoder is explicitly guided by the origin server [Knutsson 03]. Stated differently, all transcoding operations, while enforced at a proxy, are directed entirely by the content source. The authors argue that traditional proxy-based transcoding breaks the end-to-end model of the World Wide Web because the proxy is unfamiliar with the semantics of the content. The proposed SDT scheme on the other hand preserves end-to-end semantics while at the same time enabling aggressive content transformation. The authors demonstrate that SDT can be integrated into the HTTP protocol through a simple protocol extension and present several examples of valuable server-directed transformations for image content.

Zheng et al. present in [Zheng 06] MobiGATE (Mobile Gateway for the Active deployment of Transport Entities), a mobile middleware framework with extensive attention for service composition and coordination. MobiGATE's main objective consists of providing an environment in which new mobile applications can be easily and rapidly implemented by combining readily available application-level adaptation services as building blocks. Inspired by the separation of concerns principle, a clear division is enforced between the computational activities of individual services and their coordination. The benefit of this approach is that it enables dynamic reconfiguration as well as reusability of adaptation services across applications. As will be illustrated in chapter 5, the NIProxy's multimedia service provision facility exhibits similar features (although, compared to MobiGATE, its support for service composition and coordination is less pronounced and elaborate).

A general-purpose proxy-based communication model for mobile hosts is presented in [Zenel 99]. The model encompasses mechanisms to dynamically download so-called filters (i.e., services) to network intermediaries, to interpose these filters in client/server connections and to control filter operation. Analogous to NIProxy services, filters can exploit application-specific information during their operation and aim to improve the perceived quality of the network by dropping, delaying and/or transforming data that is exchanged between the communicating parties. On the other hand, the proposed filtering model differs from the NIProxy in terms of intended purpose: the filtering approach is primarily concerned with protocol performance optimization, whereas the NIProxy's service provision mechanism pursues the more comprehensive objective of improving end-user experience. As such, there exists a considerable scope difference between both schemes. To demonstrate and evaluate the filtering model, example filters for compressing text contained in HTTP responses, NFS file data compression and TCP throughput optimization in error-prone communication environments are presented.

Like the NIProxy's service provisioning functionality, the AMPS (Active Multimedia Proxy Services) proxy research platform is intended to support a broad, composable and extensible collection of next-generation streaming services [Zhang 04]. An important design goal of AMPS is software modularity. The AMPS codebase is therefore composed of a series of modules which are organized into three separate planes. The service plane provides system-wide services such as resource management, the control plane implements control signaling between the proxy and both the content source and destination, whereas the data plane encompasses so-called stream graph modules which each provide a specialized streaming operation. Another point of particular interest in AMPS is scalability. Therefore, profiling studies were executed to profoundly map the overhead of various system components. The conducted performance analysis has identified the proxy's CPU to be the bottleneck resource in the proposed design.

The Odyssey software platform promotes the establishment of a collaborative partnership between the system and the application [Noble 97]. In particular, Odyssey provides centralized and coordinated support for resource monitoring, notifies applications of relevant changes in resource availability and enforces resource allocation decisions. Adaptation reasoning on the other hand is left completely to the applications themselves. In other words, each application is responsible for independently deciding how best to adapt to modified resource availability when notified. The Odyssey system is specifically targeted at information access applications on mobile hosts.

In section 2.5, the Active Networking paradigm will be discussed and it will become apparent that this methodology holds promising prospects in terms of in-network service provision and delivery. An important economic drawback of this approach however is that it requires substantial modifications to be made to the network infrastructure. Amir et al. therefore suggest in [Amir 98] an alternative to Active Networking which preserves compatibility with present-day networks (i.e., the Internet) by restricting its attention to the deployment of application-level computation in the network (whereas Active Networking also supports service delivery at lower levels of the layered networking model). In particular, their so-called Active Service (AS) framework provides a programmable substrate on top of which application-level services can be built and provided. A comparable approach is taken in [Banka 07], where Banka et al. describe the AWON (Application aWare Overlay Networks) architecture for application-aware service provisioning in overlay networks.

Nahrstedt et al. present in [Liang 05] and [Nahrstedt 05] an integrated service composition framework for pervasive environments. The framework specifically addresses the problem of information retrieval and visualization in such environments. In particular, it allows for the collection, composition and customization of multimedia data and for the subsequent delivery of the composite content to users located in a smart room. Both the users' content interest and the presentational capabilities of the available display devices are hereby taken into account. The framework's organizational form resembles a hourglass as it supports multi-source content input and is capable of disseminating the resulting aggregated content to multiple destinations simultaneously.

Finally, the principle of providing and delivering services has not only manifested itself in a plethora of disjoint research efforts, it has even led to the emergence of a completely new architectural standard, the so-called Service-Oriented Architecture (SOA) model [Barry 03]. In this paradigm, instead of rigidly integrating functionality in applications, business logic and individual functions are modularized and encapsulated as distinct units called services. These services are made accessible over a network and can be leveraged as building blocks to compose complete applications. The defining characteristic of the SOA model is the loosely coupled nature of services: the service interface is independent of its implementation and, as each service is a completely stand-alone entity, no explicit links are a priori defined between them. This feature allows services to be flexibly assembled and even re-assembled, which in turn empowers businesses to rapidly adapt, for example to changing conditions, variable application requirements or heterogeneous user demands. Moreover, the loose coupling facilitates novel functionality introduction as the provision of a new service will not impact already existing services. Similarly, existing services can be upgraded or their implementation can be improved independently from each other; any application which relies on the upgraded service will automatically reap the benefits of the service's improved functionality.

2.4.3 NIProxy Contributions

The NIProxy's bandwidth management functionality is largely inspired by the pioneering research by Floyd and Jacobson on hierarchical link-sharing [Floyd 95]. For instance, the NIProxy's NTS methodology of organizing network flows in a so-called stream hierarchy (see section 4.1) to a great extent corresponds to Floyd and Jacobson's hierarchical class-based bandwidth sharing approach. There are however also substantial differences. In the link-sharing solution, bandwidth management is attained by applying queue management and scheduling at the gateway node. More specifically, individual network packets are dispatched to a collection of queues with unequal scheduling priority. In contrast, the NIProxy does not consider individual network packets but instead operates at a logically higher level. In particular, the NIProxy deals with network flows as a whole and hence grounds its NTS decisions on the instantaneous bandwidth consumption of entire flows. Moreover, Floyd and Jacobson have restricted their discussion to the distribution of the bandwidth capacity of a network link among institutions, protocol families or aggregate traffic types. In other words, their approach concentrates on inter-application bandwidth brokering. The NIProxy extends this research by also addressing intra-application bandwidth mediation. This is motivated by the fact that the majority of modern distributed applications involve multiple network flows with potentially heterogeneous behavior and requirements in terms of bandwidth consumption. The NIProxy's more fine-grained approach enables it to effectively distribute bandwidth among the individual network flows which jointly constitute the network traffic that is induced by a distributed application.

Section 2.4.2 has revealed that multimedia service provision is a very active research domain and that the concept in addition has already been adopted by the industry in the form of the SOA paradigm. Comparing the NIProxy with each individual system is not only practically infeasible, it is also hardly meaningful. After all, although there might be minor functional incompatibilities due to implementational differences and divergent emphases, all service-based systems are grounded on the same foundations and therefore share similar characteristics and advantages. In particular, each such system departs from the idea of encapsulating functionality in stand-alone units that are readily accessible via a computer network. The benefits that are associated with such a loosely coupled design are numerous and for instance include the following:

Composition The possibility to combine multiple (independent) services into a larger whole enables rapid prototyping and even development of distributed application

Scalability Services not necessarily need to be hosted at a single centralized location; instead, they may be scattered over the entire network topology

Extensibility Service provision and delivery platforms often allow their functionality (i.e., their set of supported services) to be extended at run-time

Adaptability Their network-based location makes services very well suited to executing adaptation operations (e.g., of content); as such, service delivery systems might considerably contribute to the adaptability of distributed applications

As will be discussed in detail in chapter 5, the NIProxy's multimedia service provision mechanism exhibits comparable traits and benefits.

Although all service-based systems share a homogeneous vision, it is apparent from section 2.4.2 that research efforts exist which specialize in a particular aspect of the service provision philosophy to distinguish themselves from other approaches. This is not the case for the NIProxy. In particular, the NIProxy aims to offer a general-purpose service provision platform that is as broadly applicable as possible. Stated differently, the objective is not to excel in a certain subtopic but instead to provide a solution that is exploitable by a heterogeneous and extensive set of distributed applications. Furthermore, as is the case with the NIProxy's bandwidth brokering facilities, the focus is not on the service provision functionality by itself, but on its possibilities regarding user QoE optimization.

In summary, the NIProxy does not innovate in terms of the traffic engineering tools which it provides. What does distinguish the NIProxy from other approaches is that it synthesizes these mechanisms in a single system and in addition does so in an interoperable and collaboration-enabled manner. Section 5.2 will discuss that the advantage of this holistic approach is that it affords a myriad of extra possibilities. More importantly, as will be illustrated multiple times in part II of this dissertation, it enables user experience optimization to be risen to a performance level that transcends the results that can be achieved by applying both mechanisms independently. This discussion immediately uncovers another NIProxy hallmark: whereas many related systems focus solely on technical parameters (e.g., QoS provision), the NIProxy exerts its engineering functionality to achieve the higher-level objective of QoE improvement. A final defining feature of the NIProxy's traffic management support is its context sensitivity. Both the network traffic shaping and multimedia service provision mechanisms have access to the NIProxy's complete catalog of contextual information which, as section 2.3 has established, includes network-, terminal- and application-related knowledge. Although context awareness is definitely not a unique trait of the NIProxy, few traffic engineering systems include such a vast and diverse context repository.

2.5 Active Networking

Transportation networks are traditionally allotted a purely passive task. In particular, their responsibility is typically confined to passively transferring data from producer to consumer. The NIProxy however attempts to "activate" the network by directly partaking in the data dissemination process. In particular, by engineering the transported traffic, the NIProxy unlocks the possibility for the networking substrate to actively influence the QoE that is witnessed by users of distributed applications. As such, the NIProxy's methodology bears significant resemblances to the fundamental principles of the Active Network (AN) philosophy [Bush 01]. This paradigm namely also bestows a much more active role on the communication network. Instead of restricting its task to traditional bit hauling, the AN research discipline in particular aims to transform the network infrastructure into a (highly) programmable environment in which the execution of customized computation on transferred data is enabled [Tennenhouse 97]. The AN research was initiated in 1994 by the Defense Advanced Research Projects Agency (DARPA), a R&D agency of the United States Department of Defense which funds the development of new technologies that are potentially useful to the US military [DARPA 10].

2.5.1 Classification

In their position paper, Tennenhouse and Wetherall categorize active networks according to the degree to which transported data and active code are interwoven [Tennenhouse 96]:

Programmable switches This category encompasses *discrete* AN solutions in which the processing of network messages and the injection of customized programs/code are architecturally disjointed. Stated differently, the common data transfer task is completely separated from the programming of the network. As such, network messages do not carry active routines (i.e., code that will be executed by the components which constitute the networking substrate); instead, the installation of active programs on network nodes (i.e., routers) is supported through a discrete mechanism such as an explicit signaling protocol. Each time a network node intercepts a message, it will dynamically apply program(s) on it based solely on information that is embedded in the message header. Discriminating the deployment of programs from message handling is an attractive alternative in case the privilege of network programming should be reserved to the proprietor of the network (i.e., the network operator).

Capsules In this *integrated* methodology, every network message is in fact a complete program that will be executed by the intermediate network components on the path from source to sink. The passive packets of traditional network architectures are in this approach hence replaced by active counterparts, which encapsulate active routines and possibly also actual data. In the AN terminology, such messages are called capsules. As they pass through the network, capsules will be executed at each intermediate network node. Tennenhouse and Wetherall envisage routers to dispatch incoming capsules to a transient execution environment, where they will then be safely evaluated. Through the provision of external methods or built-in primitives, capsules might be provided controlled access to network node resources that are external to the transient environment (e.g., global information and services such as the node's routing table or non-volatile storage).

Both the discrete and integrated approaches enable the network infrastructure (i.e., its constituting components) to perform computation on traversing packets and to potentially modify their contents. For either methodology, this processing is per user or per application customizable. In the programmable switches solution, the packet processing can be customized by the end-user or the distributed application by inserting appropriate information in the packet header (since the packet header will determine which active routine(s) will be applied while the packets are in transit). The capsules philosophy on the other hand affords the end-user or the application complete control over the active programs that will be injected into the network; as such, these parties entirely dictate the in-network computation that will be performed. It is apparent that of both alternatives, the capsules approach represents the most extreme and therefore also the most flexible and powerful application of the AN concept.

2.5.2 Benefits

The AN model allows for the interposition of user- or application-specified computation between end-hosts that are communicating via a transportation network. This principle entails a number of interesting and powerful opportunities [Tennenhouse 96]:

Enabling new applications The possibility to host highly customizable services inside the network is likely to facilitate the development and deployment of new types of distributed applications. Examples in the context of active content caching, user-aware network security and network monitoring and administration are given in [Tennenhouse 97].

Infrastructural innovation Active networking holds the promise of accelerating the pace of infrastructural innovation by decoupling network-based services from the underlying hardware and by allowing new services to be in real-time and on-demand installed in the infrastructure. Stated differently, the AN philosophy enables improvements in respectively the hardware and software domain to be disjointed. Both hardware vendors and service developers can innovate their product individually; the former party will hereby not be hampered by time-consuming software standardization efforts, while the latter will not incur delay caused by vendor consensus achievement.

Generalization In recent years, proprietary systems for enabling in-network computation have proliferated, ranging from advanced firewalls over web proxies with content adaptation functionality to routers with support for nomadic and mobile users. Most systems address a specific issue and additionally do so in an ad hoc, specialized fashion. This implies that there is a lack of a common model, which in turn considerably prohibits the individually developed solutions from interoperating with each other. Tennenhouse and Wetherall claim that this situation will give rise to an amalgam of stand-alone and presumably sub-optimal network-based services. In contrast, the AN philosophy aims to "standardize" the ability to deploy such services and hence to program the operation of the network by providing generic architectural support as well as a common programming platform.

Adaptive protocols Active networking offers the possibility to revolutionize the way people think about transportation networks and communication protocols. In particular, Tennenhouse and Wetherall propose to replace traditional protocol stacks (see section 2.1.1) with tailorable and combinable protocol modules and building blocks to afford application-specific processing. In other words, by applying a programming language perspective to networks and their protocols, a solid foundation is provided for the development of adaptive protocols which will likely improve interaction beyond the possibilities that are currently enabled by the exchange of fixed data formats.

The benefits of active networks can also be outlined from the opposite perspective, namely in terms of the issues and shortcomings of conventional networks which they overcome [Tennenhouse 97]. These encompass the impediment to efficiently integrate new technology and standards into the shared network infrastructure, poor performance caused by the layered protocol model and the resulting introduction of redundant operations at multiple levels of the protocol stack, and the impracticability to accommodate novel innovative services and distributed applications in the existing architectural design.

2.5.3 Achievements

Wetherall et al. present in [Wetherall 99] the ANTS (Active Node Transfer System) concept which encompasses both a general AN architecture and a toolkit. The ANTS architecture focuses on the promise of AN research of achieving interoperability through the definition of a generic programmable network model instead of having to rely on the conventional, lengthy process of standardizing individual communication protocols. The ANTS toolkit is a Java-based prototype implementation of the ANTS architecture that has been realized to experiment with the development of active, adaptive and/or application-specific protocols. In particular, besides providing a run-time environment to enable nodes to participate in an active network, the toolkit entails a protocol programming model which allows users and applications to customize the forwarding of their packets. In an ANTS-based network, the introduction of a new protocol therefore amounts to specifying the active routines that will be executed at the intermediate network nodes.

Like the ANTS toolkit, the protocol boosters research [Marcus 98] aims to stimulate the design of and experimentation with innovative protocols. Marcus et al. present a methodology for optimistic protocol design which is founded on the principle of incremental protocol construction through the dynamic combination of so-called protocol boosters. The twofold objective is to accelerate the evolution of general-purpose protocols as well as to annul their inefficiencies. The former issue is rooted in lengthy standardization processes, while the latter is explained by the fact that general-purpose protocols trade performance for the ability to cater to heterogeneous network environments by minimizing the functionality that is required from the directly underlying layer in the protocol stack. The protocol boosters approach on the other hand allows protocols to optimize their efficiency by adapting (i.e., being re-programmed) to the current network environment (e.g., LAN, WiFi, ...). This methodology allows protocols to be designed assuming the most optimistic case, with additional functionality being incorporated in the protocol in the form of boosters on an as-needed basis. An example booster is presented which enables protocols to cope with the noisy nature of wireless networking technologies (by adding Forward Error Correction protection). Two implementation platforms for the protocol booster methodology are described. In the first platform, protocol boosters are implemented as loadable kernel modules for the GNU/Linux operating system, whereas the second platform is a rapidly reprogrammable FPGA-based hardware prototype. Marcus et al. claim that their methodology can be viewed as an important step toward a fully programmable network infrastructure.

In [Legedza 98], Legedza et al. put forward the proposition that there exists a variety of valuable network services and active protocols which involve in-network processing and discuss their potentially beneficial impact on the end-to-end performance of distributed applications. They ratify their claim by examining the performance of an active protocol which performs intra-network caching of highly dynamic content such as stock quotes. Simulation results highlight that the active caching scheme not only reduces the load on the content servers and the intermediate routers, but also curtails average round-trip hop counts by approximately 18 percent.

The feasibility of AN technology to achieve efficient service customization is investigated by Steenkiste et al. in [Steenkiste 02]. The authors observe that there is a real need for service customization support as it is not uncommon for different users to require slightly dissimilar variants of a particular network-based service. Instead of implementing multiple service instances as a series of independent active programs, a much more elegant methodology is proposed which consists of breaking the service into a baseline component that provides the service's basic (i.e., shared) functionality and a number of customization code modules through which users are able to fine-tune the service to meet their specific requirements. Due to the availability of the baseline component, the customization modules will typically be confined in complexity as well as size. According to the terminology from section 2.5.1, the base component is combined with a screened run-time substrate to form an execution environment, whereas each customization code module can be regarded as an active program. The proposed approach is illustrated through the presentation of three example customizable network services, of which the most notable enables users/applications to tweak the QoS provision behavior of routers according to their potentially heterogeneous QoS requirements.

Lyijynen et al. present in [Lyijynen 03] the Lightning Active Node Engine (LANE) active network platform and apply it to address the content adaptation requirements that stem from user terminal diversity. The LANE architecture encompasses two distinct types of active network entities, the

Active Server (AS) and Active Router (AR). Resource-intensive and computationally complex tasks (i.e., active applications) are always executed by AS components, while the AR components are bestowed exclusively with packet routing responsibilities. Both entity types are implemented in Java. The content conversion case study is implemented as an active application and the authors identify that the AN approach allows the service to be freely relocated inside the network. This in turn offers diverse optimization options as it enables the service's deployment location to be optimized with regard to currently prevailing conditions.

The Network Processors Group (NPG) of the Georgia Institute of Technology (Georgia Tech) adopts the vision of the network infrastructure being an active and programmable computational entity which executes certain classes of computations that would otherwise have to be allocated to general-purpose end-host CPUs [GT NPG 10]. In [Gavrilovska 05], they explore the potential of exploiting heterogeneous multi-core systems which include specialized communication support (e.g., in the form of network processors) as efficient and flexible execution platforms for distributed streaming applications. In particular, they propose to interconnect such multi-core platforms into a network overlay on top of which stream manipulation services such as content transcoding can be deployed. Individual stream processing actions are hereby dynamically mapped to those platform resources that are best suited for them. As an example, communication cores or specialized communication hardware will as much as possible be dedicated to the processing of communication-related tasks, since these platform components are optimized for this type of operations. Experimental results confirm a performance improvement for distributed streaming applications. Since the proposed overlay solution includes programmable network hardware, it can be considered as a modest attempt at active networking.

Lefèvre et al. have explored the possibility to synthesize active networking technology and grid computing research. In [Lefèvre 01], they present the Active Grid (A-Grid) architecture which exploits the Tamanoir framework, a prototype active network implementation [Gelas 00], to support the communication requirements of grid environments and applications. Specific attention is hereby given to the provision of reliable multicast support as well as QoS management for data streams in grid computing environments via active network services.

2.5.4 Discussion and Comparison with the NIProxy

Although there is definitely substantial common ground between AN research and the NIProxy, it is not really feasible to compare both approaches. Active networking represents a completely novel paradigm which radically abandons the premises of current-day networks and completely revises their task and responsibilities. Transforming the networking substrate into a programmable environment definitely yields promising and powerful options in terms of, for instance, network-based traffic engineering, traffic adaptation and service provision. Consequently, the AN methodology provides ample opportunities with regard to user QoE optimization.

On the downside, as it is a revolutionary networking mindset that has only fairly recently emerged, the AN technology lacks maturity and still suffers from a considerable number of open challenges. An important issue is for example the efficient encoding of active routines. Recall that in the capsules approach each network packet corresponds to an active program. As a result, active routines will need to be efficiently encoded to keep the size of these capsules acceptable and to curtail their bandwidth requirements. Additionally, should active programs be binary encoded or, conversely, carried as source code written in a particular programming language? Binary representations will yield maximal performance, however at the expense of portability. The latter alternative on the other hand is a platform-independent solution, yet the source code will need to be compiled at each active networking node (which is a time-consuming process). Another open problem is uniform naming and addressing of network node resources such as the CPU or persistent storage. To achieve interoperability, active applications require a shared understanding as to which resources are available on the node on which they are hosted as well as how these resources are named. Since node resources are potentially shared by concurrent active applications, a fairness model will also need to be developed to prevent an active application from monopolizing them. A final and related challenge is security. Which node resources should be available to which active applications and to which extent? This will at least require some form of user authentication and authorization constructs.

Another very important disadvantage of the AN philosophy is its incompatibility with the current network infrastructure. Active networking demands part of the network hardware to be replaced with components that are capable of executing active applications. As was already discussed in section 2.1.3, historical evidence strongly counts against such a requirement due to the serious financial repercussions which it entails. In particular, economical considerations have prohibited technologies with similar requirements, like for instance the IntServ and DiffServ QoS architectures, from obtaining wide-scale adoption. Notice that since an active network will typically require a vast number of active nodes to unveil its full potential, the necessary financial investments are even likely to exceed those that are required for an IntServ or DiffServ deployment.

A final observation which pleads against the AN approach is the limited attention which it currently seems to be receiving. Whereas the AN research initially flourished under impulse of the DARPA funding, the interest in the topic appears to have rapidly dwindled. This is for instance exemplified by the just described open challenges: a lot of these issues were already identified in the position paper by Tennenhouse and Wetherall [Tennenhouse 96], but to date remain unsolved. The decrease in activity may imply that researchers (and research sponsors) are ever less convinced of the potential and/or economical feasibility of the AN paradigm.

In contrast, the NIProxy does not break the current networking model, nor is it a radically novel concept. Conversely, the NIProxy represents a pragmatic solution to user QoE optimization that is readily utilizable in existing IP-based networks and, in particular, the Internet. The NIProxy does not match the active networking approach in terms of traffic engineering options and the thereby enabled QoE optimization possibilities. As an example, performing operations and optimizations at the network or transport layer of the protocol stack will typically be much less straightforward to achieve (if even possible at all). This is explained by the larger scope of the active networking paradigm: the NIProxy was never intended to be a generic architecture which enables the operation of the network to be completely programmable. Instead, it aims to be a cost-effective solution for user QoE optimization with an exclusive focus on application-layer techniques. Finally, with regard to economic considerations, notice that the NIProxy's functionality does not come for free since the networking substrate will need to be extended with NIProxy instances. It will however commonly suffice to insert instances at a limited number of strategic locations within the network topology (see section 3.5); as a result, the required financial investment will normally only be marginal.

2.6 A Taxonomy of QoS/QoE Frameworks

The literature review that has been presented thus far has uncovered a number of divergent criteria according to which systems that are concerned with QoS and/or QoE provision can be categorized. To conclude this chapter, the most relevant of these possible classification axes will be recapitulated and the NIProxy will be situated on each of them.

A first significant classification criterion is the level of the layered networking model at which the framework conceptually operates (see section 2.1.1). The IntServ and DiffServ architectures that have been described in section 2.1.3 are examples of low-level (i.e., network-layer) solutions. The NIProxy on the other hand operates exclusively at the application layer, the topmost echelon of the layered networking model. This implies that the NIProxy approaches the problem of user QoE optimization purely from the perspective of the distributed application itself and that it does not rely on low-level, potentially specialized constructs during its operation.

Secondly, QoS/QoE frameworks can be orthogonally categorized on the basis of their compatibility with prevailing networking technology and, in particular, the Internet. The discussion in this chapter has indicated that, broadly speaking, two methodologies exist to address the lack of QoS/QoE provision features in present-day transportation networks. The first approach consists of designing and deploying a new generation of networks with adequate, built-in QoS/QoE optimization support, without worrying about compliance with the current Internet architecture and protocols during the process. Dovrolis refers to this approach as the *clean-slate* paradigm [Dovrolis 08]. Example solutions in this category are the IntServ and DiffServ platforms (see section 2.1.3) and systems that adhere to the Active Networking model (see section 2.5). The second methodology proposes to maintain current networks and to extend them with novel functionality where needed. An important objective of this class of *evolutionary* solutions is hence not to break the network architecture that is in place today. Since the NIProxy strives for complete compliance with the Internet, it belongs to this latter category. The two methodologies have opposite advantages and drawbacks [Dovrolis 08]. In particular, while a clean-slate design is likely to yield a completely optimized and highly efficient solution, it suffers from cost-efficiency issues since it discards not only the tremendous financial investments that have been made over the last decades in terms of Internet infrastructure, but also the practical experience and empirical know-how that have been accumulated through years of extensive usage. In contrast, since evolutionary solutions largely reuse the existing Internet hardware and know-how, they are characterized by a minimal time-to-market and modest deployment costs; the disadvantage of their compatibility on the other hand is that it limits their operational possibilities and that it will likely prevent them from achieving optimal results. Which of these methodologies represents the most suitable approach to introduce worldwide QoS/QoE provision support is a highly subjective question that largely depends on the mindset of the practitioner. Empirical evidence however largely points in the direction of the evolutionary approach: despite the relative maturity of a number of proposed clean-slate technologies such as IntServ and DiffServ, they have seen far from universal adoption due to the huge financial commitments which they require.

A final interesting measure for classification is deployment location. In particular, systems for QoS/QoE optimization can be characterized as being either end-system or in-network located solutions. End-system frameworks need to be integrated in the distributed application itself (commonly the client software but sometimes also the server software). This is typically the most straightforward way to support QoS/QoE provision. On the other hand, end-system approaches are likely to require considerable effort from the application developer. Concrete examples of end-system solutions include the Congestion Manager framework that has been described in section 2.4.1 and the server-side transcoding approach proposed by Chandra et al. (see section 2.4.2). The second category consists of solutions that are deployed inside the network (e.g., hosted on network proxies). The NIProxy obviously falls under this category. An attractive feature of such solutions is their reusability: contrary to end-host located approaches, the functionality of in-network solutions can usually be readily leveraged by multiple distributed applications simultaneously. This is especially important for commercial products since high applicability maximizes potential revenue. An additional advantage of network-based solutions over end-system alternatives emerges in multi-application environments. A user might be running multiple distributed applications concurrently. If in this case the QoS/QoE support is embedded in the applications themselves, correct interoperation between the different approaches might not be guaranteed, which in turn will yield outcomes that are suboptimal at best. In contrast, an in-network framework is in such situations capable of maintaining a global picture and will hence allow for the generation of optimized results. As a final remark, the class of network-based systems can be further subdivided into completely generic and transparent solutions on the one hand and hybrid solutions which demand (modest) modifications to the software of the distributed application on the other. In the latter case, the necessary end-host software modifications will typically be confined to interfacing concerns, while the bulk of the actual QoS/QoE functionality will still be located inside the transportation network. As will become apparent in section 3.4, the NIProxy is an example of such a hybrid solution.

Part I

Network Intelligence Proxy

Overview

This part forms the “theoretical” portion of the dissertation as it will profoundly discuss the Network Intelligence Proxy. In particular, this part aims to familiarize the reader with the NIProxy and its approach to QoE optimization. This will be done in a top-down fashion. As a result, chapter 3 will first provide an overview of the NIProxy’s fundamentals and intended purposes. In addition, it will outline the methodology which the NIProxy adheres to to achieve its objectives. Technical details have been deliberately omitted from this initial chapter since it is specifically intended to serve as an entry point into the biotope of the NIProxy. The two subsequent chapters are dedicated to the traffic engineering techniques that are incorporated in the NIProxy and that enable it to influence the QoE of users of distributed applications. More specifically, chapter 4 will describe the NIProxy’s bandwidth brokering and network traffic shaping functionality, whereas chapter 5 will dilate on its support for hosting services and for applying them to transiting network traffic. The rationale behind both techniques will be presented, as well as crucial decisions regarding their design and implementation. Both chapters will also present a representative example to further clarify the operation of the described traffic engineering mechanism. Chapters 4 and 5 hence already encompass a certain amount of technical information, but still primarily provide relatively high-level descriptions. All implementation-related information is namely clustered in chapter 6, the final chapter in this part, which will discuss the design of the NIProxy’s software architecture and which will describe the low-level details of its implementation.

Chapter 3

Objectives and Methodology

The Network Intelligence Proxy (NIProxy) is a context-aware proxy server or network intermediary. Like any proxy, it needs to be included in the network topology, where it needs to be interposed in the network path between sources and destinations. The objective of the NIProxy can be summarized as maximizing the satisfaction or QoE of users of distributed applications. The NIProxy attempts to achieve this objective by outfitting IP-based transportation networks with traffic engineering tools to enable them to actively control and possibly adapt the network communication behavior of distributed applications. Wherever possible, the NIProxy's traffic management decisions are coordinated by its accumulated contextual knowledge.

3.1 Context Introduction in the Network

As alluded to by its name, the NIProxy's methodology is centered around the introduction of "intelligence" or "awareness" in the networking infrastructure. To fill up its intelligence repository, the NIProxy actively collects contextual information that can be categorized into three distinct groups.

3.1.1 Network Awareness

The first context source that is queried by the NIProxy is the transportation network and the contextual knowledge in this case takes the form of quantitative network-related measurements and statistics. To obtain this type of data, the NIProxy encompasses an active network probing framework which enables estimation of the throughput, latency and error characteristics of communication links [Wijnants 05b]. The framework is largely inspired by the work by Wolski [Wolski 97b]. As is illustrated in Figure 3.1, link latency is measured by recording the time that passes between transmitting an arbitrarily small probe and receiving an acknowledgment for it. This amount of time actually equals the round-trip time; dividing it by two yields an estimate of the link latency (assuming a symmetrical delay in both directions). To determine the throughput of a network link on the other hand, the framework times how long it takes to emit a probe packet of significant size (e.g., 1000 bytes) and to receive an acknowledgment for it. The link bandwidth capacity can subsequently be calculated by dividing the used probe size by the recorded time minus the measured round-trip time. In mathematical notation, using the terminology from Figure 3.1:

$$\text{estimated throughput} = \frac{\text{data size}}{\text{data transfer time} - \text{estimated RTT}} \tag{3.1}$$

Finally, for each monitored network link, the framework also stores the total number of transmitted probes and the number of probes that were not acknowledged before a predefined time-out interval (meaning either the probe got lost on the link, or the acknowledgment did). These figures provide an insight in the error characteristics and, in particular, the packet loss rate of the communication link.

The network sensing framework is exploited to periodically probe the network links which connect the NIProxy to the end-points of the distributed applications which it is currently servicing. The outcome are measurements which provide the NIProxy with a snapshot of the state of network connections at a given moment in time. It is possible for this state to fluctuate considerably over time. Also, the produced link measurements might not always be completely accurate. The network probing framework therefore incorporates a prediction engine. In particular, each time a certain network connection was successfully probed, the results are combined with the measurements obtained during previous probing iterations to forecast the future state of the connection. The prediction engine employs mean- and median-based predictors for respectively forecasting future link throughput and latency [Wolski 97a].

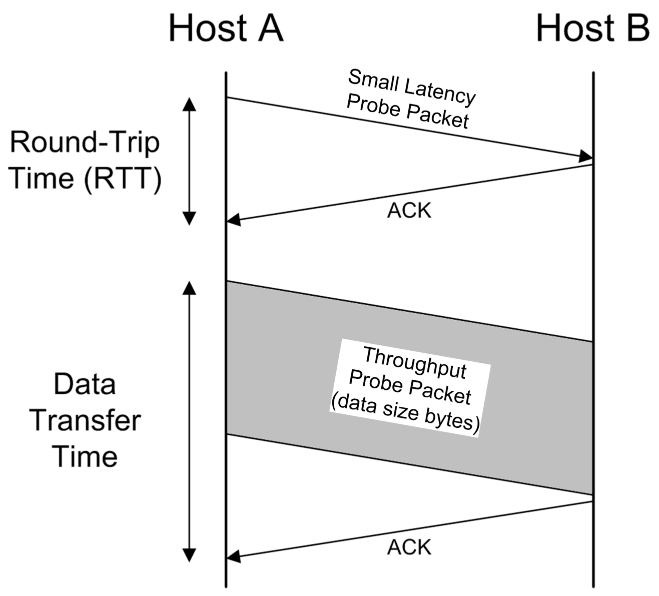

Host A
Host B
Small Latency Probe Packet
Round-Trip Time (RTT)
ACK
Throughput Probe Packet (data size bytes)
Data Transfer Time
ACK

Figure 3.1: Link latency and throughput determination using active probing.

It is apparent that the NIProxy's incorporated network probing framework is relatively rudimentary and that it is mostly suitable for estimating the performance of singular network links and wired connections which encompass only of a restricted number of intermediate hops. Accurate estimation of the capacity of multi-hop routes or complex network connection types like for instance wireless channels is likely to require more advanced network probing techniques. It is important to note however that such techniques (e.g, the pathvar tool by Jain and Dovrolis [Jain 05]) could readily be integrated in the NIProxy. Furthermore, it is not a requisite that the network-related measurements are computed by the NIProxy itself. This information could for instance just as well be derived from network simulation outcomes or it could be obtained by contacting external entities that are specialized in network performance estimation (e.g., the Spirent SmartBits hardware [Spirent 10]). To summarize, although the currently implemented network sensing framework might prove unfit to accurately estimate network channel performance, it is nonetheless appropriate to assume that the NIProxy disposes of adequate network-related measurements and information.

Finally, besides awareness of the status of the transportation network and prevailing channel conditions, the NIProxy's network awareness also comprises knowledge of the bandwidth requirements of network flows. To acquire this knowledge, the NIProxy simply monitors the network traffic that passes through it and records its bandwidth consumption.

3.1.2 Application Awareness

The second type of context which is amassed by the NIProxy consists of information regarding the serviced distributed applications. The context source in this case is the distributed application itself. Applications can provide the NIProxy with any application-related knowledge which they deem relevant or that may be valuable with regard to the QoE optimization process. As a result, the NIProxy's application-related context might vary considerably depending on the kind of distributed application under consideration. As an example, due to their radically divergent requirements and characteristics, the application context for respectively a multi-player computer game and a video conferencing application will most likely be completely different. As a general rule however, the more context the distributed application forwards, the more information the NIProxy will have at its disposal to direct its operations and hence the higher the probability of successful QoE optimization will be.

One possible example of knowledge which could contribute to the NIProxy's application context is the relationship, in terms of importance or significance, that exists between the network traffic that is generated by a distributed application. All but the simplest distributed applications typically require the simultaneous dissemination of heterogeneous types of data (e.g., control messages, voice data, audio and/or video feeds, file data, etcetera). These different traffic classes will typically not be equally crucial for the correct functioning of the application. For instance, the loss of control data is likely to have a largely detrimental impact on the operation of the application and might even lead to undefined behavior. Failure to receive a number of voice or video packets will on the other hand probably be less problematic. It might hence be wise for the application to inform the NIProxy of this inequality in the importance of network traffic types so that it can be taken into consideration during QoE optimization. As an example, based on this knowledge the NIProxy could decide to give the transmission of control traffic precedence over voice or video dissemination. Besides importance disparity between network traffic classes, intra-class significance differences are also likely to occur. In other words, the influence on user QoE of individual network flows within one particular network traffic class might also vary considerably. Consider a video conferencing application as an example. The relevance of individual video streams, which in this case each represent a certain remote participant, will probably fluctuate throughout the video conferencing session (e.g., a participant who was previously passively attending the conference now starts to actively partake in the current discussion). Also relaying such intra-class flow importance information enables the NIProxy to take deliberate per flow decisions.

3.1.3 Terminal Awareness And User Preferences

The NIProxy's final context category concerns knowledge of the end-user terminal. In the early days of computer networking, the devices employed by users to connect to networks and to run distributed applications uniformly consisted of desktop computers which all exhibited comparable capabilities and constraints. Due to the popularization of handheld devices and users' rising interest in mobile network access, we are now however living in an era of ubiquitous and pervasive computing. A direct consequence of this evolution is the loss of uniformity in the end-user device space. The range of network-capable devices grows larger every day, each having specific features, characteristics, limitations, hardware support and so on. It seems appropriate for a user experience optimization framework like the NIProxy to be able to react to such heterogeneity. Terminal awareness for instance allows for the delivery of content that is tailored to the capabilities of the device from which it was requested. As an example, due to their limited screen size, handheld devices need to spatially downscale high-resolution images or video fragments before such content can be displayed. Since the bandwidth requirements and processing overhead (caused by, for instance, decoding) is directly proportional to the resolution of the content, a more resource-efficient strategy would be to enforce resolution reduction prior to delivering the content to the handheld device.

Several solutions have been developed for the representation of terminal capabilities. The majority of the early approaches were based on proprietary formats. The main drawback of such proprietary specifications is their incompatibility. Distributed applications which employ different formats will not be able to exchange terminal information with each other. As over the years the number of proprietary solutions proliferated, the need for consensus became ever more apparent and led to the development of two standardized specifications. The first standard, Composite Capability/Preference Profiles (CC/PP) [Woodrow 04], is maintained by the World Wide Web Consortium (W3C), the main international standards organization for the World Wide Web (WWW). Unsurprisingly, the CC/PP standard concentrates on WWW issues and in particular on enabling handheld devices such as cell phones and PDAs to efficiently and effectively access Web content [Gimson 03]. The Usage Environment Description (UED) standard [ISO/IEC 04] on the other hand has been developed by the Moving Picture Experts Group (MPEG) as part of their MPEG-21 multimedia framework [ISO/IEC 02]. CC/PP and MPEG-21 UED are XML-based and both also support, besides terminal characteristics, the specification of user preferences.

Of both standards, MPEG-21 UED was selected for terminal awareness acquisition in the NIProxy. This decision is motivated by the standard's considerable focus on multimedia. As improving the multimedia handling capabilities of transportation networks forms an important aspect of the NIProxy's user QoE optimization objective, MPEG-21 UED provides exactly the features which the NIProxy requires from a language for terminal capabilities specification.

MPEG-21 UED divides the description of the usage environment into the following four dimensions [Vetro 05]:

- The **terminal capabilities** category describes the end-user device itself and includes information regarding
  - hardware properties; e.g., processor speed, storage characteristics and memory capacity
  - software properties; e.g., information concerning the operating system and media format support
  - display capabilities; e.g., screen resolution and supported refresh rates
  - user interaction support; e.g., the presence of a microphone or a pointing device
  - power characteristics; e.g., remaining battery lifetime
- Knowledge of the network connection of the device is grouped in the **network characteristics** category; elements are provided to describe not only static network properties like, for instance, the theoretical throughput of the connection, but also to communicate the current condition of the network connection, which can fluctuate dynamically over time
- The **user characteristics** class enables the representation of the user and his preferences; it encompasses elements to denote
  - the user itself; e.g., user name and e-mail address
  - possible auditory or visual impairments of the user; e.g., deafness or color vision deficiency

  - the user's physical location and mobility characteristics
  - content, presentation and modality conversion preferences; e.g., to which type of content does the user give preference and exactly how does the user want this content to be presented on his terminal?

- The **natural environment characteristics** pertain to the natural and physical environmental conditions that surround the user like, for instance, lighting properties and background noise level

An example MPEG-21 UED document, which specifies characteristics and properties from each of the four usage environment description categories, is provided in appendix B.

Important to notice is that the NIProxy does not define how terminal information and user preferences should actually be determined. Some of the terminal's properties could for example automatically be detected by executing hardware analysis tools on the end-user device. Other information might be explicitly provided by the user, for instance through a Graphical User Interface (GUI). The only requirement that is imposed by the NIProxy is that the terminal and user preferences data is provided in the form of a valid MPEG-21 UED profile.

Support for terminal and user awareness has only fairly recently been incorporated in the NIProxy. The suitability of the MPEG-21 UED standard for representing this context category and the validity of the followed approach have both been confirmed by means of small-scale pilot studies in dedicated test environments [Creemers 09]. However, the added value and true potential of terminal and user awareness with regard to QoE optimization has yet to be determined. In particular, involving it in more extensive experiments and realistic environments represents an interesting direction for future research. Notice that this implicates that all the experimental results that will be presented in part II of this dissertation have been achieved without considering, let alone exploiting, this particular type of context.

3.2 Traffic Management Techniques

Gathering contextual information by itself of course does not enhance the user experience, it is merely a means which enables it. The NIProxy therefore exerts its contextual awareness to improve the multimedia traffic handling capabilities of IP-based networks and hence to enable them to influence user satisfaction. This is achieved by furnishing the transportation network with two complementary traffic engineering facilities. The provided techniques

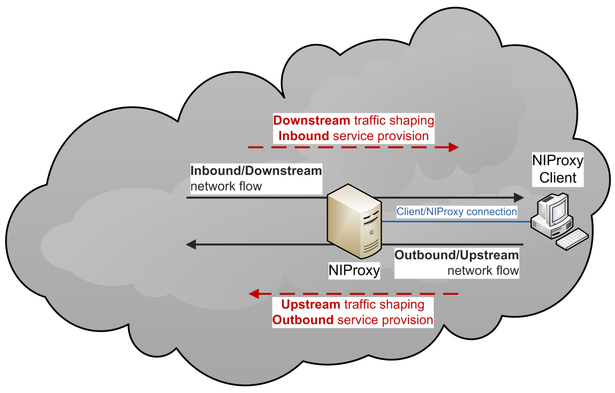

Downstream traffic shaping
Inbound service provision
Inbound/Downstream network flow
NIProxy Client
Client/NIProxy connection
NIProxy
Outbound/Upstream network flow
Upstream traffic shaping
Outbound service provision

Figure 3.2: Inbound versus outbound traffic engineering.

are network traffic shaping and multimedia service provision and will be profoundly discussed in chapters 4 and 5, respectively. In summary, the network traffic shaping functionality enables the coordination and orchestration of the bandwidth consumption of the network traffic that is induced by distributed applications; its multimedia service provision framework on the other hand allows the NIProxy to act as a service delivery platform and as such unlocks the possibility to process the (multimedia) data that flows through the network in a theoretically boundless manner.

As section 5.2 will describe in more detail, its dual traffic engineering facilities are not incorporated in the NIProxy as independent elements; instead, an integrated design has been adopted which enables both techniques to supplement each other and to cooperate during QoE optimization. This approach distinguishes the NIProxy from many related frameworks (see the findings from the literature review in section 2.4 for corroboration) and forms an important contribution to the field of QoE optimization research. In particular, the experimental results which will be presented in part II of this dissertation will confirm that this symbiotic solution yields powerful extra possibilities in terms of QoE optimization that are not available in case both mechanisms are applied in isolation from each other.

3.3 Inbound and Outbound Optimization

The NIProxy's dual traffic engineering techniques can be applied in the *inbound* as well as *outbound* flow direction. As Figure 3.2 clarifies, these terms are defined from the viewpoint of a NIProxy-managed client:

Inbound Network data that is destined for a NIProxy client is said to follow the inbound flow direction. Stated differently, inbound network traffic is emitted by a certain source, intercepted by the NIProxy and then (possibly) forwarded to one or more hosts which it is currently servicing. Possible synonyms for the inbound keyword are "incoming" and "downstream".

Outbound Outbound network traffic is defined as the collection of network flows for which a NIProxy client acts as source. In other words, the outbound keyword is employed to denote network flows which originate from hosts that are connected to a NIProxy instance. Possible synonyms include "outgoing" and "upstream".

Notice that situations may arise in which two hosts which are both managed by a NIProxy instance (conceivably even the same instance) are engaged in a communication session. In such a scenario, the data that is exchanged between both hosts will conceptually follow the inbound flow direction from the perspective of one host, while the other will regard it as outbound traffic (and vice versa in the case of bidirectional data transmission).

For inbound network flows, the goal of the NIProxy's network traffic shaping mechanism exists of regulating and optimizing their downstream delivery to the NIProxy client. In contrast, in the outbound flow direction, the upstream bandwidth consumption of the network traffic which is transmitted by the NIProxy client itself will be managed. Regarding the multimedia service provision platform, inbound services will be applied to data that is destined for the NIProxy client, whereas outbound services will process network flows which originate from such a client.

Of the two supported flow directions, inbound network traffic shaping and multimedia service provision is likely to have the largest and most immediate impact on the QoE that is experienced by the NIProxy client. The majority of the discussions in part I and the results in part II of this doctoral thesis will therefore focus on the inbound course. An exception to this rule is chapter 10, which is specifically devoted to outbound traffic engineering and which will exemplify that outbound bandwidth brokering and service delivery also have a number of meaningful applications that might, most plausibly indirectly, influence user QoE.

3.4 Network Intelligence Layer

As could have already been inferred from the discussion thus far, the NIProxy is, contrary to a typical web proxy, not a transparent network intermediary.

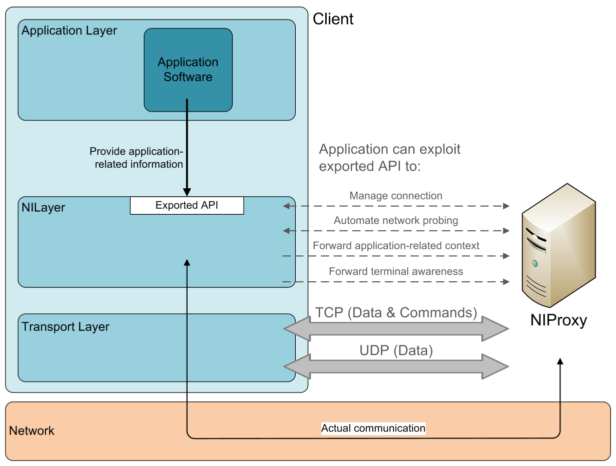


Figure 3.3: The Network Intelligence Layer (NILayer) support library.

Clients of distributed applications need to explicitly connect to and communicate with a NIProxy instance before they can take advantage of its QoE optimization features. The communication between the NIProxy and the distributed application in addition needs to adhere to a proprietary protocol, the so-called Network Intelligence (NI) protocol.

The fact that the NIProxy is not transparent implies that the software of the distributed application will need to adapted. To streamline this process and to minimize the amount of required modification effort, a generic support library called the Network Intelligence Layer (NILayer) has been developed for inclusion in the application software. As Figure 3.3 illustrates, the support library conceptually positions itself between the application and transport layers of the TCP/IP protocol stack and implements all low-level aspects related to client-NIProxy communication. In particular, the NILayer implements the NI protocol at client-side and exports an API for

- initiating connection setup and executing the log in procedure
- supplying the NIProxy with application-related context
- serializing and propagating MPEG-21 UED profiles containing terminal characteristics and user preferences data
- constructing and managing the stream hierarchy maintained by the NIProxy (see section 4.7)

- registering and unregistering application-layer protocols (see section 6.2.3)

In addition, the support library automates the process of responding to network probes issued by the NIProxy. The NILayer and its exported API hence for a large part relieve the distributed application from the responsibilities which the NIProxy imposes on it.

In the connection setup phase, the NILayer first establishes a Transmission Control Protocol (TCP) connection between the application software and the NIProxy. On success, the TCP channel will be used to implement the NI protocol, starting with the log in procedure. In other words, all NI protocol commands will always be transmitted over this connection. This approach is motivated by TCP's reliability and ordered delivery features [Postel 81b]; these are very interesting assets for this type of communication since failed and, to a lesser extent, out-of-order protocol command delivery will typically be unacceptable. Once the log in procedure has completed successfully, the NILayer supplements the TCP link with a connection of the User Datagram Protocol (UDP) type. UDP provides none of TCP's communication guarantees, which makes it a lightweight transport-layer protocol that is very suitable for the transmission of time-sensitive data [Postel 80]. The exchange of effective data (i.e., actual data as opposed to NI protocol commands) between the distributed application and the NIProxy can be carried out over either the TCP or UDP channel, depending on its characteristics and requirements. Delay-insensitive yet loss-intolerant data like, for instance, an image is likely to benefit from the transportation guarantees that are offered by the TCP connection. UDP on the other hand is presumably the recommended candidate for the propagation of time-critical information and for real-time streaming purposes. In either case, data dissemination between the distributed application and the NIProxy will always occur in a unicast fashion, irrespective of the network communication strategy that is adhered to by the distributed application itself (see section 6.2.3 for more information).

The NILayer support library has been designed for reusability and maximal applicability. It has deliberately been kept as generic as possible to ensure that it can be incorporated in a wide variety of distributed applications. Consequently, any application-specific functionality and knowledge has been barred from the support library; instead, it only provides general functionality that is potentially useful to a multitude of distributed applications. Note that this implies that the NILayer *facilitates* the NIProxy-related tasks that are imposed on the distributed application rather than completely eliminating them.

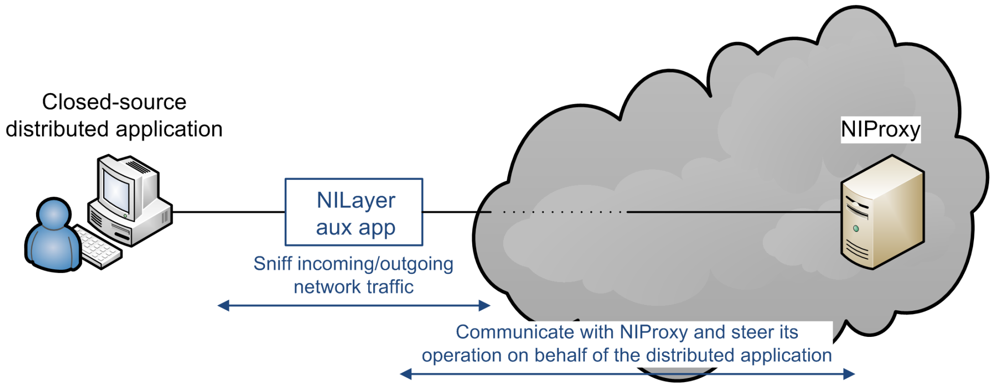

Closed-source distributed application
NILayer aux app
Sniff incoming/outgoing network traffic
NIProxy
Communicate with NIProxy and steer its operation on behalf of the distributed application

Figure 3.4: Introducing an intermediate NILayer auxiliary application to enable closed-source application to benefit from NIProxy functionality.

Modifying distributed applications might somtimes prove problematic or even completely impossible, for instance in the case of legacy or closed-source software. A possible solution to this problem is to exploit the NILayer to implement a separate auxiliary application which is subsequently deployed in close proximity to the application end-point (i.e., ideally it would be deployed on the device on which the distributed application is running). As is depicted in Figure 3.4, the NILayer auxiliary application will take care of the communication with the NIProxy on behalf of the distributed application, which hence will not require modification. For instance, since the auxiliary application is topologically located very near to the end-point of the distributed application, the network awareness that is obtained by probing the auxiliary application will for a large part also apply to the distributed application itself. As another example, by sniffing and interpreting the network traffic that originates from and is destined for the application end-point, the auxiliary application might be able to supply the NIProxy with (a limited degree of) application-related context. It is apparent however that this strategy should only be used as a last resort since it will normally not allow the distributed application to reap the full potential of the NIProxy. This will in turn yield QoE optimization results that are suboptimal at best. Stated differently, whenever possible, direct integration of the NILayer support library in the distributed application software is largely advocated.

3.5 Deployment

There are theoretically no constraints on the location where the NIProxy can be incorporated in IP-based transportation networks. Not all locations in the network topology are however equally fit for NIProxy deployment. To maximize its QoE optimization potential, the NIProxy should be deployed at (or close to) junction points where network performance alters significantly. Doing so will enable the NIProxy to mitigate the mismatch in network performance that exists at these locations by managing and possibly adapting network traffic before it reaches the lesser performant part of the network. Stated differently, at such junction points, the NIProxy can exert its traffic engineering functionality to guarantee that the resource consumption of the distributed application conforms to the limitations of the remainder of the network connection and, more importantly, to make deliberate decisions about the exploitation of the capacity that is actually available so that the user's QoE is maximized.

An example of a promising location for NIProxy deployment in a Wide Area Network (WAN) context is the conceptual boundary between the core of the network and the access network. Whereas the network core will usually be sufficiently capacitated to transport all network traffic that is introduced by the distributed application, the same will not necessarily be true for the access network. In a setting involving residential users for instance, this boundary manifests itself at the ISP gateway where the user's last mile connection is linked to the ISP backbone (e.g., at DSLAM level in xDSL-based access networks). Analogously, another suitable location to incorporate the NIProxy is at the transition from a wired to a wireless network. As a result, it might be interesting to coincide the NIProxy with a wireless access point.

No restrictions apply to the number of NIProxy instances that can be simultaneously deployed in one particular communication network. Especially for large-scale networks (e.g., the Internet) it would make sense to incorporate multiple NIProxy instances. By scattering NIProxy instances over the network topology, it becomes possible for users at geographically dispersed locations to concurrently benefit from the NIProxy's QoE optimization operations. Users could in this case connect to one of the NIProxy instances that are deployed nearby in the network topology so that a bounded client-to-proxy hop count and delay is guaranteed. Another possible motive for multiple NIProxy instance deployment is load balancing and scalability. The QoE optimization operations that are performed by the NIProxy might require significant amounts of computational power. This is particularly true for the NIProxy's multimedia service provision mechanism since services might be espe-

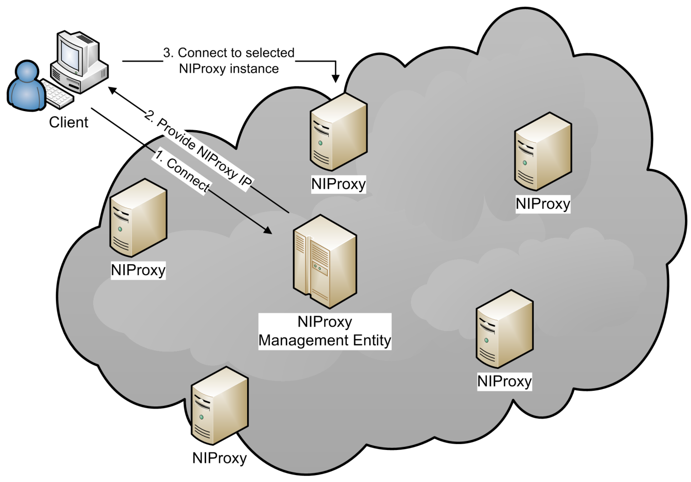


Figure 3.5: Offloading NIProxy instance selection responsibility to a centralized management entity.

cially computationally intensive. A representative example is the static video transcoding service that will be presented in section 5.3. As will be described, this service enables on-the-fly reduction of the bitrate of H.263-encoded video fragments via transcoding, which is a time-consuming and complex operation. Since the computational power of the devices on which the NIProxy instances are hosted is not limitless, they might become overwhelmed as the number of connected clients rises. Overloaded NIProxy instances are likely to stop functioning as expected or required, which in turn might yield poor QoE optimization results. Deploying additional NIProxy instances on separate devices is a straightforward approach to address this issue. A more advanced solution might involve the (temporary) transfer of computational tasks from a heavily loaded NIProxy instance to a more idle peer. At the moment, such task transfer functionality is however not yet supported.

In case multiple NIProxy instances are available, the transportation network should ideally also include a management entity which automates the process of NIProxy instance selection for clients. In particular, the management entity would be responsible for assigning clients to a particular NIProxy instance. As is illustrated in figure 3.5, in this setup, a new client would connect to the manager instead of directly to a particular NIProxy instance. The manager would then reply with the IP address of the NIProxy instance to which the client should connect. The management element could base its client allocation decisions on multiple factors, including client/NIProxy network proximity and current NIProxy load. Exploratory research in this direction has already been performed [Sels 09].

3.6 The NIProxy as Part of Larger QoE Optimization Frameworks

Ideally, a QoE optimization system like the NIProxy should be able to satisfy the QoE requirements of users of all possible distributed applications. Due to the large diversity in user expectations and preferences, application characteristics, possible multimedia content optimizations and so on, this is however neither a realistic nor practically feasible objective. The NIProxy provides a number of QoE optimization options that are potentially useful to a multitude of distributed applications. Using the NIProxy as a stand-alone entity is hence definitely justifiable as it will in many situations lead to an improvement in the QoE witnessed by the user. The NIProxy could however also be combined with other QoE management components, preferably solutions which concentrate on forms of QoE optimization which are not explicitly addressed by the NIProxy. By intelligently coordinating and exploiting the particular features and specialisms of its composing elements, such an integrated framework will normally be able to produce improved QoE optimization results that are not attainable by any of its constituting components individually. Examples of the enhanced QoE optimization potential of collaborating QoE systems will be presented in chapters 12 and 13.

Chapter 4

Network Traffic Shaping

Network traffic shaping (NTS) is the first traffic engineering facility that is provided by the NIProxy to enable IP-based networks to influence the QoE of users of distributed applications. The NIProxy's accumulated contextual knowledge is in this case exploited to in-network orchestrate the bandwidth consumption of distributed applications. An important objective for the NTS process is to ensure that distributed applications respect their available bandwidth capacity, since failure to do so might result in over-encumbered network connections and hence congestion. This in turn is likely to have seriously detrimental effects on the quality of the data dissemination. Within the constraints of the bandwidth availability, the NTS process is responsible for deciding how bandwidth should be distributed among the network flows that are involved in the execution of the distributed application. As an example, in case of a drop in available throughput, how should network flow bandwidth assignment be reshuffled? Due to the high throughput requirements and sensitivity to packet loss (a common consequence of network congestion) of multimedia content, network traffic shaping forms an important tool to arm transportation networks against the complications that are introduced by the dissemination of this type of data. This chapter will provide an in-depth description of the operation and implementation of the NIProxy's network traffic shaping framework.

4.1 Arranging Network Traffic in a Stream Hierarchy

The NIProxy implements network traffic shaping by organizing network flows in a *stream hierarchy* [Monsieurs 05][Wijnants 08b]. The goal of this tree-like structure is to capture the relationships that exist between network traffic that is introduced by a distributed application. The stream hierarchy supports both fine- and coarse-grained relationship expression, as relations can be defined on a per flow basis as well as using flow aggregates. In other words, the stream hierarchy enables the specification of bandwidth distribution guidelines for

- individual network flows (e.g., how should bandwidth be apportioned among audio stream X and video stream Y?)
- collections of network flows (e.g., what is the relationship between audio traffic and video traffic as a whole and how should bandwidth hence be distributed over both traffic classes?)

The body of the stream hierarchy is formed by internal nodes which each implement a certain bandwidth distribution strategy. As such, they dictate the bandwidth allocation process. Leaf nodes on the other hand do not provide any NTS functionality but instead represent actual network streams (e.g., a particular video flow). Multiple classes of internal as well as leaf nodes exist, each having specific characteristics and modi operandi. These will be described in sections 4.2 and 4.3, respectively.

The NIProxy's network traffic shaping operation is controlled entirely by the types of internal nodes that are used and the way these nodes are composed to model the general layout of the stream hierarchy. Once this layout has been constructed and assuming the stream hierarchy is kept up-to-date (e.g., newly initiated network flows are adequately incorporated in it), performing network traffic shaping simply amounts to appointing the correct bandwidth amount to the hierarchy root node. The internal nodes, commencing with the root node, will recursively distribute their assigned bandwidth value over their children according to the bandwidth distribution scheme which they implement. Eventually, portions of the total bandwidth capacity will reach one or more leaf nodes in the stream hierarchy, at which point this bandwidth amount will be reserved for the transmission of the network stream that is associated with the leaf node.

To be able to cater to dynamic events and to guarantee that the produced bandwidth allocation will at all times be correct, the NIProxy periodically repeats the NTS process. Such a dynamic event could for instance be the termination of a network flow. As a network flow ceases to exist, it should be removed from the stream hierarchy and the bandwidth designation should be revised since any bandwidth that was previously reserved for the terminated stream has become available for distribution over the remaining flows. Another example of a possible dynamic event is a shift in the relationship between network flows or flow aggregates, which necessitates a readjustment of the bandwidth partitioning among the involved (collections of) flows.

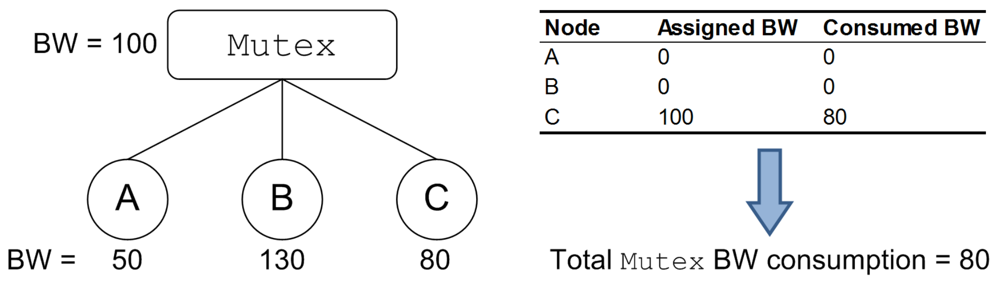
The stream hierarchy methodology for managing the bandwidth consumption of distributed applications consumes computational resources. Given the periodic repetition of this process, care has to be taken to minimize the computational overhead which it introduces. This is achieved by the incorporation of a caching scheme for bandwidth allocation values in the stream hierarchy. Only for subtrees that were affected by a dynamic event or that have changed since the previous NTS iteration, new bandwidth distribution results will be calculated (and subsequently cached). Nodes belonging to unaltered stream hierarchy subtrees simply reuse their cached value since recalculation would in these cases yield an identical result and would hence merely waste scarce computational resources.

4.2 Internal Node Types

The internal nodes structure the stream hierarchy and steer the bandwidth brokering process by designating bandwidth to their children in a specific manner. This section will provide a detailed description of the bandwidth allocation behavior that is implemented by the different types of internal nodes [Wijnants 08b].

4.2.1 `Mutex`

The most simple type of internal node is the `Mutex`. As alluded to by its name, it implements mutual exclusion behavior. The shared resource in this case equals the bandwidth that has been reserved for the `Mutex` node, while the entities that compete for this resource correspond to the child nodes. In other words, a `Mutex` node ensures that at all times at most one of its children will consume bandwidth. The child selection procedure is a function of both the available bandwidth amount and the bandwidth requirements of the children. More specifically, the distributable bandwidth BW is allotted in its



Node	Assigned BW	Consumed BW
A	0	0
B	0	0
C	100	80

Total `Mutex` BW consumption = 80

Figure 4.1: Example `Mutex` node operation.

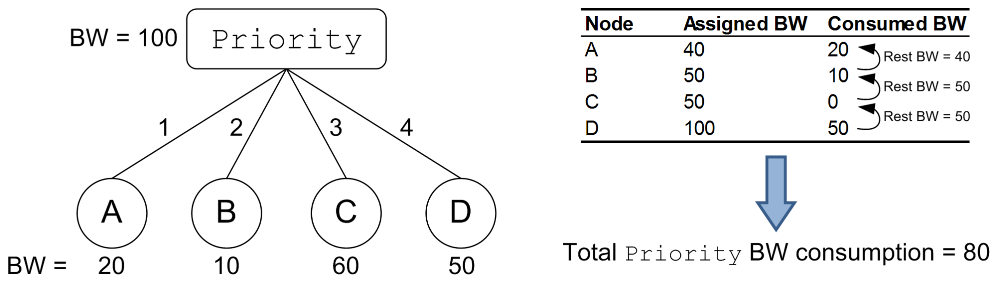
entirety to the child node which exhibits the largest still satisfiable bandwidth requirement (i.e., the child with the highest bandwidth consumption $\leq BW$); none of the other children will receive any bandwidth. In case no child node with reconcilable bandwidth demands exists, the subtree that is rooted at the `Mutex` node will consume no bandwidth at all.

Figure 4.1 illustrates the bandwidth distribution approach that is implemented by the `Mutex` node type through a simple example. The entire bandwidth amount that is available to the `Mutex` node is dedicated to C, the child node with the largest still satisfiable bandwidth requirement. As a result, the complete `Mutex` subtree consumes 80 BW. The superfluous capacity, 20 BW, remains unused and becomes available for use by siblings of the `Mutex` node in the stream hierarchy, if any.

4.2.2 `Priority`

Children of this type of internal node have a *priority value* $p \in \mathbb{N}$ associated with them and bandwidth is partitioned statically in order of descending priority. In particular, the bandwidth amount BW that is available to the `Priority` is first allocated to child node c with the largest priority value p_c, resulting in a child bandwidth consumption of $BW_c \leq BW$. In case this child does not fully consume the bandwidth amount which it was assigned, the excess bandwidth (i.e., $BW - BW_c$) is subsequently designated to the node with the second largest priority value. This process is repeated until either all available bandwidth has been distributed or all children have been considered. In summary, `Priority` children are allocated the fraction of BW that is left unconsumed by peer nodes with a higher priority value.

An illustrative example of the `Priority` node's operation is depicted in Figure 4.2. As node D has the highest priority value, it is first presented the distributable bandwidth and consumes half of it. The remaining 50 BW is



Node	Assigned BW	Consumed BW
A	40	20
B	50	10
C	50	0
D	100	50

Figure 4.2: Example `Priority` node operation.

subsequently made available to node C, which has the second highest priority value. The dedicated amount however does not suffice to satisfy C's bandwidth requirements and hence C consumes no bandwidth. Finally, nodes B and A are considered, in this order, and respectively granted 50 and 40 BW, which results in a respective bandwidth consumption of 10 and 20 BW. The accumulated bandwidth usage of the `Priority` subtree hence equals 80 BW.

4.2.3 `Percentage`

The `Percentage` internal node type implements a two-phase bandwidth regulation process. The `Percentage` node starts by granting each child c its corresponding *percentage value* $p_c \in [0, 1]$ of the distributable bandwidth BW. In other words, child c is apportioned a bandwidth amount

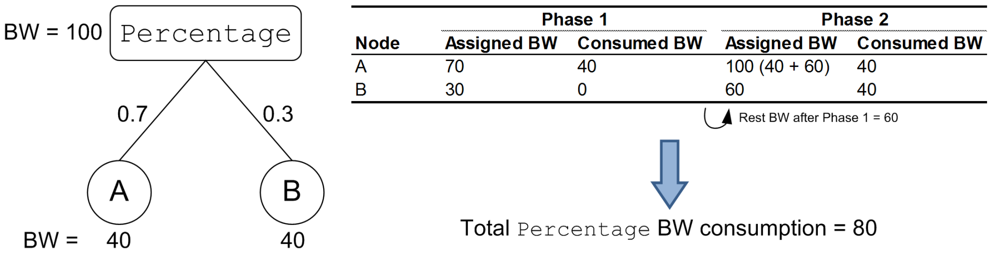
$$BW_c = p_c \times BW$$

Child percentage values are hereby scaled so that they sum up to unity:

$$\sum_{children} [p_{child}] = 1$$

After executing this initial phase, a portion of BW might be left unallocated due to children not fully consuming their assigned bandwidth percentage. If so, the `Percentage` node enters the second phase of its execution, where it attempts to distribute this excess bandwidth by assigning it to child nodes on a one-by-one basis, in order of decreasing percentage value. The second phase in other words allows children to upgrade their bandwidth consumption based on surplus bandwidth inherited from phase 1, hereby favoring children with high percentage values.

Figure 4.3 exemplifies the operation of the `Percentage` node type. In the initial phase, nodes A and B receive 70 and 30 percent of the available


BW = 100
Percentage
0.7
0.3
A
B
BW = 40
40
Rest BW after Phase 1 = 60
Total Percentage BW consumption = 80

	Phase 1		Phase 2	
Node	Assigned BW	Consumed BW	Assigned BW	Consumed BW
A	70	40	100 (40 + 60)	40
B	30	0	60	40

Figure 4.3: Example `Percentage` node operation.

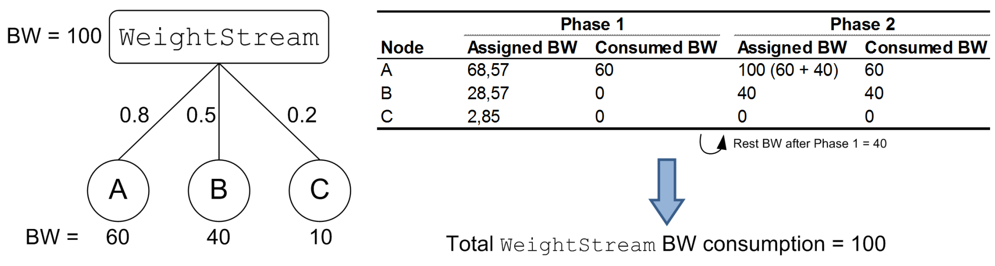
bandwidth capacity, respectively. At the beginning of phase 2, only 40 BW has already been consumed (by node A). The residual 60 BW is in the second phase first assigned to node A, the child with the highest percentage value. As node A cannot exploit this additional bandwidth to upgrade its bandwidth consumption, the 60 BW is subsequently allocated to node B, which employs it to satisfy its bandwidth requirement of 40 BW. The bandwidth consumption of the `Percentage` node in this example hence totals 80 BW.

4.2.4 `WeightStream`

Like `Percentage` nodes, the `WeightStream` node type operates in two consecutive phases. In the first phase, all children are considered collectively and the available bandwidth BW is partitioned among them as a function of both their current *weight value* $w_c \in [0, 1]$ and their maximal bandwidth consumption $MaxBW_c$. In mathematical notation, each child node c is apportioned a bandwidth amount BW_c as follows:

$$BW_c = w_c \times MaxBW_c \times f; f = \frac{BW}{\sum_{children}[w_{child} \times MaxBW_{child}]}$$

In this formula, the factor f includes the combined weighted maximal bandwidth consumption of all children. It is a scaling factor as it enforces that, on completion of the initial phase, the collective child bandwidth consumption will not exceed the available bandwidth capacity (i.e., $\sum_{children}[BW_{child}] \leq BW$). The second phase of the `WeightStream`'s bandwidth distribution strategy mimics the operation of the `Percentage` node type. In other words, any bandwidth which is left unallocated in the initial phase is exploited in the second stage to attempt to increase the bandwidth usage of individual child nodes successively, this time however in order of decreasing weight value. Notice that the total of child nodes' weight values not necessarily needs to equal 1.


BW = 100
WeightStream
0.8
0.5
0.2
A
B
C
BW = 60
40
10
Rest BW after Phase 1 = 40
Total WeightStream BW consumption = 100

	Phase 1		Phase 2	
Node	Assigned BW	Consumed BW	Assigned BW	Consumed BW
A	68,57	60	100 (60 + 40)	60
B	28,57	0	40	40
C	2,85	0	0	0

Figure 4.4: Example **WeightStream** node operation.

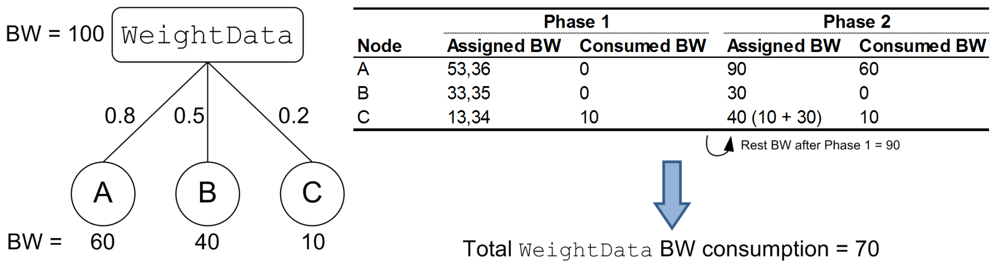
The bandwidth distribution policy of the **WeightStream** node type is exemplified in Figure 4.4. In this example, the bandwidth values that are granted to children A, B and C in the first phase respectively amount to 68.57, 28.57 and 2.85 BW. Notice that the sum of these individual amounts is already smaller than the total bandwidth volume that is available to the **WeightStream** node. On the basis of the initially assigned values, only node A is capable of satisfying its bandwidth requirements. The total bandwidth consumption upon completion of phase 1 hence corresponds to 60 BW. In the second phase of the **WeightStream**'s operation, node A, the child with the highest weight value, is considered first. Node A however has no use for the 40 BW that is inherited from the first phase. The next node to consider according to the weight value hierarchy is child B, whose unsatisfied bandwidth requirement exactly matches the superfluous bandwidth volume. Consequently, node B completely seizes the still available bandwidth, at which point the execution of the second phase of the **WeightStream** node is terminated as there is no bandwidth left to divide. The cumulative bandwidth consumption of the **WeightStream** subtree hence equals 100 BW, the input bandwidth volume.

4.2.5 WeightData

As is reflected in their names, **WeightData** and **WeightStream** are related internal node classes. They implement an identical bandwidth brokering approach, except the **WeightData** type disregards the maximal bandwidth consumption of its children [Wijnants 08a]. As a result, the bandwidth allocation that is enforced in the initial phase equals

$$BW_c = w_c \times f ; f = \frac{BW}{\sum_{children}[w_{child}]}$$

Although their implementation differs only minimally, the **WeightData** and **WeightStream** classes produce radically divergent bandwidth brokering


BW = 100
WeightData
0.8
0.5
0.2
A
B
C
BW = 60
40
10
Rest BW after Phase 1 = 90
Total WeightData BW consumption = 70

	Phase 1		Phase 2	
Node	Assigned BW	Consumed BW	Assigned BW	Consumed BW
A	53,36	0	90	60
B	33,35	0	30	0
C	13,34	10	40 (10 + 30)	10

Figure 4.5: Example `WeightData` node operation.

results. This is confirmed by the example that is provided in Figure 4.5. The depicted stream hierarchy is identical to the one that was described in section 4.2.4, both structurally and in terms of the weight values and bandwidth requirements of child nodes; the only difference is that the root node has been substituted with a `WeightData` instance. The bandwidth distribution outcomes of both setups are however far from alike. In particular, in the `WeightData` example, the initial child node bandwidth allocations correspond to 53.36, 33.35 and 13.34 BW for nodes A, B and C, respectively. Only for node C, the appointed bandwidth capacity exceeds the required amount. The residual bandwidth after the execution of the first stage consequently equals 90 BW. The order in which children are consulted in the second phase is identical to the situation in section 4.2.4. As a result, the excess bandwidth is first allotted to node A, which subtracts 60 BW from it to satisfy its bandwidth requirements. Next, node B is consulted, whose bandwidth demands surpass the remaining 30 BW. Finally, node C is considered, again without effect as it is already operating at maximal bandwidth consumption. The cumulative bandwidth utilization in this case hence amounts to 70 BW (versus 100 BW in the `WeightStream` example). Additional illustrations of the differences between the `WeightStream` and `WeightData` node types in terms of bandwidth distribution behavior will be provided during the discussion of the NIProxy's experimental evaluations in part II of this thesis.

As will be discussed in detail in chapters 8 and 9, the `WeightData` node type was introduced due to `WeightStream`'s empirically observed failure to adequately manage bandwidth in the presence of non-real-time data traffic. In particular, the inclusion of the maximal bandwidth consumption of child nodes in the distribution scheme has shown to cause unexpected behavior in case at least one child represents a non-real-time bulk data transfer (e.g., the transmission of a file).

4.3 Leaf Node Types

A leaf node in the stream hierarchy always corresponds to an actual network flow. As is the case for internal stream hierarchy nodes, there also exist multiple leaf node classes [Wijnants 08b]. However, whereas internal nodes are categorized according to their bandwidth distribution approach, the classification of leaf nodes is based on their capabilities to control and modify the bandwidth consumption of the network stream which they represent.

4.3.1 `Discrete` Leaf Node

Leaf nodes belonging to the `discrete` category define a discrete number of increasing bandwidth levels and are capable of toggling the bandwidth usage of the network flow which they represent between those levels. In its most rudimentary form, a `discrete` leaf node supports only two levels, which correspond to respectively a zero and maximal stream bandwidth consumption. This kind of `discrete` leaf node is in other words confined to turning its associated network flow off and on and can hence only effectively model single-quality network flows in the stream hierarchy. To be able to correctly manage scalable or multi-layer encoded network traffic, more advanced `discrete` leaf node implementations will typically be required which specify one or multiple intermediate bandwidth consumption levels that lie in between the extremes of turning the stream off and forwarding it at its maximal bitrate. Scalable network flows namely consist of a base layer, which provides the transported content at a certain baseline quality, and one or more enhancement layers, which each enhance the reception quality of the content in a particular manner. Although this type of network traffic in theory enables dynamic control over its transmission rate, a source might consistently transmit all layers of the scalable network stream and leave it up to the destination to select and receive only those layers which it currently requires or is capable of processing. Consequently, a suitable `discrete` leaf node for this type of network traffic would include a number of monotonously increasing bandwidth levels, where each higher level would permit the forwarding of an additional layer to the destination.

`Discrete` leaf nodes are very lightweight. They demand a minimal bookkeeping effort as the supported discrete levels and their corresponding bandwidth values need to be stored. The operation of this class of leaf nodes however does not require complex or time-consuming processing to be performed on the network flow which they are associated with. In particular, whenever a `discrete` leaf node is granted an amount of bandwidth by its parent in the stream hierarchy, it will exploit this bit budget to simply switch the embodied network flow to its largest satisfiable discrete bandwidth consumption level. This implies that `discrete` leaf nodes introduce nearly no computational overhead. As will be elaborated on in chapter 8, they are therefore very well suited to represent real-time network traffic in the stream hierarchy, since such traffic is characterized by stringent reception delay constraints and hence needs to be received by the destination in a timely manner.

4.3.2 `Continuous` Leaf Node

Unlike `discrete` leaf nodes, the class of `continuous` leaves is able to modify their associated stream's bandwidth consumption in a continuous manner. Stated differently, these nodes define an entire bandwidth consumption range instead of a discrete number of bandwidth values and they provide functionality to set their corresponding flow's bandwidth usage to an arbitrary value within this interval. In its most complete form, the lower bound of the bandwidth range that is provided by a `continuous` leaf node equals zero, while the upper bound coincides with the associated stream's maximal bandwidth consumption or the throughput of the destination's network connection, whichever is lower. When a leaf node of the `continuous` category is apportioned an amount of bandwidth by its parent in the stream hierarchy, its corresponding network flow will be forwarded at exactly this rate[1].

`Continuous` leaf nodes are obviously more powerful than their `discrete` counterparts since they enable additional flexibility and dynamism to be introduced in the network traffic shaping process. This however comes at the expense of increased resource consumption and processing requirements. In particular, contrary to `discrete` leaf nodes, the operation of `continuous` leaves typically requires the adoption of resource-intensive and possibly computationally complex techniques like, for instance, mid-stream buffering, real-time transcoding and/or dynamic rate control. As a result, the `continuous` leaf node type is innately less tailored to the management of real-time network traffic (although, as will be demonstrated in chapter 13, they could nonetheless effectively be used for exactly this purpose). On the other hand, chapter 8 will establish that their provided functionality makes them obvious candidates for handling non-real-time data like a file transfer.

[1]This statement assumes that the allocated bit budget is enveloped in the supported continuous bandwidth consumption interval. In case the reserved bandwidth amount is smaller than the lower bound of the supported range, the associated stream will not be dedicated any bandwidth; conversely, whenever the bit budget exceeds the upper bound, the forwarding rate will match this upper limit, which implies that a fraction of the leaf node's granted bandwidth will remain unexploited.

Table 4.1: Calculating the maximal bandwidth consumption of stream hierarchy nodes.

Node Type	**Maximal Bandwidth Consumption**
`Discrete` leaf node	Bandwidth value that is associated with the largest discrete level
`Continuous` leaf node	Upper bound of the supported bandwidth consumption range
`Mutex`	The maximal bandwidth consumption of the child with the highest bandwidth requirements
`Priority`, `Percentage`, `WeightStream`, `WeightData`	Sum of the maximal bandwidth usages of all child nodes

4.4 Maximal Bandwidth Consumption

In the discussion thus far, the term "maximal node bandwidth" has appeared a number of times. It is possible for stream hierarchy nodes to support various bandwidth consumption amounts. This is true for both leaf and internal nodes. An example in the former category is a `discrete` leaf node which defines multiple non-zero bandwidth levels. For internal nodes, it is even the rule rather than the exception. Internal nodes act as root of subtrees in the stream hierarchy and their bandwidth consumption is an accumulation of the bandwidth usage of their children. As a result, internal node bandwidth usage can often take on a range of values. Maximal bandwidth therefore refers to the amount of bandwidth which a node can at most consume. Table 4.1 summarizes how this value is computed for the different leaf and internal node types.

4.5 Sibling Dependencies Framework

The NIProxy's bandwidth management mechanism includes a framework that enables dependencies to be specified and enforced between sibling nodes in the stream hierarchy [Wijnants 09b]. Siblings are in this context defined as nodes which share the same parent. Since this is a relatively recent addition, so far only the `SD_BW_ALLOC_CONSTRAINED` dependency has been defined. The set of supported dependency types could however readily be extended, for example based on future network traffic shaping requirements. The `SD_BW_ALLOC_CONSTRAINED` dependency introduces a bandwidth allocation constraint between sibling nodes in the stream hierarchy. In particular, the existence of such a dependency between nodes A and B specifies that node B is allowed to consume bandwidth if and only if A's bandwidth consumption is non-zero. In case this restriction is violated, node A is allowed to alter the current bandwidth distribution outcome by "borrowing" the bandwidth amount that was assigned to node B. If the combined bandwidth reserved for nodes A and B remains insufficient to satisfy node A's minimal bandwidth requirements, both nodes will be turned off and a new network traffic shaping iteration will be performed from which the subtrees that are rooted at nodes A and B are virtually excluded. As a result, the bandwidth that was originally designated to node B will become available for the other nodes in the stream hierarchy (recall that node A's bandwidth consumption was zero in the initial iteration). If on the other hand the additional bandwidth does enable node A to switch to its minimal non-zero bandwidth consumption, node A is allowed to do so; any bandwidth that remains from the combined amount will subsequently be allocated to node B. Note that this latter value will inherently be smaller than the bandwidth that was at first reserved for node B, which implies that node B might be forced to lower its bandwidth consumption or might even get completely disabled. An example of the operation of the sibling dependencies framework will be given in chapter 11.

4.6 Overflow Prevention Buffer

The bandwidth behavior of network flows that are managed by the NIProxy (i.e., which are incorporated in a client's stream hierarchy) might alter outside of the NIProxy's control. It is apparent that the NTS scheme will need to react to such changes so that the correctness of the bandwidth brokering outcome is ensured.

A first possibility is that the throughput of a network stream changes permanently or at least for a prolonged period of time. For example, the data source might for whatever reason fundamentally modify its transmission rate. Such long-term bandwidth changes are automatically dealt with quite adequately by the NIProxy's NTS framework. In particular, the NIProxy's estimate of the flow's bandwidth consumption will gradually converge to the new value, from which point on the validity of the bandwidth distribution results will again be guaranteed. As a result, no specific functionality or constructs have been implemented to address this issue.

The bitrate of network traffic might on the other hand also exhibit swift and very temporary variations. The absence of a rate controller at the origin, for instance, could lead to severe fluctuations in flow bandwidth consumption (i.e., short periods of either decreased or increased throughput). Such bursty changes are more problematic to cope with compared to the previously described steady bandwidth modifications. In particular, due to the short-lived nature of these bursts and due to the fact that the NIProxy updates its flow bandwidth requirement measures only periodically (typically once every second, see section 6.2.2 for more information), the sudden drop or increase in bandwidth consumption might sometimes remain unnoticed by the NIProxy for a short period of time. This will in turn cause the NTS scheme to base its calculations on incorrect values, with potentially faulty bandwidth brokering decisions as outcome. In case the fluctuation yields a momentary reduction in throughput, the consequences will typically be acceptable, if not negligible: it will at most result in an incomplete exploitation of the available bandwidth during a short time interval because the NIProxy temporarily overestimates the bandwidth requirements of the involved flow. Conversely, if the bandwidth usage is subject to sudden peaks, the situation becomes more troublesome. In this case, the variation might cause the NIProxy to temporarily underrate the flow's bandwidth consumption, which might in turn lead to infringement of the available bandwidth capacity. Failure to respect the bandwidth constraints of a network connection will typically entail detrimental effects such as congestion, elevated delay and/or an increased packet loss ratio. Since such network anomalies are in turn likely to negatively impact user QoE, it is apparent that connection overflow should be prevented as much as possible.

To address the issue of possible violation of network link capacity due to ephemeral surges in the throughput of network traffic, the NIProxy's NTS framework is furnished with an overflow prevention buffer. In particular, it is possible to exclude a fraction of the available bandwidth volume from the bandwidth allocation process. This portion will hence not be distributed among the involved network flows, but will instead be exerted to fulfill the role of safety buffer so that a certain amount of resilience to swift increases in network flow bandwidth consumption is guaranteed. In the optimal case, the overflow prevention buffer will completely neutralize bandwidth usage peaks and will as such prevent link capacity violations from occurring. In case the bandwidth margin does not suffice to absorb the sudden increase, it will at least moderate its impact. The operation of the overflow prevention margin will be exemplified in chapters 8 and 9.

Parametrization of the size of the safety buffer is supported, even on a per client basis. As a result, it is possible to attune it to the current context of use. Possible factors which might influence the determination of the buffer size include the composition of the network traffic mix and knowledge of the existence of a network flow with a bursty bandwidth consumption behavior. Note that this implies that it is just as well possible to completely disable the safety buffer (i.e., set its size to zero) in environments in which its presence is unnecessary or undesired. As an example, certain users might prefer not to include a NTS safety margin so that the utility of their bandwidth capacity is at all times maximized and are prepared to cope with the resulting heightened risk of network connection overflow.

4.7 Stream Hierarchy Construction and Management

Constructing and governing the stream hierarchy nearly exclusively falls under the responsibility of the managed client. Certain aspects of stream hierarchy management can be implemented autonomously by the NIProxy, without direction from the managed client; an example hereof will be described in section 5.3.1. The bulk of the stream hierarchy operations however does require client participation. In particular, the following tasks are imposed on the client:

- devising and installing a suitable general structure for the stream hierarchy
- guaranteeing that the stream hierarchy remains up-to-date (i.e., that all relevant network traffic is adequately incorporated in it)

Both topics will be discussed in more detail next. Before proceeding however, ascertain that the stream hierarchy is maintained entirely at the NIProxy. By delegating tasks and decisions concerning the stream hierarchy to the client, it becomes necessary for the client to communicate desired stream hierarchy actions to its NIProxy instance. This responsibility is alleviated by the availability of the client-side NILayer support library. In particular, as was mentioned in section 3.4, the NILayer implements the NI communication protocol and a part of its exported API is devoted specifically to stream hierarchy-related information exchange between the distributed application and the NIProxy. Furthermore, to coordinate and accelerate the decision making process for operations that are related to the stream hierarchy, the NILayer stores a simplified representation of the stream hierarchy at client-side. This stripped version does not contain any NTS functionality; instead, it simply provides the

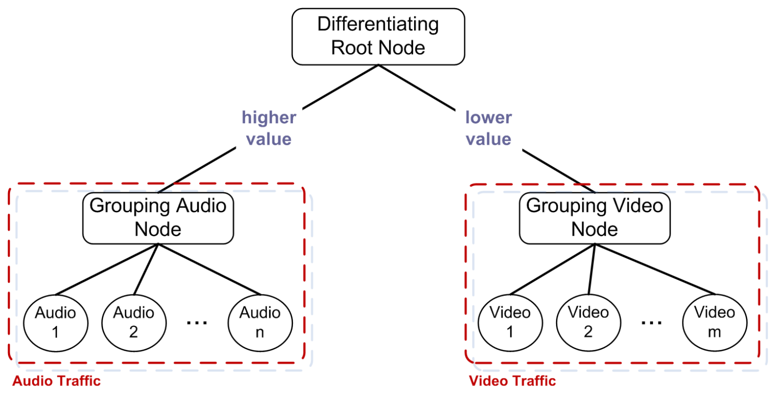


Figure 4.6: Notifying the NIProxy's NTS mechanism of the fact that the user prefers audio to video through appropriate stream hierarchy organization.

client with a "read-only" view that is consistent with the stream hierarchy's current state at NIProxy-side. As a result, the client can found its decisions on this local simplified copy, this way effectively eliminating unnecessary stream hierarchy-related communication between the client and the NIProxy.

4.7.1 Global Stream Hierarchy Layout Determination

Selecting a suitable stream hierarchy layout can be considered as an act of providing the NIProxy with application awareness. Indeed, the global organization of the stream hierarchy indicates how the different network streams in which the client is interested relate to each other, in terms of significance for the client. For example, to inform the NIProxy that it attaches larger importance to audio as compared to video, the client could devise a stream hierarchy layout which resembles the one depicted in Figure 4.6. In this (conceptual) example, the client has captured its preference for audio by relying on some type of differentiating internal node to distinguish between audio and video network traffic and by subsequently assigning the grouping audio node a higher value than its video sibling. Which internal node classes should be employed as root of the stream hierarchy and as grouping audio and video node (i.e., `Priority`, `Percentage`, `WeightStream` or `WeightData`) depends on the user's preferences and possibly some other application-related information. As a result, the responsibility for making such decisions would again lie with the client.

4.7.2 Stream Hierarchy Governance

The data which a distributed application injects in the transportation network during its execution might not be homogeneous over time. In particular, the composition of the generated network traffic ensemble is likely to vary since new network flows might at run-time be initiated, while others conversely might be terminated. The managed client bears the responsibility for keeping its stream hierarchy in sync with the network traffic production behavior of the distributed application. In particular, since only network traffic that is represented in the stream hierarchy is accounted for by the NIProxy's NTS process, the client needs to ensure that all active network flows in which it is interested are adequately included. In contrast, discontinued network flows will typically need to be purged from the stream hierarchy to prevent them from wastefully hogging scarce network bandwidth; as an additional incentive, a smaller stream hierarchy size will typically amount to a faster operation of the NTS algorithm.

Supposing the client has perfect and complete knowledge of the distributed application's network traffic production scheme and of the scheme's evolution over time, it could entirely autonomously modify the state of its stream hierarchy to correctly reflect the current situation. This is unfortunately often an unrealistic assumption. For a video conferencing application, for instance, it is typically impossible to predict how many users will be participating in the session and which type of network traffic (e.g., voice, video, shared whiteboard data, etcetera) each will inject in the network. To address this issue and hence to facilitate the client's task of keeping its stream hierarchy up-to-date, the NI-Proxy and the NILayer auxiliary library jointly implement a *stream detection* as well as *stream termination notification mechanism.*

To enable the client to be notified of the discovery of concrete network flows, the NI protocol entails a message type through which a managed client can indicate to the NIProxy that particular network traffic categories (e.g., audio data, video data, P2P data, etcetera) need to be `block`-ed (see also section 6.2.2). Whenever the NIProxy detects a network flow which conceptually belongs to a `block`-ed traffic category and which the NIProxy's NTS framework in addition has not encountered before, a so-called *stream peek* is generated. Besides concretely identifying the newly discovered network stream, a stream peek also encompasses a limited amount of the data that was intercepted on the stream and which hence triggered the stream detection. The constructed stream peek is subsequently forwarded to the client (according to the NI protocol) instead of the detected flow itself. Upon arrival, the client will need to decide on the desired future treatment of the discovered network stream and will need to communicate its decision back to the NIProxy (again using a NI protocol message). The client can specify three possible actions:

- *accept* the discovered network flow (i.e., the NIProxy should uniformly forward all future network packets which are intercepted on this flow to their destination)
- *drop* the discovered network flow (i.e., the NIProxy should uniformly discard all future network packets which are intercepted on this flow)
- incorporate the discovered network flow in the stream hierarchy

The first option is mainly interesting for network streams which should always be delivered to the destination and which have only marginal bandwidth requirements. An example could be control data; such data is typically imperative for the correct functioning of the distributed application and, compared to multimedia traffic, usually consumes negligible bandwidth amounts. As a result, it might be justifiable to permanently accept such network traffic without including it in the NIProxy's bandwidth brokering calculations. The drop decision on the other hand could, for example, be exploited in case the client judges that the detected network traffic should under no circumstances consume bandwidth. Finally, by adhering to the last alternative, the NIProxy's NTS framework will start reckoning with the new network stream during its bandwidth management operations. Notice that, through clever stream hierarchy organization, it is possible to enforce stream priority and hence to ensure that certain network traffic will always be forwarded by the NIProxy, in case sufficient bandwidth is available to do so. Consequently, although it is likely to require some additional effort from the client, the stream hierarchy approach is preferred to the permanent accept action, especially for network traffic of significant volume, since it will yield more complete and correct (i.e., optimal) bandwidth management results.

Compared to the stream detection notification scheme, which actually implements a simple form of admission control, the stream termination counterpart is more straightforward. As will be discussed in section 6.2.2, the NIProxy monitors network flow liveliness (through expiration counters). As soon as the termination of a particular network flow is deduced, each client to which the flow was relevant will be informed. The notified clients can subsequently undertake any necessary action, which is likely to at least encompass the removal of the terminated flow from their stream hierarchy at NIProxy-side. The notification is again conveyed in the form of a NI protocol message.

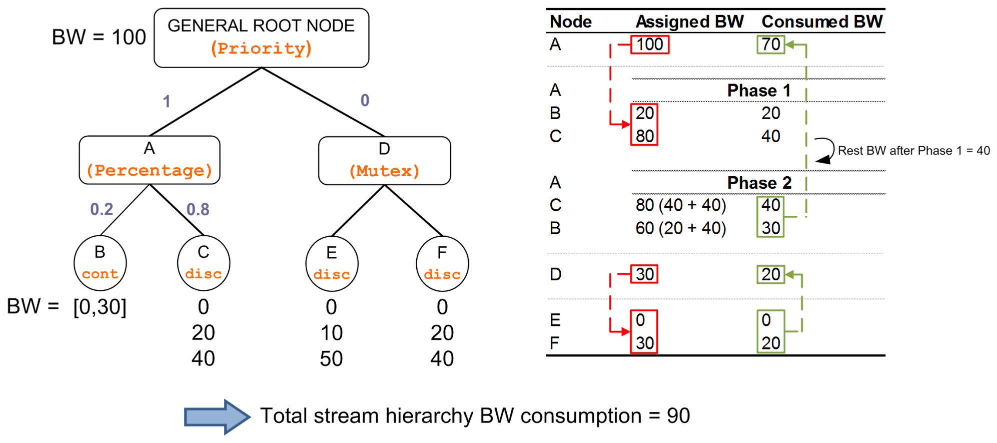


Figure 4.7: A comprehensive stream hierarchy and the resulting bandwidth brokering outcome.

4.8 A Comprehensive Example

A concrete example of a fairly comprehensive stream hierarchy is provided in Figure 4.7. The root of the stream hierarchy consists of a `Priority` node and has two direct children, nodes A and D, which are of type `Percentage` and `Mutex` and which are assigned a priority value of 1 and 0, respectively. The `Percentage` node in turn has two children of its own, namely `continuous` leaf node B with a percentage value of 0.2 and `discrete` leaf node C which has a percentage value of 0.8. Node B's bandwidth consumption can assume an arbitrary value in the range $[0, 30]$ BW; node C on the other hand supports 3 discrete bandwidth levels which respectively consume 0, 20 and 40 BW. `Mutex` node D also serves as parent for two leaf nodes (i.e., E and F), which are this time however both of the `discrete` type. Both leaf nodes define 3 discrete levels which correspond to bandwidth requirements of 0, 10 and 50 BW for node E and 0, 20 and 40 BW for node F.

In the provided example, the distributable bandwidth equals 100 BW. As was described in section 4.1, the network traffic shaping process is initiated by appointing this bandwidth amount to the hierarchy root node. As this node is of type `Priority`, the bandwidth budget is first presented in its entirety to node A, the child with the highest priority value. At this point, the two-phase bandwidth brokering policy that is implemented by the `Percentage` node type is started, with 100 BW as input bandwidth. In the initial stage, this results in a bandwidth allocation of respectively 20 and 80 BW to nodes B and C. Since node B's assigned bandwidth value falls within its supported continuous range, its effective bandwidth consumption is set to exactly this amount. Node C on the other hand leverages the granted bandwidth to switch to discrete level 2. The cumulative bandwidth consumption of the subtree that is rooted at `Percentage` node A therefore equals 60 BW at the end of the first stage of A's bandwidth brokering scheme. The unexploited 40 BW is in the second phase first dedicated to child node C (because it has the largest percentage value associated with it). As this node is however already operating at its maximal discrete level (i.e., the level with the largest corresponding bandwidth consumption), the superfluous bandwidth cannot be allocated here. Next, node B is considered, which subtracts 10 BW from the apportioned amount to increase its bandwidth consumption to the upper limit of its supported interval. This concludes the second stage of node A's operation, yielding a total subtree bandwidth consumption of 70 BW. The remaining bandwidth budget (i.e., 30 BW) is subsequently transferred to `Mutex` node D, the second child of the stream hierarchy root node. This budget suffices for both its children (i.e., leaf nodes E and F) to enable their discrete level 1. Notice however that the bandwidth requirements that are associated with these levels are unequal (i.e., 10 BW for node E versus 20 BW for node F). Since the bandwidth allocation strategy of the `Mutex` node type dictates that the input bandwidth is allotted exclusively to the child with the largest still satisfiable bandwidth demand, the available 30 BW is reserved for leaf node F. The assigned bandwidth is exploited by child F to activate its discrete level 1. The current network traffic shaping iteration is thereby concluded since all nodes of the stream hierarchy have been considered. The complete stream hierarchy hence consumes 90 BW and leaves 10 BW unexploited.

Chapter 5

Multimedia Service Provision

The previous chapter has discussed network traffic shaping, the first traffic engineering technique that is supported by the NIProxy. The topic of this chapter is multimedia service provision, the NIProxy's second tool to affect network data dissemination and hence user QoE. The NIProxy acts as service provision and delivery platform, meaning it provides a substrate for the execution of services on (multimedia) network streams as they pass through it. In other words, the NIProxy introduces the possibility to perform processing on multimedia traffic during its dissemination through the communication network. The range of services that can be provided is theoretically limitless, spanning from completely generic to highly application-specific and from very lightweight to computationally complex. Examples include simple data filtering, the transcoding and transmoding of multimedia data, network flow encryption to increase security and privacy, services which increase the resilience of network traffic to transmission errors, and so on. Analogous to the network traffic shaping functionality, the multimedia service provision framework is context-aware, which implies that NIProxy services can consult the NIProxy's repository of contextual information. This feature enables the implementation of highly effective services, as it allows them to attune their operation to the current context of use and hence to make sure that the processing which they implement will actually lead to an improvement of the end-user experience. The contextual knowledge could however also be exploited for secondary purposes, for instance to optimize the efficiency of resource-intensive services. Its service provision facilities enable the NIProxy to address the growing adaptability and dependability requirements of emerging distributed applications, since it allows multimedia content to be meticulously reconciled with, for instance, current channel conditions or the hardware limitations of the end-user's terminal.

5.1 Pluggable Design

Implementation-wise, the multimedia service provision framework adheres to a pluggable design [Wijnants 07]. Services correspond with NIProxy plug-ins that can be on-demand loaded and unloaded during NIProxy execution. In other words, the functionality which is implemented by services can dynamically be plugged into the NIProxy when desired or as it is demanded by current contextual conditions. Conversely, when a service is no longer needed, its plug-in is unloaded, this way effectively freeing up NIProxy resources so that these can be employed for other purposes. All of this can be done at run-time (i.e., without requiring a NIProxy reboot).

By opting for a plug-in-based design, services become stand-alone entities that are conceptually separated not only from each other but also from the NIProxy's general software architecture. This decoupling confers several advantages; the most important ones are enumerated below:

- The process of adding new services to the NIProxy is simplified and accelerated. Providing a new service never requires that changes are made to the NIProxy's software architecture. Instead, it suffices to simply implement a new NIProxy plug-in

- Third-party service development is facilitated since the necessity for service implementers to have extensive knowledge of the NIProxy's internals is eliminated

- Due to services being dynamically installable during operation, maximal flexibility is guaranteed and run-time extensibility of the NIProxy's functionality is achieved

- Contrary to the NIProxy itself, services are not required to be generic. While the provision of truly generic services whose functionality can be exploited in heterogeneous contexts is definitely supported, providing services that are to a lesser or larger degree tailored to a specific distributed application is just as well possible. Although this latter class of services physically resides at the NIProxy (as plug-ins), they can be conceptually thought of as being part of the distributed application which they are targeted at. Supporting both application-independent and application-specific services enlarges the applicability of the NIProxy since it enables the provision of valuable and efficient services for a wide variety of distributed applications, even concurrently. As such, the need to deploy and administer separate network infrastructure for each distinct application is eliminated, which in turn is likely to yield considerable expenditure curtailment.

Despite their isolation from each other, inter-service collaboration is enabled through the notion of *service chaining*. Multiple services, each implementing a well-delineated function, can be combined to form a service chain. Services belonging to a chain are applied consecutively so that the output that is produced by a constituting service serves as input for the next service in the chain. The support for service chaining in other words enables collaborative processing of single network streams by multiple, possibly independent NIProxy services. One way of looking at a service chain is as a complex service whose functionality equals the composition of the singular functions that are provided by its constituting services. Service chains can be personalized on a per client basis, this way yielding an increased amount of flexibility and enabling the NIProxy to efficiently satisfy each user's particular requirements and constraints.

5.2 Network Traffic Shaping Interoperation

A distinctive trait of the NIProxy is that interaction between its both traffic engineering techniques is supported. In particular, instead of treating the network traffic shaping functionality and multimedia service provision framework as isolated entities, the NIProxy allows them to interoperate and collaborate with each other. To be more precise, services are able to query and even influence and supplement the bandwidth brokering strategies which the NTS mechanism draws up for NIProxy clients.

Practical experience and experimental evaluations have corroborated that this feature enables the development of services with a high level of effectiveness, performance and resource-efficiency. More importantly however, the proposition is put forward that it affords a myriad of end-user QoE optimization possibilities and results which could not be attained by applying both mechanisms independently. This argument will be illustrated and confirmed numerous times in part II of this dissertation, where the NIProxy's practical QoE optimization performance will be evaluated through the presentation of several experimental results.

5.3 A Representative Example: Static Video Transcoding

This section will discuss the operation and implementation of an example service which introduces real-time H.263 video transcoding functionality in the NIProxy [Wijnants 07]. The service in other words enables the NIProxy to lower the bitrate of H.263-encoded video by on-the-fly reducing its quality parameters. The service will also serve as demonstrator of the interoperability potential of the NIProxy's network traffic shaping and multimedia service provision functionality (see section 5.2). The video transcoding service has been exploited in numerous NIProxy experiments. An example of the results that are achievable by the service will for instance be presented in chapter 7.

As input, the video transcoding service accepts the H.263 video stream as it is intercepted by the NIProxy; the produced output equals the transcoded version of this bitstream. At load-time, the operation of the service can be configured by specifying the required quality parameters for the output video. Spatial, temporal and video quality downscaling is supported through the specification of the desired output resolution, framerate and bitrate[1], respectively. Once an instance of this service is running however, altering the output video parameters becomes impossible. Consequently, the service is said to implement *static* transcoding, as all bitstreams that are generated by a particular instance of the service will exhibit identical quality characteristics and will consume largely comparable amounts of network bandwidth. At a later stage of this PhD research, a *dynamic* video transcoding service has also been implemented which is not confined to a single output quality but instead supports video transcoding to a range of qualities and bitrates. This dynamic variant will be described in chapter 13.

[1]The video transcoding service attempts to satisfy the desired bitrate solely by adjusting the quality parameter (Q-factor) and by increasing the step size for quantizing the transform coefficients. In other words, a bitrate transcoding target leaves the video resolution and framerate unaltered.

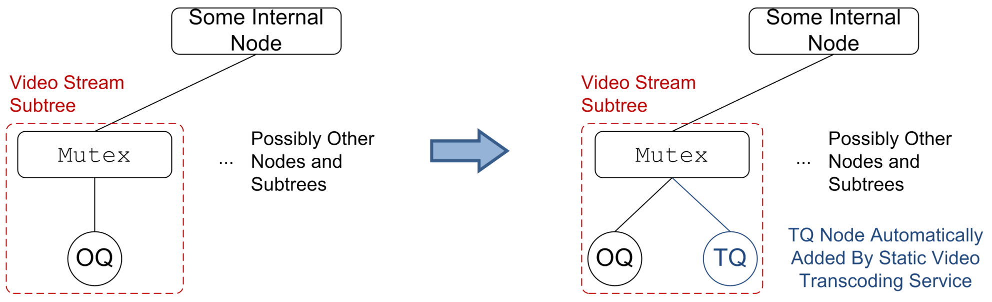

Some Internal Node
Video Stream Subtree
Mutex
OQ
... Possibly Other Nodes and Subtrees
Some Internal Node
Video Stream Subtree
Mutex
OQ
TQ
... Possibly Other Nodes and Subtrees
TQ Node Automatically Added By Static Video Transcoding Service

Figure 5.1: Example stream hierarchy before and after modification by the static video transcoding service.

5.3.1 Stream Hierarchy Incorporation

Instead of physically arriving at the NIProxy, transcoded video streams are automatically generated by the video transcoding service. This service in other words introduces a new type of network traffic of which the NIProxy is by default unaware. To inform the NIProxy of the existence of this new flow type and to ensure that it is taken into consideration during bandwidth brokering, it needs to be introduced in the stream hierarchy. The static video transcoding service therefore extends the stream hierarchy by including a new `discrete` leaf node for each distinct video flow that it processes. The new node is linked to the transcoded video stream and is added as a sibling of the node that represents the original version. The incorporated `discrete` leaf node defines two bandwidth levels which respectively correspond with a zero and maximal bandwidth consumption. The maximal bandwidth consumption of the transcoded video flow is determined on the basis of the target output bitrate that is set for the service instance. Once this incorporation has been completed, the NIProxy's network traffic shaping mechanism can start making deliberate decisions about which video version to assign bandwidth to.

Figure 5.1 exemplifies the stream hierarchy modification operations that are performed by the static video transcoding service. As can be seen, in the unmodified stream hierarchy, the Original Quality (OQ) of the video stream is represented by a subtree which consists of a `Mutex` internal node and a leaf node that corresponds with the actual network flow. In the updated stream hierarchy, the video transcoding service has incorporated an additional `discrete` leaf node to embody the Transcoded Quality (TQ) of the video flow; the new node was made a sibling of the original video node (i.e., OQ).

In the just described example, an intermediate internal node of the `Mutex` type was relied on to relate the nodes which are linked to the original and transcoded video versions in the stream hierarchy. This approach might be a good practice in many situations in which multiple content versions are involved. Such variants namely differ in their encoding but otherwise transport identical content (e.g., in this case exactly the same video fragment); as a result, for bandwidth efficiency reasons, a destination will typically not want to receive multiple of these versions simultaneously. Such behavior can effectively be enforced by grouping the different quality variants of a single flow as the sole children of a `Mutex` node.

Recall from section 4.7 that the client software bears the responsibility for ensuring that the quality grouping node (i.e., the `Mutex` node in the previous example) is included in the stream hierarchy. To alleviate this responsibility, the NILayer support library (see section 3.4) provides constructs through which the client software can register the classes of network traffic whose transported content is available in multiple qualities. Each time the client software initiates the incorporation of a network flow which belongs to one such class, the NILayer will automatically make sure that a quality grouping node is inserted in the stream hierarchy before the representation of the network flow itself is added. By default, an internal node of the `Mutex` type is used as intermediary node, but this behavior can be overridden by the client software when registering the multi-quality network traffic class with the NILayer.

5.3.2 Mode of Operation

To determine whether intercepted video frames (i.e., network packets exchanged on a particular video flow) need to be processed, the video transcoding service exploits its interface to the NIProxy's network traffic shaping framework. More precisely, incoming video frames are transcoded if and only if consultation of the appropriate stream hierarchy indicates that the transcoded version of the video stream to which the intercepted frames belong is currently enabled (i.e., the `discrete` leaf node which represents the transcoded video version is set to its non-zero bandwidth consumption level). As a result, the computational footprint of the service is kept to a minimum since unnecessary transcoding operations are at all times avoided.

5.3.3 NTS Interoperation

The static video transcoding service comprehensively demonstrates the possibility for the NIProxy's network traffic shaping and multimedia service provision facilities to conspire and the potential benefits that are entailed by such cooperation. In particular, section 5.3.1 has illustrated that a NIProxy service is able to alter and extend stream hierarchy instances and hence to influence the NIProxy's bandwidth brokering operations. On the other hand, section 5.3.2 has exemplified that network traffic shaping decisions in turn might effectively direct the operation of NIProxy services, which in this case yielded an increased computational efficiency.

5.3.4 Implementation

The service is implemented according to the cascaded pixel-domain approach to transcoding [Vetro 03]. In this methodology, a video bitstream is transcoded via a decode/encode cascade. Stated differently, the bitstream is first decoded and subsequently completely re-encoded with modified quality settings. The advantages of this category of transcoding algorithms are their ease of implementation and the resulting perceptual quality, which is quite high. Their main drawback however are their computational complexity and low efficiency. The bitstream is transferred from the compressed domain to the pixel-domain and back again, hereby effectively discarding all of the compressed information that is present in the original bitstream. More efficient transcoding schemes attempt to reuse (part of) this compressed data to reduce their computational overhead. As the service was intended as a proof-of-concept rather than a commercial product, transcoding efficiency and performance were not considered to be imperative and hence the choice for the cascaded video transcoding scheme is justified. To perform the actual decoding and encoding, the service employs libavcodec, the multi-platform audio and video codec library of the open source FFMPEG project [FFMPEG 10].

Since video transcoding is a stateful process, a separate transcoder instance is maintained for each individual video stream. In particular, on the discovery of a video stream, the service automatically instantiates a new transcoder and couples it with the just detected flow. Each packet that is subsequently intercepted on this network flow will be processed by this particular transcoder instance. Conversely, when a video stream is classified as terminated, its associated transcoder is destructed and purged.

Chapter 6

Software Architecture

The NIProxy is a completely software-based system that is coded exclusively in the C++ programming language. Other than it running the GNU/Linux operating system, the NIProxy does not require any support or specific functionality from the hardware on which it is installed. This lack of dependence on particular hardware maximizes its deployment possibilities. Stated differently, the NIProxy can be hosted on any networking-enabled and GNU/Linux-based device.

This chapter will shed some light on the NIProxy's implementation by exploring its codebase and high-level software architecture. Two software architecture descriptions will actually be presented. Approximately in the middle of this PhD research, the NIProxy has namely been subjected to a complete redesign and code refactorting. This decision was motivated by the fact that practical experimentation with the original NIProxy realization had uncovered a number of limitations, disadvantages and shortcomings. These could nearly all be attributed to the initial software architecture, which turned out to be too rigid and unintuitive. The refactoring phase halted the progress of the NIProxy for a considerable amount of time (i.e., approximately three months) because the time which was invested in redesign and refactoring could not be devoted to the implementation of new features or improvements. The refactoring decision has nonetheless proven to be the right one. Upon completion, the NIProxy's codebase and design had namely become much more robust, manageable and susceptible to extension. Both maintaining existing NIProxy functionality and introducing new features hence became considerably more straightforward and less cumbersome compared to the initial NIProxy instantiation. As a result, the time that has been spent on refactoring was more than compensated for by the higher pace with which NIProxy code extensions and modifications could afterwards be applied. The refactored version of the NIProxy's software architecture is still in use today and all QoE optimization results that will be presented in part II of this thesis have been produced using this NIProxy revision. For completeness reasons, a sample of the most significant accomplishments of the original NIProxy design will be provided in appendix A.

An important remark in this context is that the redesign applied exclusively to the underlying implementation. The NIProxy's objectives and methodology (as described in chapter 3) were determined at the very beginning of this PhD period and were carefully retained during the redesign and refactoring process. Stated differently, the goal of the refactoring operation consisted of improving the efficiency of the NIProxy's implementation instead of addressing hiatuses or issues in its methodology and fundamental principles.

6.1 Netfilter-Based Design

The implementation of the initial version of the NIProxy was founded upon the netfilter and iptables frameworks of the GNU/Linux operating system. This section will therefore first introduce both tools and summarize their functionality before exploring the actual software architecture of this NIProxy instantiation.

6.1.1 Netfilter

Netfilter is a software framework which provides packet handling, manipulation, mangling and filtering facilities within the 2.4 and 2.6 series of the GNU/Linux kernel [Netfilter 10][Bandel 01]. It is licensed under the GNU General Public License (GPL) and was introduced in 1998 as the redesigned and heavily improved (for instance, in terms of security) successor of the GNU/Linux 2.2 ipchains and GNU/Linux 2.0 ipfwadm tools. The netfilter framework was devised and designed by Rusty Russell, based on his experience as author and driving force of the ipchains predecessor.

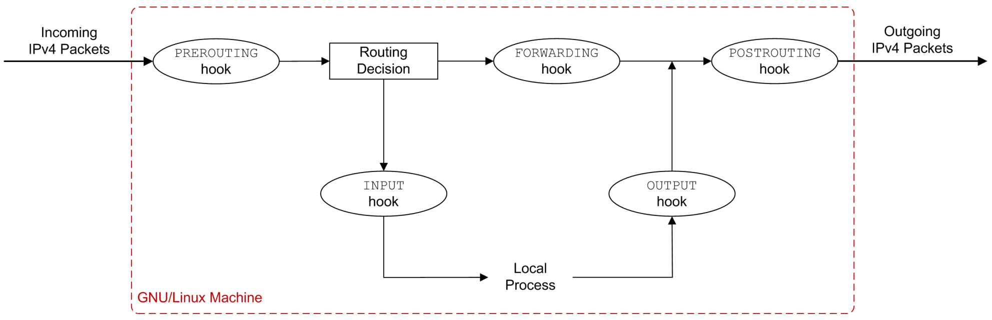


Figure 6.1: Netfilter packet traversal diagram for the IPv4 protocol stack [Netfilter 10].

For each network protocol, netfilter defines a set of *hooks* for which kernel modules can register interest in the form of a callback function. Each hook corresponds to a well-defined location in a packet's traversal of a particular network stack. As an example, the different hooks which are defined for the IPv4 protocol are depicted in Figure 6.1. This figure illustrates that

- **incoming** IPv4 packets which are destined for a local process will always first cross the `prerouting` hook and subsequently the `input` hook before they will be handed over to the receiving application
- IPv4 packets that are not addressed to the local GNU/Linux machine but instead need to be **forwarded** (i.e., routed) will successively traverse the `prerouting`, `forward` and `postrouting` hooks
- IPv4 packets that are generated and **transmitted** by a local process need to successfully pass the `output` and `postrouting` hooks before they will be output on the transmission channel

Whenever a network packet arrives at a protocol hook, the network protocol will invoke the netfilter framework with the packet and the hook number as parameters. Netfilter will subsequently verify whether interest has been registered for this particular protocol and hook. If so, the associated callback function will be launched. In case multiple callback functions were installed for the same protocol and hook, these will be triggered consecutively, in order of registration. Each callback function is allowed to inspect and even alter the packets which it is handed over at its own discretion. After it finishes its processing, the callback must return the (possibly modified) network packet to the netfilter framework and must specify further packet treatment by issuing a *verdict*. The most straightforward verdicts are `NF_ACCEPT` and `NF_DROP` which instruct netfilter to let the network packet continue its traversal of the network stack and to discard it at the current hook, respectively.

All operations that have been described thus far are performed entirely inside the GNU/Linux kernel. Netfilter however also supports network packet treatment in userspace via the `NF_QUEUE` verdict. In case a (kernelspace) callback function issues this verdict for a certain network packet, netfilter will asynchronously transfer it (i.e., using a FIFO queue) from the kernel to a userspace application for further processing. This functionality is provided because kernel module coding requires considerable skill and attention from the programmer. These modules namely run directly inside the GNU/Linux kernel, which implies that incorrect module behavior caused by, for instance, dangling pointers or improper memory addressing might compromise the stability of the entire operating system and can hence have a disastrous impact. In contrast, implementing userspace applications is much more programmer-friendly and is hence likely to be the preferred approach in case the packet handling involves complex (i.e., everything but straightforward) logic. Analogous to a netfilter kernel module, the userspace application must accompany the network packet with a verdict and subsequently return it to the netfilter framework once it has finished its processing. Keep in mind however that there will always be a minor speed penalty associated with transferring packets from the kernel to userspace and that a `NF_QUEUE`-based strategy might hence not be a valid alternative in absolutely time-critical scenarios.

To exemplify netfilter's mode of operation, suppose that a kernel module has been implemented which discards packets on the basis of some classification method and that this module registers interest for the IPv4 `input` hook. As a result, arriving IPv4 packets that match the rules in the classification scheme will be assigned a `NF_DROP` verdict and will hence never be delivered to the local process for which they were originally destined.

6.1.2 iptables

Using callback functions that are implemented in kernel modules to perform operations on network packets is a non-trivial approach which might only be suitable for at least moderately seasoned GNU/Linux administrators. To also open up (part of) the netfilter functionality to typical GNU/Linux users, a generic table mechanism called iptables has been developed on top of it which facilitates IPv4 network packet selection in userspace [Netfilter 10, Bandel 01, Underwood 01]. iptables defines multiple table structures which each enforce packet selection for a specific purpose. Examples include the "filter" table (which is employed for IPv4 packet filtering) and the "nat" table (which is used during Network Address Translation). A table in iptables consists of a number of *chains*, which in turn are checklists of *rules*. The "filter" table for instance encompasses three built-in chains: `input`, `forward` and `output`. These chains are respectively responsible for filtering network packets which are destined for a local socket, packets that are being relayed through the GNU/Linux machine and locally-generated packets. Looking again at Figure 6.1, it can be seen that these chains conceptually coincide with the homonymic hooks in the underlying netfilter framework. As a result, each time an IPv4 packet arrives at one of these hooks, the corresponding chain will be consulted to determine the fate of the packet.

An iptables chain consists of zero or more rules, which each specify how a network packet should be handled in case its header matches the conditions that are specified in the rule. Available rule verdicts include `NF_ACCEPT`, `NF_DROP` and `NF_QUEUE`. These verdicts are semantically identical to their namesakes in the netfilter infrastructure. Other possible rule verdicts are a jump from the current chain to another (user-defined) chain inside the same table as well as the command to resume packet matching at the following rule in the higher-layer (i.e., invoking) chain (through the `NF_RETURN` keyword). These latter verdict types enable a hierarchical packet selection solution, for instance by allowing for the encapsulation of related rules in a dedicated chain.

When a chain is invoked, the conditions that are specified by its constituting rules will be checked consecutively against the header of the network packet under consideration. In case the check fails, the next rule in the chain will be consulted. On the other hand, as soon as a match occurs, iptables will execute the verdict that is associated with the current rule. In case the end of a chain is reached (i.e., no rule in the current chain matches), the default verdict as specified by the chain's policy will determine the fate of the packet.

By default, the tables contain only the built-in chains. iptables however provides a (command-line) front-end through which these can be supplemented with user-defined chains. This interface in addition allows users to insert new rules in chains, to remove existing rules from them, to modify the policy of chains, to inspect table information, etcetera. As such, iptables is a user-friendly solution that unlocks IPv4 packet selection on the GNU/Linux operating system for kernel programming agnostics.

As a concluding remark, iptables counterparts are available for network protocols other than IPv4. In particular, ip6tables and arptables enable packet selection for respectively IPv6 and the Address Resolution Protocol (ARP). Finally, ebtables provides packet filtering and manipulation options at the datalink layer (i.e., for Ethernet frames). Given the NIProxy's focus on IPv4-based telecommunications networks, these counterparts will not be elaborated on; the interested reader is referred to the netfilter/iptables homepage [Netfilter 10].

6.1.3 Implementation

The initial variant of the NIProxy was implemented as a GNU/Linux userspace process which relied on the iptables framework (and hence, indirectly, the netfilter framework) for packet interception [Wijnants 05b, Wijnants 05c, Wijnants 06]. In particular, the iptables tool served a dual purpose. First of all, it was exploited to directly enforce trivial packet filtering and routing behavior on the machine on which the NIProxy was deployed. As an example, to guarantee the generation of reliable latency and throughput estimates, the NIProxy's active network probing framework (see section 3.1.1) requires a minimization of the delay that its network communication incurs from components other than the probed communication channel. This behavior was achieved by inserting rules into the `input` and `output` chains of the “filter” table which specified that NIProxy-initiated network probes as well as the responses that are solicited by these probes should be provided immediate passage (by issuing a `NF_ACCEPT` verdict). The second application of the iptables functionality was to `NF_QUEUE` all network packets which actually belonged to the communication session of one of the NIProxy's currently connected clients to the userspace process.

The software architecture of the userspace process is schematically summarized in Figure 6.2. The NIProxy userspace process was functionally decomposed into two major subsystems: a generic base layer and a plug-in mechanism. The generic base layer clustered several general functions and was completely application-agnostic. The plug-in subsystem on the other hand ensured NIProxy extensibility as it enabled the run-time inclusion of extra (possibly application-specific) functionality. This section will proceed with a description of the exact responsibilities of both subsystems.

Generic Base Layer

The generic base layer formed the heart of the NIProxy's initial software architecture. On the one hand, it encompassed basic support code that is likely to recur in any implementation of a proxy entity. An example is user management functionality. The base layer therefore provided modules for client connection establishment and for performing client-NIProxy communication. On the other hand, the base layer included functionality that was tailored to the NIProxy's specific objectives, methodology and design approach. In this latter category, it was responsible for:

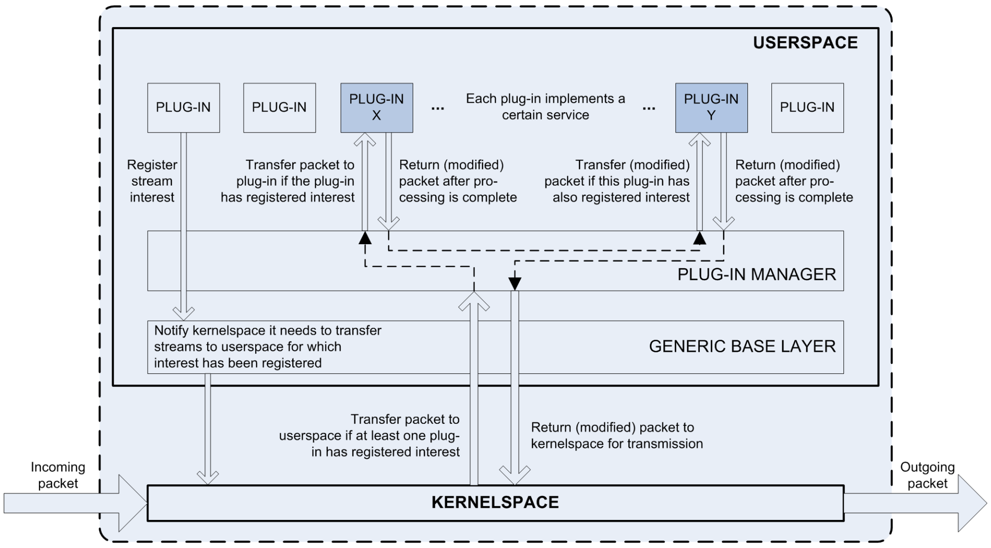

USERSPACE
PLUG-IN
PLUG-IN
PLUG-IN X
...
Each plug-in implements a certain service
...
PLUG-IN Y
PLUG-IN
Register stream interest
Transfer packet to plug-in if the plug-in has registered interest
Return (modified) packet after processing is complete
Transfer (modified) packet if this plug-in has also registered interest
Return (modified) packet after processing is complete
PLUG-IN MANAGER
Notify kernelspace it needs to transfer streams to userspace for which interest has been registered
GENERIC BASE LAYER
Transfer packet to userspace if at least one plug-in has registered interest
Return (modified) packet to kernelspace for transmission
Incoming packet
KERNELSPACE
Outgoing packet

Figure 6.2: Schematic overview of the software architecture of the initial version of the NIProxy.

- Receiving packets from kernelspace and for reinjecting packets in the kernel after they had been processed by the NIProxy userspace process. As is depicted in Figure 6.2, the base layer guaranteed that an intercepted network packet was transferred to userspace if and only if the NIProxy instance was actually interested in it (i.e., in case the packet belonged to the communication session of a currently connected client); other packets simply flowed through the NIProxy without modification and even without ever leaving kernelspace. To control which packets needed to be transferred to userspace, the iptables framework was leveraged.

- Collecting and managing the NIProxy's contextual knowledge. Notice that this was limited to network and application awareness, as terminal intelligence was not yet supported at this stage of the NIProxy's lifecycle.

Plug-in Subsystem

The central component of the plug-in subsystem was the *plug-in manager* which enabled and managed the installation of *plug-ins* on top of the generic base layer. Firstly, plug-ins were able to register interest for certain network streams with the plug-in manager. After storing such stream interest information locally, the plug-in manager relayed it to the generic base layer, where it was translated into an appropriate iptables command to ensure that packets which were exchanged on the specified flow would start to be `NF_QUEUE`-ed to userspace. Secondly, when a network packet was actually transferred to userspace, it was the responsibility of the plug-in manager to dispatch it to the plug-in which had specified interest for the flow which this packet was logically part of. If more than one such plug-in was installed, the plug-in manager sequentially delivered the packet to all those plug-ins, hereby following the order in which they had registered interest for the stream. After a plug-in had finished processing a network packet, it needed to return the (possibly modified) packet to the plug-in manager together with a verdict. The plug-in manager only continued the traversal of any subsequent interested plug-ins in case the just consulted plug-in returned a `NF_ACCEPT` decision; otherwise, the packet was immediately dropped by issuing a `NF_DROP` iptables verdict. Finally, as soon as the last plug-in in the sequence of interested plug-ins was reached, the plug-in manager made sure that the network packet and the verdict that was returned by this plug-in were handed over to the generic base layer for reinsertion into kernelspace.

To further clarify the operation of the plug-in manager, consider the example scenario which is depicted in Figure 6.2. In this use case, two separate plug-ins (i.e., the highlighted plug-ins X and Y) have registered interest for the network flow which the incoming packet belongs to. However, plug-in X was the first to do so. Consequently, the plug-in manager would start traversing the sequence of interested plug-ins by handing the intercepted packet to plug-in X. Upon completion of its processing (during which the network packet could possibly have been modified), plug-in X issues a `NF_ACCEPT` verdict. As a result, the plug-in manager would subsequently dispatch the (potentially altered) network packet to plug-in Y. Since this is the last interested plug-in, the packet and the verdict returned by plug-in Y would be reinserted into the kernel.

The just presented scenario clearly exemplifies that the implementation of the plug-in subsystem allowed for multiple plug-ins to collaborate on a single network stream. Of course, cooperating NIProxy plug-ins needed to be carefully geared to each other. For instance, in case plug-in X had issued a `NF_DROP` instead of a `NF_ACCEPT` verdict in the previous example, plug-in Y would have never received the packet, even though it had registered interest for it.

An important advantage of the developed design for this subsystem was that it enabled the provision of generic as well as application-aware plug-ins. To be more precise, although the plug-ins physically resided inside the network (i.e., on the machines which hosted the NIProxy instances), they could conceptually be considered as forming a part of the distributed application which they serviced. Consequently, in contrast to the base layer, plug-ins were not required to be generic but instead could leverage application-specific knowledge as well as provide application-dependent functionality. For instance, in section A.1.4 the implementation of a video transcoding plug-in for the initial NIProxy instantiation will be presented which was targeted at a particular Networked Virtual Environment (NVE) application. This application supports real-time video communication between users but requires them, for reasons of scalability, to stream not just one but three predefined qualities of the images captured by their webcam. The video transcoding plug-in was implemented to relieve NVE clients from this burden. In particular, the plug-in generated the two remaining video qualities that are required by the NVE by on-the-fly transcoding the highest quality version of a user's webcam input. As a result, scarce client upstream bandwidth was economized since it now sufficed for NVE clients to transmit only this latter fidelity of their video stream. Sections A.1.7 and A.2.2 in appendix A will present condensed descriptions, including experimental results, of two additional examples of application-aware plug-ins for the initial NIProxy implementation.

6.2 Refactored Design

6.2.1 Motivation

Practical experimentation with the netfilter-based NIProxy implementation revealed a number of limitations, disadvantages and shortcomings. A crucial issue turned out to be the rigidity of its underlying software architecture. As development progressed, introducing new functionality and features became ever more cumbersome and time-consuming; in addition, it began to require increasing numbers of implementational anomalies and inconsistencies, which in turn gave rise to maintainability issues. A second substantial drawback of the initial design was the absence of a clear-cut location for the NIProxy's network traffic shaping functionality. Rather than it being an integral part of the software architecture, the NTS algorithm was integrated in the NIProxy by encapsulating it in a plug-in. This plug-in registered interest for all possible data streams and, for each intercepted network packet, issued either a `NF_ACCEPT` or `NF_DROP` decision after consultation of the current bandwidth brokering strategy for the client which the packet was destined for. Besides this solution being hardly optimal from a software engineering perspective, the NTS plug-in often needed to be amended on a per application basis (by implementing a customized plug-in that extended or even completely overrode the original). Furthermore, collaboration between the network traffic shaping functionality and other NIProxy plug-ins was not optimally supported, which rendered the NIProxy's dual QoE optimization techniques somewhat detached and incompatible.

Being faced with these observations, the alternatives were either to continue working with the netfilter-based software architecture and to cope with its shortcomings as best as possible, or to execute a profound redesign. The latter alternative is obviously the recommended long-term solution since it structurally addresses the identified issues instead of merely working around them. On the downside, redesign and refactoring consume time and result in a period of functional status quo. As the issues with the original software architecture were recognized fairly early in the course of this PhD research, scheduling a refactoring phase was considered to be a justifiable overhead. After all, reworking the NIProxy's software architecture was not only expected to improve the performance and effectiveness of its already available functionality, it was also anticipated to result in considerable time savings during the future implementation of new features and QoE optimization tools in the remainder of the PhD period.

To summarize, after careful deliberation, it was decided to completely rework and refactor the netfilter-based design. The objective of this operation was to improve the flexibility and robustness of the software architecture, without however compromising the general NIProxy methodology (i.e., while staying faithful to the majority of its chief ideas and principles).

6.2.2 Implementation

Figure 6.3 depicts a high-level blueprint of the NIProxy's refactored software architecture [Wijnants 07][Wijnants 09a]. As is illustrated, the software architecture on the one hand comprises a number of static components that are not related to a particular client. Depicted are the **Network Probing**, **Connection Establishment**, **Client Administration** and **Terminal Awareness Repository** modules. The former contributes to the NIProxy's network-related awareness as it implements the active probing framework which was described in section 3.1.1. The Connection Establishment module is responsible for processing incoming connection requests and for completing the client log in procedure, while the Client Administration module manages the list of currently logged in clients (which are, conform to the terminology that was outlined in section 1.3, labeled NIProxy clients in Figure 6.3). Finally, the Terminal Awareness Repository component centralizes the NIProxy's terminal and user preferences knowledge (see section 3.1.3). It is implemented as

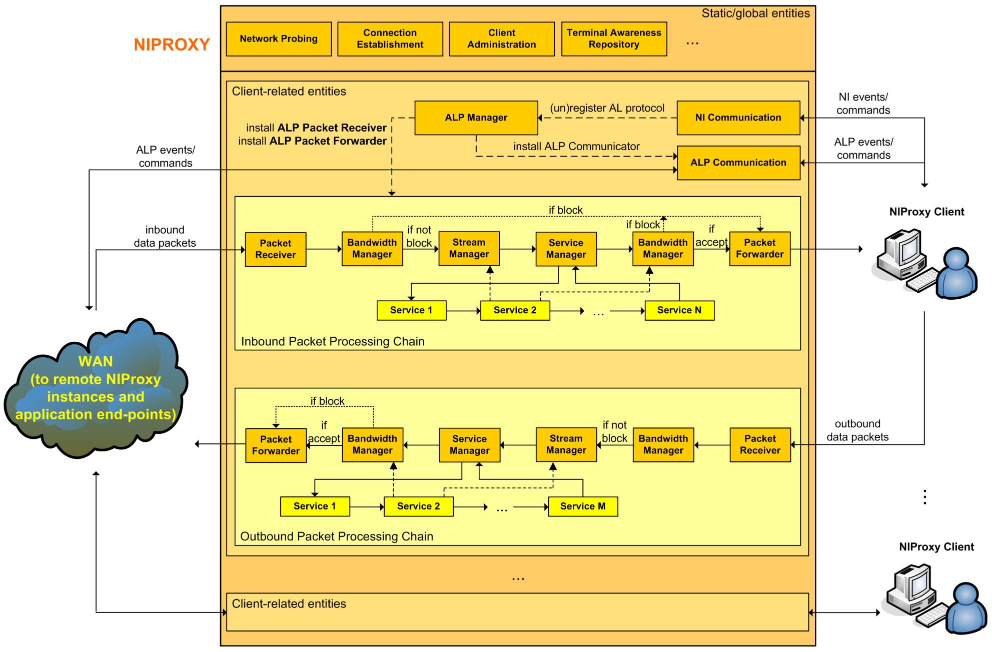


Figure 6.3: Schema of the software architecture of the refactored NIProxy implementation.

a singleton repository which maintains terminal information for all active NIProxy clients and is globally accessible in the NIProxy's software architecture.

On the other hand, the design also includes a *packet processing chain* (PPC), a composite object whose instances are always linked to a single NIProxy client. An *inbound* as well as *outbound* version of the PPC exists, both of which are instantiated immediately after a client has successfully logged in to the NIProxy. Network packets which are intercepted from the Wide Area Network (WAN) are fed to the inbound chain that is coupled with the NIProxy client for which they are destined. Conversely, network traffic that is generated by a NIProxy client itself ought to traverse the client's associated outbound PPC. Notice from these descriptions that the inbound and outbound denominations match the flow direction terminology from section 3.3. As its name insinuates and as will be described more precisely next, each PPC instance adequately processes the network packets that flow through it. During their passage, packets are accompanied by a description of the network flow on which they are transmitted. After successful chain traversal, the (possibly transformed) network packet will be forwarded to its actual destination.

The packet processing chain is a compound object that links together multiple software components [Wijnants 07]:

Packet Receiver Ensures that all necessary network packets will be received by the chain as input. In case of the inbound PPC, this component will intercept the packets that belong to flows in which the associated NIProxy client is currently interested (i.e., that are destined for this client). Analogously, the Packet Receiver of the outbound PPC will seize all network packets that are emitted by the corresponding NIProxy client.

Bandwidth Manager Implements network traffic shaping and is responsible for enforcing the bandwidth arbitration strategy that is devised for the associated NIProxy client.

Stream Manager Performs network stream bookkeeping. Its principal task is to record the bandwidth requirement of individual network flows. This is achieved by simply adding the size of the intercepted packet to the current bandwidth consumption value of the network flow that is identified by the stream description which accompanies the packet. Flow bandwidth consumption is measured and aggregated over a configurable time interval; typically an interval of one second is used to yield per second bandwidth consumption information. Besides stream bandwidth usage monitoring, the Stream Manager stores all other kinds of network-related information and is, for instance, responsible for signaling other components of the detection of the instantiation of a new flow and for keeping track of stream aliveness. The latter is achieved by governing an expiration timer for every active network flow which is reset each time a packet is intercepted on it; as soon as a timer expires, a stream termination signal will be emitted. Notice that this component hence plays an important role in the stream termination notification mechanism that was described in section 4.7.2. Also remark that, combined with the Network Probing module, the Stream Manager constitutes the NIProxy's repository of network-related context.

Service Manager Manages the loading and unloading of services on behalf of its associated NIProxy client and applies the currently loaded services to the network packets which it receives.

Packet Forwarder Transmits completely processed packets to their destination.

Of these components, the Bandwidth Manager and the Service Manager are the most important ones since they embody the two network traffic engineering techniques which are currently provided by the NIProxy. This section will therefore continue by describing the implementation and the mode of operation of these entities in more detail.

Bandwidth Manager

The Bandwidth Manager encapsulates the NIProxy's network traffic shaping algorithm that has been described in chapter 4. As a result, it is responsible for maintaining and managing its associated client's stream hierarchy and for executing the bandwidth distribution process. In the inbound PPC, this component will manage the NIProxy client's *downstream* bandwidth. In the outbound counterpart, it is the *upstream* client bandwidth consumption that will be shaped.

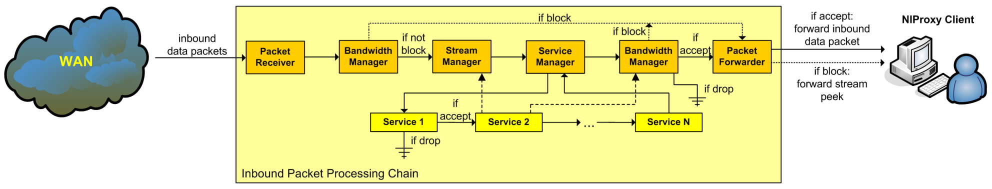
Every time the Bandwidth Manager is handed over a network packet, it consults the appropriate (i.e., downstream or upstream) stream hierarchy to compute a verdict for it. Three possible packet verdicts are defined:

`block` Indicates that the packet belongs to a network flow that the NTS framework is not yet familiar with.

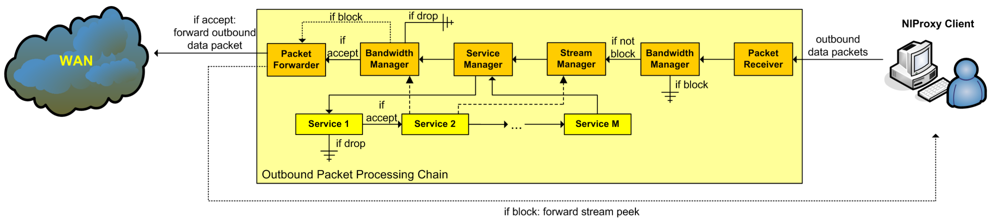
`accept` Indicates that the network stream on which the packet is transferred is known to the bandwidth distribution algorithm and that the algorithm in addition has decided to allocate bandwidth to this stream.

`drop` Indicates exactly the opposite as an `accept` decision (i.e., the packet is transported on a recognized network stream which should currently not be allocated bandwidth).

As is illustrated in Figure 6.4, the Bandwidth Manager actually appears twice in both the inbound and outbound packet processing chain. The first time it is accessed, it simply ensures that a `block`-ed network packet does not continue its traversal of the chain. In the inbound variant, prior to terminating the PPC execution for the `block`-ed packet, a *stream peek* will be transmitted to the sink (i.e., the NIProxy client) by the Packet Forwarder component. The peek includes information which identifies the network flow to which the intercepted packet belongs as well as a limited amount of data from the packet's payload. Combined, the inbound Bandwidth Manager's first appearance and the Packet Forwarder entity hence implement the NIProxy's auxiliary blocked stream notification mechanism which was described



(a) Inbound PPC



(b) Outbound PPC

Figure 6.4: Inter-component packet flow conditions in the packet processing chain.

in section 4.7.2. Remember from this previous discussion that, based on the information that is available in the received peek, the client is responsible for determining the way future packets on the `block`-ed network stream should be treated by the NIProxy. Also notice that the blocked stream notification system is omitted at the corresponding location in the outbound PPC. In the outbound case, the NIProxy client has knowledge of and complete control over the involved network flows (since it is the client itself that emits them). As a result, it is assumed that the client explicitly makes the NIProxy aware of each of its outbound flows and explicitly specifies how each should be handled by the network traffic shaping scheme. Like all client-NIProxy communication, the exchange of stream peeks as well as client-specified hints regarding the desired future treatment of `block`-ed network flows adheres to the NI protocol.

The second occurrence of the Bandwidth Manager fulfills an identical task in both the inbound and outbound implementation of the PPC. In particular, it guarantees that only network packets for which an `accept` decision is computed will be output on the appropriate network link for transmission to their destination. Stated differently, only packets which are conceptually part of a network flow for which the NTS algorithm has reserved capacity will be allowed to actually consume bandwidth. Packets which are assigned a `drop` verdict are not relayed to the Packet Forwarder but instead are discarded at this stage in the packet processing chain to prevent them from illegally claiming either downstream or upstream client bandwidth. Finally, `block`-ed packets will also be discarded and instead, analogous to the behavior during the first encounter of the Bandwidth Manager component in the inbound PPC, a stream peek will be forwarded to the NIProxy client.

Service Manager

Besides bandwidth mediation, the NIProxy also supports multimedia service provision as a traffic engineering technique. In the refactored software architecture, this functionality is provided by the Service Manager component. As was previously discussed in section 5.1, services are implemented as NIProxy plug-ins. The Service Manager is first of all able to dynamically load and unload these plug-ins during NIProxy operation. This behavior can either be triggered by the NIProxy itself or could be initiated by a NI protocol command that was issued by the NIProxy client. Secondly, the Service Manager performs stream interest accounting. In particular, it provides an interface through which services can register (and unregister) interest for one or multiple network flows. A service could, for instance, automatically register its stream interest at load-time. Third and most importantly, the Service Manager is responsible for directing the application of services on intercepted network flows. Whenever a network packet is delivered to the Service Manager, it will consecutively pass it to the currently installed services that have registered interest for the network stream on which the packet is transported. The service traversal order will hereby equal the sequence in which the services have registered interest for the network flow that is currently being processed.

As soon as a plug-in is handed over a packet, it can process it any way it seems fit. This implies that it is allowed for a NIProxy service to alter (part of) the content of the packets which it receives or to even completely replace it (e.g., in case of a video frame, by transcoding it to a lower quality). Besides the network packet itself, it is also possible for services to modify the description which accompanies the packet and which identifies the network flow to which it belongs. This facility is, for instance, valuable for services which actively transform a network flow. As an example, the static video transcoding service from section 5.3 updates the associated stream description to indicate that the output packet no longer belongs to the original video flow but instead its transcoded variant. By doing so, any subsequent services as well as components that succeed the Service Manager in the PPC will be able to ascertain that they are dealing with a transcoded video packet. Another possibility for a service is to specify that the processing of the input packet by subsequent services, if any, should be aborted and that the Service Manager

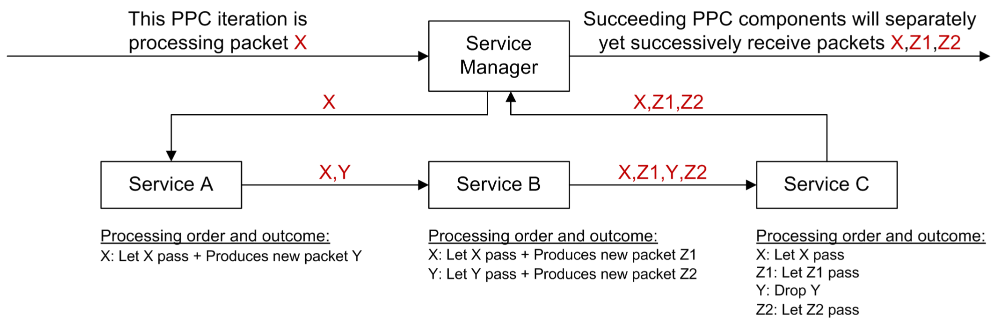

This PPC iteration is processing packet X
Service Manager
Succeeding PPC components will separately yet successively receive packets X,Z1,Z2
X
X,Z1,Z2
Service A
X,Y
Service B
X,Z1,Y,Z2
Service C
Processing order and outcome:
X: Let X pass + Produces new packet Y
Processing order and outcome:
X: Let X pass + Produces new packet Z1
Y: Let Y pass + Produces new packet Z2
Processing order and outcome:
X: Let X pass
Z1: Let Z1 pass
Y: Drop Y
Z2: Let Z2 pass

Figure 6.5: Clarification of the packet flow within a NIProxy service chain.

should drop the packet to halt its processing and to purge it from the PPC. In Figure 6.4, this option is only visualized for “Service 1” to prevent the scheme from becoming needlessly complex, but it is indeed available for all NIProxy services. Finally, services are allowed to spawn completely new packets during their operation and hence to produce multiple packets as output.

Once a service has completed the processing of the current network packet, its produced output (i.e., the possibly altered input packet combined with its associated stream description, as well as any newly generated packets and their stream descriptions) will be transfered by the Service Manager to the next interested service. As such, the Service Manager implements the *service chaining* principle from section 5.1 and hence enables inter-service collaboration. In case a service produces multiple output packets, these will sequentially flow through the remainder of the list of interested services (and afterwards one at a time through the subsequent components in the current PPC iteration) as is exemplified in Figure 6.5.

By now it should be apparent why the Bandwidth Manager component is consulted twice during a single PPC iteration. Its first occurrence ensures that the processing is ceased for packets which are transmitted on network flows that have not yet been incorporated in the NIProxy’s network traffic shaping strategy for the corresponding client. Actual decisions regarding bandwidth allocation however have to be deferred at this stage of the PPC since the intercepted packet still needs to pass through the client’s service list. Services are namely allowed to modify input packets so that they logically start belonging to a different flow. In addition, services might spawn completely new packets and it will need to be confirmed whether or not these packets are entitled to transmission bandwidth. Therefore, the Bandwidth Manager is contacted again after service processing is complete, at which point the client’s current bandwidth allocation scheme is actually imposed. Services’ ability to conceptually transfer packets between network flows (by updating their associated stream description) and to spawn additional network packets also explains the presence of the blocked stream notification scheme at the second Bandwidth Manager appearance in the outbound packet processing chain (see Figure 6.4(b)). The stream descriptions which are output by the Service Manager no longer necessarily identify network flows which are emitted by the NIProxy client itself. As a result, the client will need to be notified of the existence of these novel outbound streams, so that it can instruct the NIProxy on their required future treatment.

As was stated in section 5.2, a defining property of the NIProxy is the possibility for its two provided end-user QoE optimization tools to conspire. Adequate support for this feature is probably the most significant improvement that is introduced by the refactored software architecture (recall from section 6.2.1 that it was only deficiently supported in the netfilter-based implementation). As is suggested in Figure 6.3 in the form of dashed arrows, NIProxy services are provided with an interface to both the Bandwidth Manager and Stream Manager instances in their encompassing PPC. Through the former interface, services are able to consult the current bandwidth brokering strategy which the NIProxy's network traffic shaping mechanism has formulated for its associated client. This information could, for instance, be applied to optimize service execution efficiency. More importantly, this interface even allows for services to alter or extend the stream hierarchy instance that is maintained for the corresponding client. As such, it becomes possible for services to effectively influence the NIProxy's bandwidth brokering operations. An example hereof was already presented in section 5.3.1 in the context of the NIProxy's static video transcoding service. The second interface on the other hand enables services to access as well as supplement the NIProxy's network-related contextual knowledge. For instance, in case service processing results in a modification of the size of the input packet or in case it introduces additional packets, the interface with the Stream Manager can be exploited to update the bandwidth consumption measurements of the involved network streams accordingly. An illustration of this concept is again provided by the static video transcoding service, which supplies the Stream Manager with information regarding the bandwidth requirements of transcoded video flows.

PPC Technicalities

To conclude this section, some minor technical details regarding the implementation of the packet processing chain are succinctly enumerated below:

- Every PPC stands on its own and completely shields its constituting software component instances. In particular, it is impossible for component instances to address components that belong to a different PPC.

- Since the NIProxy maintains separate packet processing chains on a per client basis, each client has personal instances of the PPC's comprised software components.

- The above observation also applies to the Service Manager component and the NIProxy services. As a result, the refactored software architecture enables the NIProxy to provide each client with a personalized service list that is tailored to its specific needs and/or constraints.

- For each client, the inbound and outbound equivalents of the Bandwidth Manager, Stream Manager and Service Manager components correspond to a single instance [Wijnants 09a]. In other words, a client's inbound and outbound PPC share their Bandwidth Manager, Stream Manager and Service Manager instances. In case of the Bandwidth Manager, for example, this instance simultaneously maintains the corresponding client's downstream and upstream stream hierarchy and ensures that the correct variant is consulted depending on the flow direction of the data packets which it is handed over. Similarly, the type of services that are applied by the Service Manager (i.e., either inbound or outbound) is simply determined by the direction of the network flow to which the intercepted network packet belongs.

6.2.3 Application-Layer Protocol Support

As was already discussed in section 3.4 and as is again illustrated in Figure 6.3, the basic communication that occurs between a client and its NIProxy instance follows the Network Intelligence (NI) protocol. This proprietary protocol conceptually introduces a new layer in between the transport and application layers of the TCP/IP protocol stack; it is employed by application software and proxy to establish a connection and to subsequently exchange commands and events. As an example, the NI protocol enables the distributed application to steer the global operation of the NIProxy (e.g., by instructing it to load or unload a specific service) and to provide the NIProxy with application awareness (e.g., through adequate stream hierarchy construction; see section 4.7.1). Recall that at client-side, NI protocol handling is to a large extent performed by the NILayer support library. As a result, only minor modifications to the software of the distributed application will in general be required for the application to be able to start leveraging the NIProxy's features.

Distributed applications however typically implement end-to-end communication and data exchange according to one or more specific protocols which logically operate on top of the NI layer in the protocol stack. To enable QoE optimization, the NIProxy needs to be able to cope with such Application-Layer Protocols (ALPs) and might benefit from knowledge of their internal operation. In the netfilter-based software architecture, this issue was "circumvented" by exploiting the iptables framework to perform packet selection and by devolving the responsibility for application-layer protocol parsing and handling upon NIProxy plug-ins. More specifically, all possible application-layer network flows were simply `NF_QUEUE`-ed to userspace and it was assumed that at least one plug-in contained sufficient knowledge of the application's communication scheme to be able to interpret these flows and to deduce which type of data they carried. As such, the need for explicit application-layer protocol support in the general software architecture was eliminated. The drawback of this approach, besides it being inelegant, was that it yielded an environment in which nearly every plug-in was exclusively tailored to one specific distributed application. In case reuse of plug-in functionality across multiple distributed applications was intended, the plug-in needed to incorporate code which enabled it to understand each individual application's communication semantics.

During the refactoring operation, it was decided that this issue needed to be more properly addressed by making ALP support an integral part of the NIProxy's software architecture. As Figure 6.3 depicts, an **Application-Layer Protocol Manager** (ALP Manager) module was therefore introduced and the NI protocol was extended to allow clients to (un)register ALPs with the NIProxy. On the reception of a request to register a certain application-layer protocol, the ALP Manager performs two actions. It first of all instantiates an **Application-Layer Protocol Communication** (ALP Communication) entity to enable application and proxy to communicate according to the specified application-layer protocol. In other words, the ALP Communication instance bears the responsibility for correctly parsing and processing the ALP messages that are transmitted by the NIProxy client. Secondly, the ALP Manager ensures that the NIProxy is able to intercept and forward data that is exchanged conform to the application's communication scheme. This is achieved through the installation of respectively a new **ALP Packet Receiver** and **ALP Packet Forwarder** element[1]. Unregistering an application-layer pro-

[1]In Figure 6.3, the modules for packet reception and transmission in the inbound as well as outbound PPC lack the ALP prefix in their designation and are represented as a single object. The depicted components actually correspond to wrapper objects which encompass one or multiple ALP Packet Receiver and ALP Packet Forwarder instances, respectively.

tocol on the other hand results in the destruction of the objects that were instantiated on the protocol's registration. Also note that the NIProxy allows clients to have multiple ALPs registered at the same time. This way, support is attained for distributed applications which rely on more than one application-layer protocol as well as scenarios in which a client is running multiple distributed applications concurrently.

As a final remark, notice that the ALP Manager requires knowledge of ALP semantics to be able to instantiate the different components that are necessary for its proxy-side processing. To guarantee maximal flexibility and to achieve run-time extensibility, it should be possible to incorporate support for a particular ALP in the NIProxy without requiring a reboot and without demanding a modification or even recompilation of its codebase. Given the large parallelism with the requirements that were identified for the NIProxy's multimedia service provision mechanism, a similar plug-in-based design was resorted to to address ALP knowledge extensibility. In particular, through the NI protocol, the application software can deliver ALP knowledge to the NIProxy in the form of a dynamically (un)installable plug-in.

6.3 Summary

The NIProxy is a completely software-based entity for the GNU/Linux operating system which is coded exclusively in the C++ programming language. This chapter has explored the NIProxy's implementation and has provided an insight in the evolution of its codebase and software architecture. In its initial design, the NIProxy was implemented as a generic base layer on top of which potentially application-aware plug-ins could be installed. The application-independent base layer only offered basic, proxy-related functionality to maximize NIProxy usability. A plug-in subsystem allowed this generic functionality to be extended with additional capabilities via encapsulation in dynamically loadable plug-ins, which could be either universally applicable or tailored to the needs and characteristics of a particular distributed application. To achieve network packet selection, this NIProxy instantiation relied on the netfilter/iptables framework to manipulate the packet filtering table of the GNU/Linux kernel.

Based on empirical findings, approximately halfway the PhD period the netfilter-based NIProxy implementation underwent a vigorous refactoring. Experimental evaluation had namely unveiled a number of constraints and shortcomings which could practically integrally be attributed to the NIProxy's initial design. Also, due to the software architecture lacking flexibility, introducing new functionality became increasingly cumbersome, time-consuming and unintuitive. The objective of the refactoring was therefore to improve the design and robustness of the software architecture, however without renouncing the global NIProxy methodology or any of the NIProxy's founding principles in the process.

The redesign resulted in an increased functional decomposition and hence a much more modularized software architecture. By more closely complying with the separation of concerns paradigm [Sommerville 06], the maintainability of the refactored NIProxy implementation was drastically improved. In the new design, which is still in use at the time of writing, the NIProxy provides both an inbound and outbound packet processing chain (PPC) instance for each currently connected client. These composite objects group together a number of software modules and define the way data flows between them. Combined, the constituting software components are responsible for processing the data packets that are destined for a NIProxy-managed application end-point (in the inbound PPC) as well as those which are transmitted by the end-point itself (in the outbound case). Due to the high level of modularity, the current software architecture enables the implementation of a particular constituting component to be swapped without impacting any of the other software modules in the PPC.

Distributed over the entire chapter, a number of concrete advantages of the refactored software architecture over the original implementation has been highlighted. The most important benefits are recapitulated in the following enumeration:

- In the initial design, the NIProxy relied on the netfilter/iptables framework to perform packet selection and mangling. Since this framework is only available on the GNU/Linux operating system, NIProxy deployment was limited to network nodes running GNU/Linux. The refactoring operation eliminated this dependency, this way opening up the possibility to port the NIProxy to other platforms and operating systems.

- The refactored software architecture exhibits improved robustness, maintainability, flexibility and extensibility. This not only beneficially impacted the performance and effectiveness of the functionality which was already available prior to the refactoring, it in addition enables accelerated incorporation of new NIProxy features.

- Both versions of the NIProxy implementation supported the multimedia service provision principle. However, whereas in the initial design a service corresponded to a global entity which applied to all clients simultaneously, the NIProxy now provides personal service instances on a per client basis. The drawback of the former scheme is that it required each service to perform at least a moderate amount of client administration and accounting. This in turn gave rise to undesirable service code bloat. In contrast, in the current design, client administration and accounting is tackled centrally (i.e., by the Service Manager).

- The netfilter-based design was missing adequate application-layer protocol support. Instead of explicitly addressing this issue in the software architecture, the responsibility for ALP parsing and processing was placed with the plug-ins. This design decision again contributed to service code bloat and code duplication since ALP-related functionality possibly needed to be repeated across multiple plug-ins.

- During the redesign, specific attention was given to the proper integration of the network traffic shaping functionality in the software architecture. An indirect yet extremely important effect hereof was the transformation of the NIProxy's dual traffic engineering methods from isolated techniques into fully collaboration-enabled QoE optimization partners. The extensive interoperation possibilities between the NTS and multimedia service provision schemes have turned out to become a distinguishing feature of the NIProxy (see the findings from the comparative literature study that were presented in section 2.4.3 for confirmation).

Given all these improvements, scheduling a refactoring period has shown to be legitimate and definitely not in vain. Stated differently, temporarily suspending the functional evolution of the NIProxy and instead allocating attention to its redesign has proven to be the only viable long-term solution.

Part II

Practical QoE Optimization Results

Overview

After the rather theoretical discussion of the NIProxy and its facilities in the previous part, part II will present the outcomes and findings of a number of practical studies. In particular, each time the NIProxy was extended with new functionality and features for QoE optimization, it was subjected to an experimental evaluation. This part will describe the results of these evaluations in a largely chronological order. Stated differently, the subsequent chapters will each focus on a specific challenge in terms of user experience improvement and will present incrementally more elaborate and sophisticated practical results. Please see section 1.4 for a detailed outline as well as a summarized description of each constituting chapter.

All results that will be provided in this part have been captured objectively, for instance by tracing the traffic that traversed the transportation network during the conducted experiment. For each test, the recorded results will be interpreted analytically, after which conclusions will be drawn regarding their implications on user perceived QoE. Notice that this implies that the presented conclusions will be rather speculative in nature since to date traditional usability research methods (e.g., user surveys) have not yet been exerted to formally confirm them. The achieved experimental results are however of such a magnitude that their beneficial influence on user QoE will be intuitively evident. Nonetheless, since subjective interpretations and human factors play a significant role in QoE optimization, collecting qualitative feedback regarding the NIProxy's performed operations by means of user studies is an important topic of future research.

As a related remark, note that the NIProxy is merely a *framework* for QoE optimization rather than a ready-made solution. The NIProxy offers a number of facilities via which the QoE of users of distributed applications *may* be improved. However, whether an improvement in user QoE is actually achieved largely depends on how these facilities are exactly applied. Stated differently, the NIProxy's behavior needs to be tweaked to the context of use if actual QoE optimization is to be attained; (subjective) user feedback is likely to play a crucial role in this process, which again underscores the importance of submitting the NIProxy to qualitative studies.

The NIProxy being a *framework* for QoE optimization also implies that the evaluations that will be described in this part are mainly meant to provide the reader with an *idea* of the NIProxy's QoE optimization potential. As a result, the absolute values and the statistical significance of the realized experimental results are of lesser importance. Every chapter will therefore present experimental findings that are *representative* of the NIProxy's competence for addressing the QoE optimization challenge that is under investigation. None of the chapters will include a statistical analysis of the achieved results.

Chapter 7

Reference Scenario and Baseline Results

As was discussed in chapter 6, approximately halfway the course of this doctoral research, the NIProxy has undergone a major refactoring of its software architecture. The redesign operation was followed by an incremental improvement and expansion of the QoE optimization tools that are provided and supported by the NIProxy. This chapter will present the results of a practical evaluation of the NIProxy immediately after completion of the refactoring effort [Wijnants 07]. This particular instantiation of the NIProxy can be considered the post-refactoring reference version: only elementary network traffic shaping functionality was supported and the set of available services was limited to the static video transcoding service that was described in section 5.3. In addition, support for outbound traffic engineering was not yet provided. The QoE optimization results that will be presented in this chapter hence represent baseline achievements which will be further extended and embellished in subsequent chapters (e.g., through the incorporation of new specific functionality in the NIProxy).

7.1 Evaluation Environment

The first practical trial of the revised version of the NIProxy was performed using an in-house developed Networked Virtual Environment (NVE) application. At the time of evaluation, this application was leveraged regularly by our research department to test new ideas and principles regarding network communication. Given its academic background and intended purpose, the NVE application did not set great store on virtual environment and avatar visualization. Users were presented a top-down two-dimensional view of the shared world in which they were embodied by a colored circle with viewing vector (see Figure 7.1). In contrast, a major point of interest of the application was interpersonal communication and in particular the effect of the network traffic that it introduces on the transportation network. Users were therefore provided with real-time voice as well as webcam video streaming capabilities as means to communicate with remote peers. The speex and H.263 codecs were employed for voice and video compression, respectively.

Before the test process could be initiated, the application had to be modified so that it was capable of connecting to the NIProxy and, in a later stage, of providing the NIProxy with application awareness. The availability of the NILayer auxiliary library, which was introduced in section 3.4, turned out to facilitate this task tremendously. The largest obstacle proved to be the NVE application's awareness manager[1]. In particular, due to its straightforward implementation, the awareness manager was incapable of providing sufficiently detailed information regarding the relative importance of the network traffic which was generated by the application. This in turn resulted in a non-optimal operation of the NIProxy. To mitigate this problem, an additional scheme was implemented on top of the awareness manager to enable more fine-grained stream importance determination. In particular, to simulate the attenuation of the strength and hence audibility of a voice signal with increasing distance (as occurs in the physical world), the scheme assigned importance to voice traffic inversely proportionally to the spacing in the virtual world between the local user and the stream source. This implies that as a remote user was virtually positioned more distantly from the local user, the

[1]Awareness management in a NVE (and possibly other types of distributed systems) is concerned with identifying which objects and information are of relevance to a particular user [Singhal 99]. As such, it is typically exploited to manage (i.e., restrict) the amount of information that must be received and processed by a client. In this experimental evaluation, client-side manipulation of the NVE's communication scheme by the awareness management system was disabled. Instead, the traffic engineering task was fulfilled by the NIProxy, hereby relying on the client's awareness manager as primary source of application-related intelligence.

latter would attach lower relevance to the former's voice stream. For video traffic on the other hand, the scheme not only resorted to virtual distance but in addition considered virtual orientation information. As a video source moved farther out of the Field of View (FOV) of the local user, an increasing penalty value was subtracted from the importance of its emitted video stream.

It is worth noting that the implemented stream importance determination scheme is not particularly tailored to the considered application and would in fact make more sense for a NVE application which provides users with a first-person perspective in an immersive 3D virtual world. It was nonetheless decided to employ this scheme because it allowed for a clear and comprehensive investigation of the NIProxy's added value in terms of user QoE optimization.

7.2 Experiment Description

In the just described evaluation environment, multiple experiments were conducted. This chapter will report on only one such test because the results that were generated during this experiment are representative of the benefits, in terms of user QoE optimization, that were attainable by the initial instantiation of the refactored NIProxy.

Four NVE clients were involved in the experiment, of which only one was connected to a NIProxy instance. The three remaining clients, which will be denoted by C1, C2 and C3, ran the unmodified version of the test application and hence did not leverage any of the NIProxy's functionality. The clients simulated four geographically dispersed residential users which were interconnected by a Wide Area Network (WAN), to which each was connected through an access link. An NIProxy instance was deployed on the transition point where the managed client's last mile connection and the WAN converged. The goal of the experiment hence consisted of investigating the way the NIProxy controlled the last mile dissemination of multimedia data and, more broadly, of registering the multimedia experience that was witnessed by the managed user.

To confine the complexity of the results, only one type of multimedia traffic was considered in the experiment. Uncomplicated baseline results improve intelligibility and, more importantly, enable meaningful comparison with results which will be presented in future chapters. Given the multimedia streaming alternatives supported by the NVE application (i.e., audio and video), it was decided to opt for video because it is more demanding on the transportation network, especially in terms of bandwidth consumption. Resorting to video streaming in addition allowed for the investigation of the NIProxy's multimedia service provision mechanism and, in particular, of the static video

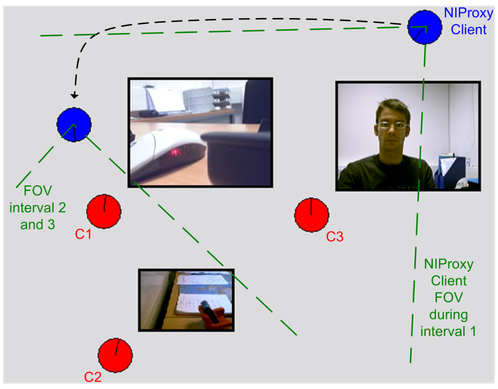


Figure 7.1: Screenshot of the NVE application which illustrates the client positioning in the virtual world during the experiment.

transcoding service from section 5.3, the sole service that was implemented at this point in the NIProxy's lifecycle.

The experiment logically spanned three distinct intervals. Within each interval, all contextual factors were invariant, while every interval transition was triggered by a change in one or more conditions. The client positioning in the virtual world during the experiment is illustrated in Figure 7.1. It appears that in the initial interval, the NIProxy client was oriented towards clients C1, C2 and C3, which were consequently all enclosed in its FOV. After approximately 35 seconds, the NIProxy client navigated along the depicted dashed path to arrive at a new virtual location, at which point the second interval of the experiment commenced. Notice that at this point C3 fell outside the NIProxy client's FOV. Again approximately 40 seconds later, the transition to the third experiment interval was triggered by artificially decreasing the downstream bandwidth that was available to the NIProxy client. In particular, during the third interval the NIProxy client had a downstream bandwidth of only 160 Kilobits per second (Kbps) at its disposal, against 240 Kbps during the first two intervals.

The NIProxy's static video transcoding service was configured to perform video quality as well as resolution reduction. The quality relation between the original and transcoded video variants is exemplified in Figure 7.1, where the video streams of remote clients C1 and C3 on the one hand and client C2 on the other are running at original and transcoded fidelity, respectively.

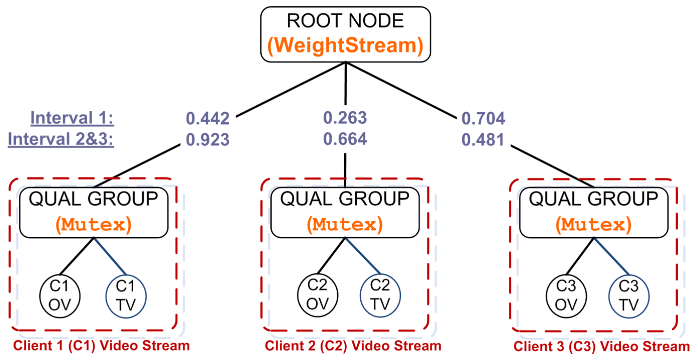


Figure 7.2: Stream hierarchy maintained during the experiment.

7.3 Experimental Results

The stream hierarchy that was maintained for the managed client during the experiment is depicted in Figure 7.2. An internal `WeightStream` node was used to distinguish the remote video sources from each other. The stream hierarchy was constructed entirely by the NIProxy client itself, except for the edges in blue and the leaf nodes with a blue outline, which were added by the NIProxy's static video transcoding service. In particular, each leaf node that was instantiated by the NIProxy client represented the Original Version (OV) of a specific video stream, whereas its sibling corresponded to the Transcoded Version (TV) of this stream. These transcoded counterparts did not belong to the test application's communication scheme but instead were generated automatically by the video transcoding service. Both the OV and TV leaf nodes were of the `discrete` type and defined 2 discrete levels which corresponded to a zero and maximal stream bandwidth consumption, respectively. As was explained in section 4.3.1, the leaf nodes were in this experiment hence confined to toggling their associated network flow between an off and on state. Finally, in line with the reasoning from section 5.3.1, an internal node of type `Mutex` was exploited to cluster the two fidelities of each video stream, this way effectively preventing the NIProxy client from ever receiving both versions simultaneously.

Recall from section 7.1 that virtual distance as well as virtual orientation information served as basis for the client-side calculation of relative stream significance. In the experiment, the outcome of the stream importance determination scheme was translated into appropriate weight values for the subtrees which represented the 3 remote video sources in the stream hierarchy. Notice that in Figure 7.2 all weight values are depicted in pairs. The uppermost value of each pair applied during the first interval of the experiment, while the value below it applied during the two following intervals. The identical weight values for the latter two experiment periods are explained by the fact that no user relocations occurred at the transition from the second to the third interval.

The exact calculation of video source importance, and hence of their corresponding weight values, occurred as follows. In the first experiment interval, all remote clients lay in the NIProxy client's FOV. As a result, stream significance depended solely on the virtual distance between the NIProxy client and the individual sources. Consequently, the video stream transmitted by C3 was assigned the highest weight value, while C2's video stream received the lowest. However, at the end of the first interval, the NIProxy client traveled to a new virtual location, which resulted in shifts in stream importance. In particular, during the second and third experiment period, C1 was considered to be the most important video source because it was now located nearest to the NIProxy client and in addition was contained in its FOV. Furthermore, although the virtual distance between the NIProxy client and clients C2 and C3 was almost identical, C2's video stream was allocated a significantly higher weight value since C3 resided outside of the NIProxy client's virtual viewing angle.

Based on the just described stream hierarchy, the NIProxy client's downstream bandwidth was brokered as is illustrated in Figure 7.3. The depicted network trace indicates which video streams were received by the NIProxy client during the experiment and at which quality, the bandwidth consumed by each and the total amount of downstream bandwidth that was available to the NIProxy client. For instance, in the first interval, the NIProxy client had a downstream bandwidth of 240 Kbps at its disposal and this bandwidth was exerted by the NIProxy for the delivery of

- the original version of the video streams from C1 and C3
- the transcoded version of C2's video stream

This situation is reflected in the screenshot in Figure 7.1. Notice the correspondence between the delivered video qualities and the NIProxy client's stream hierarchy: during the first interval, the subtrees which represented C1's and C3's video stream were assigned a higher weight value compared to the subtree that was associated with C2's video stream.

The network trace that is shown in Figure 7.3 comprehensively demonstrates the capabilities and benefits, in terms of QoE enhancement, of the initial version of the refactored NIProxy. First, it illustrates that the NIProxy delivered an important contribution to congestion avoidance on the

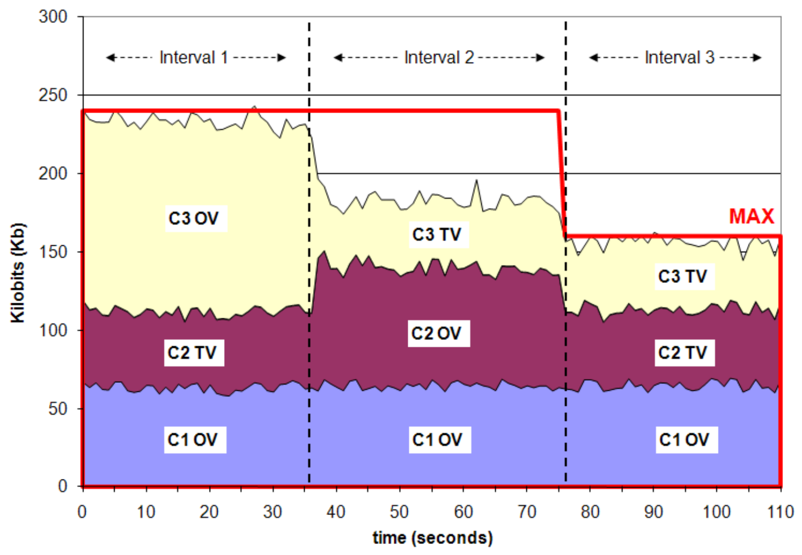


Figure 7.3: Stacked graph which visualizes all video network traffic received by the NIProxy client during the experiment.

managed client's access link, since the downstream capacity of the client's last mile connection was at all times respected. Stated differently, no more traffic was forwarded to the NIProxy client than its network connection could handle. Consequently, packet loss and packet latency were kept to a minimum throughout the experiment, which in turn resulted in the reception of the disseminated multimedia data at client-side under ideal circumstances. The final outcome was an optimal playback of the received video traffic and hence a maximal viewing experience for the user. Secondly, the last mile downstream bandwidth that was actually available to the NIProxy client was partitioned correctly (i.e., in accordance with relative stream significance). In particular, the multimedia flows that the NIProxy client deemed most important were at all times delivered at the highest possible quality, conceivably at the expense of less important network traffic. The two achievements that have been described thus far can be attributed to the NIProxy's NTS mechanism and, more specifically, to the exploitation of respectively network and application awareness during bandwidth brokering. In contrast, the third benefit is related to the NIProxy's service delivery facilities. The presented experimental results namely also exemplify the user QoE optimization options that are unlocked by the static video transcoding service and hence, through generalization, by the NIProxy's multimedia service provision platform. In the absence of such a service, shortage of client downstream bandwidth might force the NIProxy to mercilessly drop specific video flows on a quite regular basis. Now however, it becomes in such situations possible for the NIProxy to still forward (some of) the video streams which would otherwise have needed to be discarded to the managed client, albeit in a lower quality. Notice that this is not merely the merit of the video transcoding functionality that is provided by the service, but to a large extent also of the service's correct interaction with the NTS process. Although no formal user inquiry was conducted, it seems intuitively apparent that the ability to deliver a lower-fidelity video stream instead of no video at all is likely to appeal to (at least a considerable subset of) users of distributed applications. In any case, if the video transcoding behavior and its implications on the bandwidth brokering outcome would not be desired by a particular person, it can easily be disabled for this user as NIProxy services are applied on a per client basis.

For reasons of completeness, this section is concluded with some remarks on the experiment and the presented results:

- Notice from the chart in Figure 7.3 that not all original video streams consumed identical amounts of network bandwidth. For instance, the original version of C3's video stream demanded considerably more bandwidth compared to the video streams that were sent out by C1 and C2. This was caused by the difference in complexity of the streamed video content. In particular, the webcams of clients C1 and C2 recorded a static scene which did not change at all during the experiment, while C3's webcam captured the face of a user (see Figure 7.1).

- The bandwidth consumption of the original video stream did not influence the bitrate of the transcoded version of this stream. As was described in section 5.3, the NIProxy's static video transcoding service needs to be configured with a target bitrate on instantiation. All output video streams that will be generated by a service instance will adhere to the specified target bitrate, irrespective of the bandwidth consumption of the input flows. In this experiment, the target bitrate equaled 50 Kbps.

- A considerable amount of the NIProxy client's downstream bandwidth remained unused during the second interval of the experiment. This is explained by the fact that this idle bandwidth did not suffice to boost C3's video stream to a higher quality and that it could not be exploited in any other way (i.e., all other multimedia flows that were involved in the experiment were already running at maximal quality).

7.4 Conclusions

The refactoring of its software architecture (see chapter 6) represented an important milestone in the NIProxy's lifecycle. Since then, the NIProxy's functionality and capabilities have been further improved and extended. This was however achieved without radically modifying the underlying software architecture. As a result, the version that was yielded by the refactoring operation can be considered to be the most basic and limited instantiation, with regard to supported QoE optimization options, of the NIProxy in its current form (i.e., at the time of writing). Through the discussion of the outcome of a representative experiment, this chapter has investigated the results that could be attained by this initial post-refactoring NIProxy implementation. In other words, this chapter has illustrated the most elementary QoE optimization capabilities that are available in the current revision of the NIProxy. As such, it has provided a point of reference against which the reader will be able to offset improvements and extensions that were introduced after the redesign phase and which will be presented in subsequent chapters.

The presented experimental findings were generated using a top-down 2D NVE application which was leveraged regularly in our research department to investigate, explore and evaluate networking-related concepts at the time of the NIProxy refactoring. In the experiment, a NIProxy instance was responsible for regulating the downstream transmission of NVE-related traffic over a user's access connection. Three important baseline observations were distilled from the produced results:

- By exploiting its network awareness during network traffic shaping, the NIProxy succeeded in preventing over-encumbrance of the user's network connection. As such, it contributed to a stable network environment, which in turn resulted in optimal data delivery.
- The network traffic shaping process was directed by the NIProxy's application awareness, which led to an intelligent distribution, from the user's point of view, of the available downstream access bandwidth over the involved network traffic. In this particular scenario, the user received the flows that he deemed most important at maximal quality, conceivably at the expense of less important network traffic.
- The ability for NIProxy services, in this case exemplified in the form of the static video transcoding service from section 5.3, and the NTS framework to collaboratively address the user QoE optimization task was confirmed, as were the benefits that are potentially unlocked by such collaboration.

Since these findings can be attributed to the most primary functionality of the current revision of the NIProxy, they will recur in a similar form in subsequent chapters. These chapters will however in some way embellish or further extend the baseline results that were already achieved here through the introduction of particular improvements or by addressing a specific facet of user QoE optimization. This implies that subtraction of the baseline achievements from the results that will be presented in future chapters will disclose their specific contributions.

Chapter 8

Combined Real-Time and Non-Real-Time Network Traffic Shaping

Depending on its characteristics, multimedia content is transmitted over the transportation network in either a real-time or non-real-time fashion. Real-time network flows have stringent timing constraints and therefore require timely arrival at the destination. An example is a video fragment in a video streaming environment. Chapter 7 has already confirmed that the NIProxy's NTS framework is able to adequately cope with this class of network traffic. Non-real-time network traffic on the other hand exhibits more relaxed delay constraints but in contrast typically requires perfect and reliable delivery to the destination. These hallmarks are for instance manifested in a file transfer. Although the foundations for managing the latter type of traffic were present in the NIProxy's NTS scheme from the very outset (in the form of the `continuous` leaf node category), empirical findings have unveiled that extra functionality was required if the goal was to achieve sensible results. This chapter is exactly devoted to this topic as it will report on the incorporation of practical support for the shaping of non-real-time network traffic in the NIProxy [Wijnants 08b]. As will become apparent, this involved the adoption of specialized buffering as well as rate control techniques. Through experimental validation, the NIProxy's capability to successfully manage client downstream bandwidth in the presence of competing real-time and non-real-time network traffic will be demonstrated. The attained experimental results will in addition be collated with the default scenario in which the NIProxy is not exploited. This qualitative comparison will expose a substantial improvement in user QoE in case the NIProxy's NTS functionality is leveraged.

8.1 Real-Time versus Non-Real-Time Network Traffic

One possible way to roughly categorize network traffic is to distinguish between *real-time* and *non-real-time* network flows:

- Real-time data flows transport content or media with real-time characteristics, such as interactive audio and video. As a result, real-time network traffic is very sensitive to delay and needs to be delivered to the destination "in time". Outmoded delivery renders the received data worthless and has hence a just as detrimental impact as lost data. Real-time network flows are typically continuous, long-lived streams that are transmitted using a relatively basic and hence lightweight transport- or application-layer protocol such as the User Datagram Protocol (UDP) [Postel 80] or the Real-time Transport Protocol (RTP) [Schulzrinne 03].

- In contrast, non-real-time network traffic does not impose similar strict demands on its delivery period, although in many situations it is still preferable to receive the transported content as soon as possible. On the other hand, while real-time network traffic can typically cope with small amounts of packet loss, non-real-time content should usually be delivered reliably and free of errors. This explains the popularity of more advanced transport-layer protocols like, for instance, the Transmission Control Protocol (TCP) [Postel 81b] for handling the dissemination of non-real-time content. Non-real-time network traffic is often bursty and relatively short-lived in nature and typically carries information such as file or P2P data.

The real-time versus non-real-time classification can also be considered from the perspective of network traffic *elasticity*. Real-time network traffic is typically injected in the network at a certain rate and lacks network- or transport-layer mechanisms to adapt its data throughput and hence bandwidth consumption. Note that this definition does not necessarily imply that the source emits the data at a fixed rate. Streaming a media fragment that was encoded according the the Variable Bitrate (VBR) approach is for instance likely to yield a data flow whose bandwidth consumption fluctuates considerably over time[1]. A real-time flow will however typically not adapt its behavior and bandwidth consumption to the state of the transportation network and is therefore considered to be non-elastic. In contrast, elastic traffic displays the ability to dynamically determine and adjust its transmission rate according to the network's currently prevailing delay and throughput values. As an example, TCP includes a network congestion avoidance scheme that is based on an additive-increase multiplicative-decrease (AIMD) strategy [Tanenbaum 02]. In short, the protocol cautiously increases the transmission rate as long as the network is sensed as being congestion-free, but rapidly reduces its data throughput on congestion detection (in an attempt to guarantee high network performance). Since non-real-time network traffic is disseminated predominantly using TCP, this type of traffic will usually be elastic.

8.2 Implementation

During the NIProxy's design phase, it was already anticipated that the management of real-time and non-real-time network traffic would likely demand considerably dissimilar approaches. To address this issue, the NIProxy's network traffic shaping scheme from the very beginning included different types of leaf nodes (see section 4.3). In particular, the `discrete` leaf node class was developed for the purpose of managing real-time flows, whereas its `continuous` counterpart was envisioned to do the same for non-real-time network traffic. The `discrete` leaf nodes have been found to be competent to fulfill their expected purpose without requiring significant modification or additional support. Empirical evidence has however established this to be untrue for the `continuous` category. This section will elaborate on the complementary functionality that needed to be implemented to enable effective non-real-time network traffic shaping.

[1]VBR-encoded fragments vary the amount of output data per time segment. VBR allows a higher bitrate to be allocated to the more complex segments of media fragments at the expense of segments with lower coding complexity. Adjusting the target bitrate based on the varying complexity of the source data typically improves the overall quality of the encoded bitstream.

8.2.1 Buffering and Rate Control

Recall from section 4.3.2 that a `continuous` leaf node provides constructs which enable it to set the bandwidth consumption of the network flow which it represents in the stream hierarchy to a continuous range of values. This implies however that functionality is required to enforce the calculated bit budget in practice. The implementation of the `continuous` leaf node was therefore complemented with buffering and rate control features as follows:

- the data which the NIProxy intercepts on the associated (non-real-time) network flow is buffered locally
- buffered data is allowed to trickle through at a rate that equals the amount of bandwidth that has been granted to the `continuous` leaf node by the NIProxy's bandwidth distribution algorithm
- processed data (i.e., data that has been forwarded to its destination) is purged from the `continuous` leaf node's local buffer

In effect, the implementation of the `continuous` leaf node class has been extended so that its operation, behavior and real-world impact resemble those of a *leaky bucket* [Tanenbaum 02].

8.2.2 Determination of Maximal Bandwidth Consumption

Another problem that needed to be addressed was the estimation of the maximal bandwidth consumption of non-real-time network traffic. Remember from chapter 4 that this value plays an important role in the NIProxy's bandwidth distribution scheme. For real-time network flows, the issue of maximal bandwidth consumption determination is resolved by registering the rate at which the flow arrives at the NIProxy. A similar straightforward approach could however not be adopted for non-real-time network traffic, since the discussion in section 8.2.1 has established that non-real-time content is possibly buffered by the NIProxy and relayed to its destination at a pace that differs from the rate at which it was intercepted. It was therefore decided to set the maximal bandwidth consumption of a non-real-time network flow to the current amount of data (in bytes) which the NIProxy has already received on this flow, but which has not yet been processed. In other words, the maximal bandwidth usage of `continuous` leaf nodes equals the current amount of data that is stored in their associated buffer. There is one exception to this approach: in case the quantity of buffered data exceeds the throughput of the destination's network connection, the maximal bandwidth consumption will be bounded by the latter value.

8.2.3 Granularity Level

A final implementational dilemma concerned the granularity at which the shaping of non-real-time data was going to be supported. It was decided to opt for a relatively coarse approach in which non-real-time network traffic is managed at flow-level. This implies that all data that is transferred over the same non-real-time network flow will be treated identically by the NIProxy's NTS framework.

An alternative solution would have consisted of incorporating each individual non-real-time content item in the stream hierarchy (i.e., even in case some of these items are conceptually transported on the same non-real-time network stream). Such low-level support is however deemed to be useful and meaningful only to a handful of distributed applications, while it on the contrary will likely be considered as burdensome by the vast majority. In particular, a notable drawback of the fine-grained alternative is that it will rapidly result in the management of the stream hierarchy becoming a complicated or at least tedious task. This is due to the properties of non-real-time network traffic, which is inherently composed of a number of relatively small content objects that are being requested and transmitted in a bursty fashion. Since the responsibility for the management of the stream hierarchy in large measure lies with the client (see section 4.7), controlling the shaping of non-real-time network traffic at a fine granularity would force the client to update its stream hierarchy rather frequently and would impose the need for substantial stream hierarchy bookkeeping at client-side. Besides these disincentives from the client's perspective, an additional undesirable consequence at NIProxy-side is that the network traffic shaping calculations would become more expensive in terms of computational complexity and time consumption due to the increased size of the stream hierarchy. This drawback is further aggravated by the iterative nature of the bandwidth brokering process: as section 4.1 has discussed, it is repeated quite frequently to enable the NIProxy to react to dynamic events in a timely fashion.

It is worth remarking that the subtle level of control that is provided by the fine-grained solution can also be achieved by the implemented flow-level scheme, albeit at the cost of some additional overhead. In particular, in case a distributed application requires control over the bandwidth consumption of individual non-real-time content items, it could transfer each item over a separate non-real-time network flow. As stated previously however, only a limited number of distributed applications is expected to benefit from such low-level command.

8.3 Evaluation

This section harbors representative experimental results that will comprehensively demonstrate that the NIProxy succeeds in shaping non-real-time network traffic. In particular, the bandwidth brokering outcome of two distinct experiments will be described. The objective for the NIProxy was identical in both experiments, namely to mediate the delivery of multimedia content over the final part of an end-user's network connection. The first experiment only involved non-real-time data, whereas in the second experiment the NIProxy needed to manage client downstream bandwidth in the presence of real-time as well as non-real-time network traffic. In both cases, the results that were achieved by the NIProxy's bandwidth brokering mechanism will be equated to the default scenario in which the functionality of the NIProxy was not exploited and it will be investigated whether the introduction of the NIProxy led to an improvement of user QoE.

8.3.1 Test Setup

The evaluation of the NIProxy's NTS framework and, in particular, of its support for the management of non-real-time network traffic, was performed in the context of a simple test application that allowed users to set up both real-time and non-real-time network streams with remote hosts. Video was selected as representative of real-time network traffic because it is the type of real-time multimedia content that is the most demanding in terms of network resources. The test application was implemented so that after a real-time network connection had been set up with some remote host, the latter would immediately start streaming video to the local client (using RTP). In contrast, establishing a non-real-time network connection with a remote host did by itself not result in the local client receiving data on this new connection. The user instead needed to explicitly request some content items (i.e., files) on the non-real-time network stream before the remote host would actually start transmitting data over it. In other words, a simple form of P2P file sharing was imitated to generate the non-real-time network traffic in the experiments. Non-real-time content was exchanged using TCP, since this protocol offers a number of traits which make it very suitable for this kind of communication (e.g., reliable and in-order delivery of transmitted data). Finally, requests for file transmission were serviced sequentially and on a one-by-one basis, in the order in which they were received. Stated differently, the distributed application's policy for responding to file requests was based on the First-In-First-Out (FIFO) principle.

8.3.2 Experiment 1: Managing Non-Real-Time Network Traffic

Experiment Description

The first experiment was conducted to verify the NIProxy's potential to manage client downstream bandwidth in settings that exclusively involve the dissemination of non-real-time multimedia data. To simulate such a scenario, the test application from section 8.3.1 was employed to create two separate P2P TCP connections between a client and a single remote host, which will be referred to as P2P stream 1 and P2P stream 2 in the remainder of this discussion. After the TCP connections had been set up, the following files were requested:

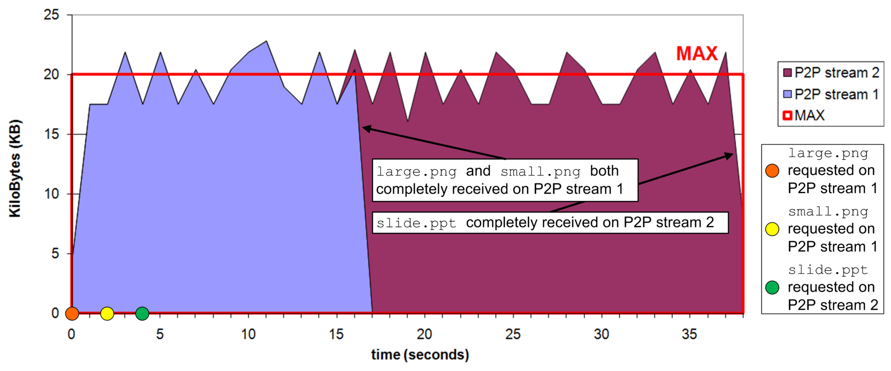
- `large.png`: An image of size 209286 bytes, requested on P2P stream 1
- `small.png`: An image of size 102879 bytes, requested on P2P stream 1
- `slide.ppt`: Slideshow of size 416768 bytes, requested on P2P stream 2

The files were requested in the order in which they are enumerated, with 2 second delay intervals applying between each two consecutive requests. Finally, the client's available downstream bandwidth was constrained to 20 KiloBytes per second (KBps) during the experiment. This may appear an unrealistically low value given today's broadband Internet connections. Enforcing such a limited bandwidth amount however allows for the demonstration to be kept simple and the generated results compact. In addition, in the event of contention from real-time network traffic, it might very well be possible that only such a small fraction of the client's total downstream capacity will be reserved for the reception of non-real-time content.

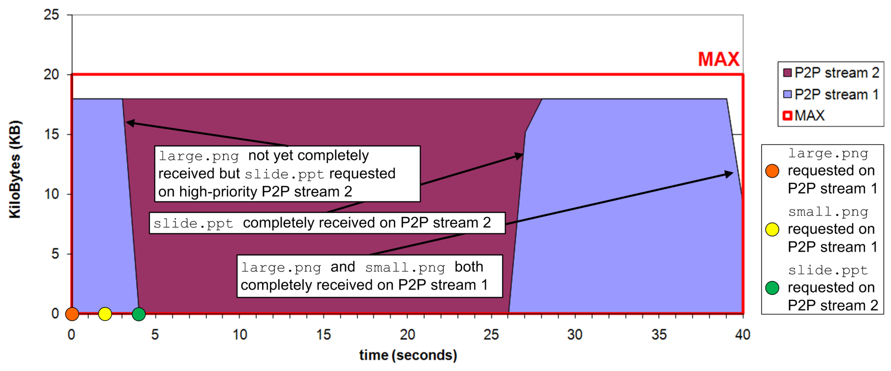
The experiment was repeated three times, each time under different circumstances. First of all, the experiment was executed without involving the NIProxy. Next, the experiment was conducted twice more, with the client's data reception in both cases being subjected to the NIProxy's network traffic shaping operations. The difference between the latter two iterations lay in the type of internal node that was relied on to differentiate between the two involved P2P streams in the client's stream hierarchy. In particular, respectively a `Priority` and `WeightStream` node were employed for this purpose.

Experimental Findings

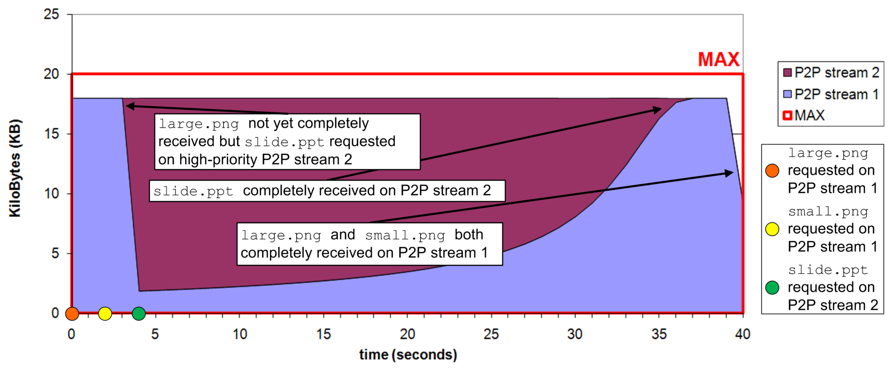
During each of the executions of the experiment, the client-side reception of network traffic was recorded. The results are plotted as stacked graphs in



(a) Without NIProxy

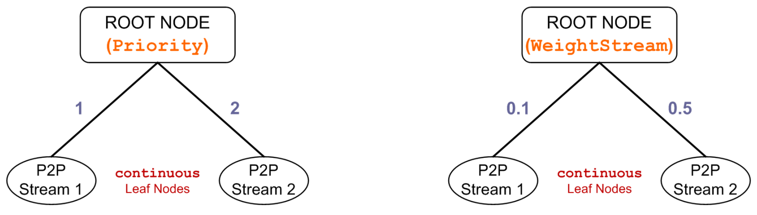


(b) With NIProxy (`Priority` node)



(c) With NIProxy (`WeightStream` node)

Figure 8.1: Stacked graph plots of the media content received by the client during the three different executions of the first experiment.



(a) Second iteration (b) Third iteration

Figure 8.2: Stream hierarchy instances maintained by the NIProxy during the second and third iteration of the first experiment.

Figure 8.1. The exact layouts of the client's stream hierarchy during the second and third execution of the experiment are depicted in Figures 8.2(a) and 8.2(b), respectively. In both stream hierarchy instances, all leaf nodes were of the `continuous` type.

A thorough analysis of the generated results is in order. First of all, remark from Figure 8.1(a) that, even in case the NIProxy was not included in the experiment, the client's available downstream bandwidth volume was more or less respected. This can be attributed to TCP's built-in congestion control mechanism: as was mentioned in section 8.1, TCP dynamically tunes its transmission rate to the network's current bandwidth capacity. Secondly, the network traces depicted in Figures 8.1(b) and 8.1(c) prove that the rate control functionality of the `continuous` leaf node type works very effectively. In particular, the bandwidth consumption curves of the network flows that were represented by the leaf nodes are perfectly smooth. Notice the heavy contrast with the bitrate irregularities that are exhibited in Figure 8.1(a). Third, Figures 8.1(b) and 8.1(c) illustrate the overflow protection buffer of the NIProxy's NTS scheme (see section 4.6). In particular, the network traces reveal that a small amount of the client's available bandwidth was left untouched to prevent swift surges in network flow bandwidth consumption over which the NIProxy has no control from inducing bandwidth capacity violations. However, this feature appears to be solely useful in the presence of real-time network traffic; non-real-time network traffic is rate controlled by the NIProxy and is hence propagated to its destination at a perfectly steady rate (i.e., after NIProxy processing, unanticipated increases in stream bandwidth consumption will never be experienced). The overflow prevention margin immediately also explains why it took slightly longer to receive the requested files in the two executions of the experiment in which the NIProxy was involved. Recall from section 4.6 however that per client parametrization of the size of the safety buffer is supported. As a result, it is possible to completely disable the NIProxy's overflow protection mechanism (i.e., set the buffer's size to zero) in environments in which only non-real-time network traffic is present, this way effectively eliminating the disadvantage of increased reception times for non-real-time content[2].

The bandwidth allocation evolution in Figure 8.1(c) is worthy of special attention. At first sight, this evolution might seem somewhat surprising. In particular, it can be seen that, after the instantiation of P2P stream 2, the amount of bandwidth that was granted to P2P stream 1 gradually increased over time, even though the former had a higher weight value associated with it in the stream hierarchy throughout the entire experiment. This effect is accounted for by the fact that, as was described in section 4.2.4, the maximal bandwidth usage of its children plays an important role in the bandwidth distribution policy of the `WeightStream` node type. Since file `slide.ppt` initially (i.e., immediately after it was requested) was allocated significantly more bandwidth compared to the files that were requested on P2P stream 1, the maximal bandwidth consumption of P2P stream 2 initially also diminished at a much higher pace. As the experiment progressed, the growing difference between the maximal bandwidth consumption of both P2P streams increasingly outweighed their associated weight values, which explains why a gradually larger portion of the available bandwidth was allocated to P2P stream 1. This kind of behavior was not anticipated and is likely to confuse the user. It might even cause users to unjustly accuse the NIProxy of malfunctioning. While previous empirical evidence confirms the adequacy of the `WeightStream` node in a real-time data exchange environment, this experiment has hence diagnosed it as being not particularly fit to shape non-real-time network traffic. This insight in turn led to the development of the related `WeightData` internal node type, which disregards the maximal bandwidth consumption of child nodes during its bandwidth brokering calculations but otherwise operates identically to the `WeightStream` node (see section 4.2.5). As will be confirmed in chapter 9, using the `WeightData` node type to shape non-real-time network traffic does yield intuitive results.

[2]Based on this discussion, one would have expected the overflow prevention margin of the NIProxy's network traffic shaping scheme to be disabled in this experiment. The justification for the use of a non-zero value is given by the fact that the safety buffer will recur in the discussion of the second test, where real-time and non-real-time network traffic competed for the available downstream bandwidth and where the danger of bandwidth capacity overflow was hence a tangible reality. Due to the higher complexity of the latter experiment however, the repercussions of the safety buffer on the network traffic shaping outcome were less pronounced, which in turn complicates their explanation. It was therefore decided to already discuss the overflow prevention margin in the simpler context of the first experiment.

Interpretation

Based on the discussion thus far, the inclusion of the NIProxy in the experiment appears to have had negative rather than positive implications on the user QoE (i.e., an increase was noticed in the time that was required to receive the requested files). However, an important feature of the NIProxy which has not yet been mentioned until now is its ability to introduce differentiation in the treatment of the network flows that are relevant to the client. As an example, suppose that the non-real-time network connections which were requested in this experiment corresponded to respectively a low- and high-priority P2P communication channel. Consequently, the client would expect files requested on P2P stream 2 to be delivered with a higher priority (i.e., faster) compared to files solicited on P2P stream 1. This kind of behavior can readily be enforced by the NIProxy's bandwidth brokering framework. It suffices to create a stream hierarchy in which the two P2P streams are adequately discriminated from each other using some type of internal node. Moreover, since the NIProxy supports multiple types of bandwidth distribution techniques (i.e., internal stream hierarchy node classes), the differentiation can even be tuned to the specific requirements of the user or the application.

The bandwidth distribution strategies that were implemented by the NIProxy during the second and third iteration of the experiment were based on the example requirement from the previous paragraph. Respectively a `Priority` and a `WeightStream` node were relied on to differentiate between the involved non-real-time network streams. By now, the added value of incorporating the NIProxy in the experiment emerges: comparing Figures 8.1(b) and 8.1(c) with Figure 8.1(a) indicates that file `slide.ppt`, which was requested on the high-priority P2P connection, was received sooner in case the bandwidth management functionality of the NIProxy was exploited. The achieved gain respectively equaled 11 and 2 seconds. Consequently, leveraging the NIProxy's NTS support led to the application conforming more closely to the user's expectations, which is likely to be a key prerequisite if the goal is to deliver a high QoE.

Before concluding the discussion of the first experiment, it is worth noting that the produced results could presumably also be achieved by modifying the client software of the employed test application. In particular, in its current implementation, the only mechanism that is available to express the relevance of content items is to request them in an appropriate order. By refining this implementation, it could become possible to attain results that are comparable to the ones delivered by the NIProxy. However, leveraging the NIProxy's functionality allows for the implementation of the client software to be kept simple and hence for the application development time to be significantly reduced. Moreover, embedding NTS features directly in the client software is not a reusable solution since it isolates the functionality in one particular application. In contrast, the NIProxy's bandwidth management facilities can be exploited by a broad range of distributed applications, even concurrently, which renders it a much more favorable solution from an economic point of view (see also section 2.6).

8.3.3 Experiment 2: Simultaneously Managing Real-Time and Non-Real-Time Network Traffic

Experiment Description

While the first experiment focused solely on non-real-time network traffic, the objective of the second experiment was to validate whether the NIProxy manages to orchestrate a client's downstream bandwidth consumption in case the traffic mix includes both real-time and non-real-time flows. To simulate a scenario in which both types of network traffic were involved, the test application was employed to set up

- a real-time video connection between the monitored client and two remote hosts H1 and H2, and
- a non-real-time communication channel between the monitored client and remote host H1 as well as another remote host that represented a dedicated File Server (FS).

On the non-real-time connections, the following files were requested:

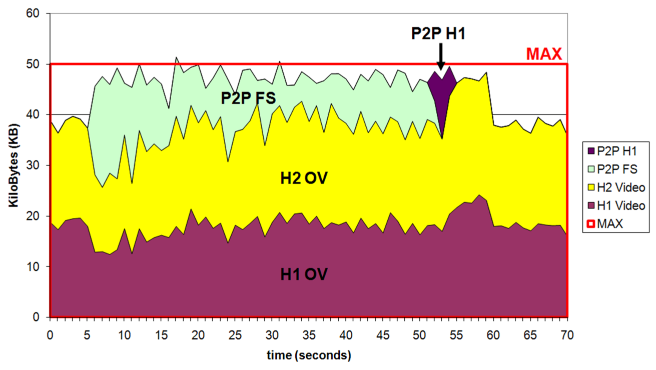
- `geom.3ds`: A 3D model of size 484652 bytes, requested from FS
- `img.png`: An image of size 23419 bytes, requested from remote host H1

The file requests were issued by the client in immediate succession, in order of enumeration. Furthermore, to allow for a comprehensive demonstration of the potential of the NIProxy's network traffic shaping mechanism, some additional constraints and requirements were defined. More precisely, it was specified that the video stream that was emitted by remote host H1 had a higher significance for the managed user than H2's video stream, and that priority should be given to files requested on the non-real-time network connection with the file server (compared to files requested from remote host H1). The reasoning behind this latter constraint might be, for instance, that the dedicated file server maintained files that were vital for the execution of the application and which hence needed to be delivered as quickly as possible, while the other non-real-time connection simply enabled the managed client and host H1 to exchange files in a P2P fashion (and therefore was far less important). Finally, the requirement was imposed that non-real-time communication should receive a "fair" amount of the managed client's total downstream bandwidth volume (which was this time set to 50 KBps). In particular, it was determined that at least 30 percent of the client's downstream bandwidth had to be designated to the reception of non-real-time data.

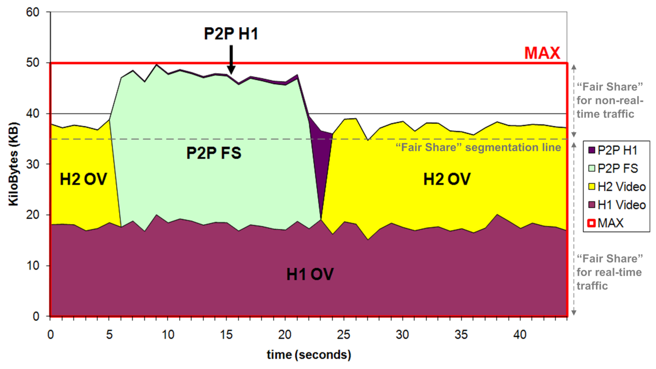
Analogous to the approach in section 8.3.2, the experiment was repeated thrice. In the initial execution, the NIProxy was excluded from the experimental setup. In contrast, during the second and third experiment iteration, the NIProxy's network traffic shaping functionality was exploited to govern the data delivery towards the monitored client. These latter two repetitions differed from each other in terms of NIProxy configuration. In particular, in these experiment executions, the NIProxy's static video transcoding service from section 5.3 was respectively disabled and enabled. To recapitulate, this service enables the NIProxy to reduce the bitrate of video flows by on-the-fly reducing their quality parameters.

Experimental Findings

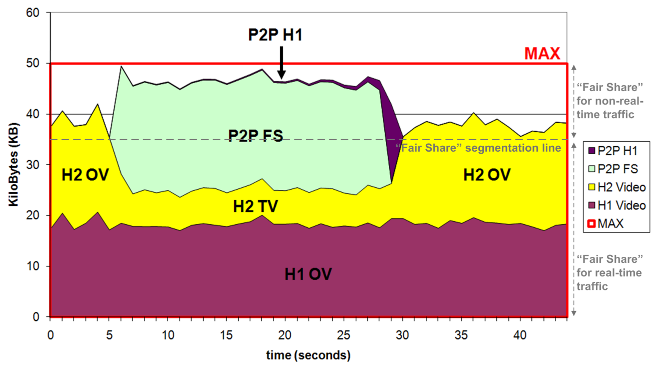
Network traces that illustrate the network traffic that was received by the client during the different executions of the experiment are shown in Figure 8.3, while Figure 8.4 depicts the stream hierarchy instances that coordinated the NIProxy's NTS behavior during the second and third iteration. As can be seen, both stream hierarchy instances were structured identically and merely varied in their representation of the remote video sources. This divergence is of course attributed to the inclusion of the static video transcoding service in the final repetition of the experiment, which caused each video source to be represented by two separate `discrete` leaf nodes instead of just one. The leftmost node of each pair corresponded with the Original Version (OV) of the video stream (i.e., the video stream as it was transmitted by the source), while the rightmost node represented the Transcoded, lower-quality Version (TV) of this stream (i.e., the variant that was on-demand generated by the video transcoding service). A more important observation regarding Figure 8.4 is that the general layout of the stream hierarchy instances was inspired by the constraints which were identified at the beginning of this section. In other words, it was attempted to construct the stream hierarchy instances in such a manner that the specified constraints and requirements would be satisfied



(a) Without NIProxy

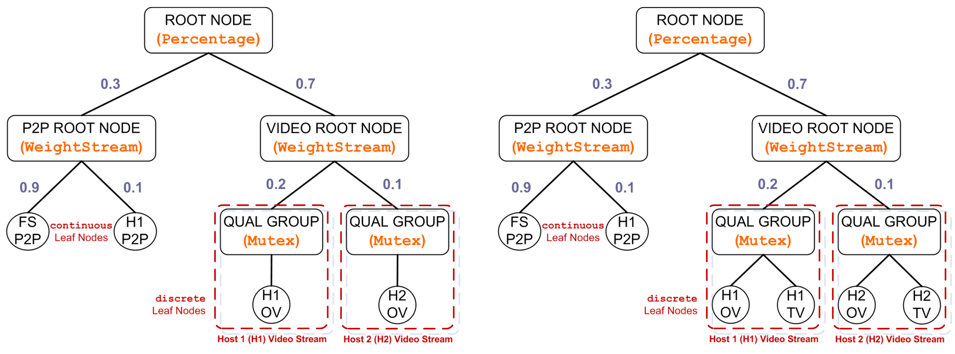


(b) With NIProxy (video transcoding disabled)



(c) With NIProxy (video transcoding enabled)

Figure 8.3: Stacked graph plots of the media content received by the client during the three different executions of the second experiment.



(a) Second iteration (b) Third iteration

Figure 8.4: Stream hierarchy instances maintained by the NIProxy during the second and third iteration of the second experiment.

during the experiment.

Analysis of Figure 8.3(a) reveals that, in the initial execution of the experiment, the simultaneous reception of real-time and non-real-time data resulted in a number of important issues. First of all, the contention for the client's available downstream bandwidth yielded a wrongful penalization of the non-real-time network traffic. In particular, whereas the video sources continued to transmit at a constant rate, irrespective of the existence of concurrent traffic, the non-real-time TCP connections automatically downscaled their transmission rate to avoid over-encumbrance of the client's downstream access connection. Consequently, the non-real-time network traffic needed to content itself with the amount of downstream bandwidth that was not arrogated by the real-time flows, which is in disaccord with the stated requirement that the non-real-time communication should receive a fair share of the client's downstream bandwidth capacity. Secondly, although the real-time video flows claimed the majority of the available bandwidth, they still suffered from the contention from the non-real-time traffic. In particular, the contention introduced small amounts of packet loss and caused the delivery of the real-time video traffic to become more irregular, which in turn resulted in a deteriorated client-side video playback.

Looking at Figure 8.3(b), it is clear that similar issues did not arise when the bandwidth management functionality of the NIProxy was exploited. More specifically, by relying on a `Percentage` node to differentiate between real-time and non-real-time network traffic, the latter was guaranteed a certain fraction of the client's downstream bandwidth volume throughout the entire experiment (i.e., at least 30 percent). To confirm this statement, the percentual bandwidth segmentation is visualized in the network plot in Figure 8.3(b) (and also in Figure 8.3(c)) as a horizontal line[3]. The outcome was a much faster delivery of the requested files, however at the expense of the least important video stream being discarded as long as there was non-real-time data available to forward to the client. Secondly, thanks to the NIProxy successfully rate controlling non-real-time network traffic when transmitting it to its destination, the reception of real-time video traffic at client-side remained unaffected and, as a result, no deterioration in video playback was noticed. Finally, as can be deducted from Figure 8.3(c), enabling the NIProxy's static video transcoding service entailed the additional advantage of allowing the real-time video traffic to consume its entitled percentage of the client's downstream capacity to a greater extent. In particular, thanks to the availability of the video transcoding service, the NIProxy's NTS algorithm was able to forward the transcoded version of H2's video stream at the moment the non-real-time network traffic was initiated. This is explained by the lower bitrate of the transcoded video version (compared to its original counterpart); consequently, its transmission did not cause the non-real-time traffic to be denied its due share of the downstream bandwidth volume.

Interpretation

The advantage of incorporating the NIProxy in the second experiment was twofold. First of all, it resulted in the client's downstream channel capacity being apportioned correctly among the involved real-time and non-real-time network flows. Secondly, it allowed for the requirements which were specified at the beginning of this section to be met with minimal effort (i.e., without requiring substantial modifications to the client software). Based on these observations, it is argued that the introduction of the NIProxy in this experiment is expected to have had a favorable influence on user QoE.

An important final remark is that the generated experimental results once again exemplify the tremendous potential and added value of the cooperative interface that is defined between the NIProxy's network traffic shaping and multimedia service provision facilities. In this experiment specifically, the

[3]Notice that both involved traffic types were allowed to exceed their fair share in case their "competitor" failed to fully consume its entitled bandwidth percentage. This is due to the two-phase operation of the `Percentage` node: recall from section 4.2.3 that the residue of the bandwidth that was initially assigned to a particular child will be distributed over its siblings in a subsequent stage. In this particular situation, the surplus bandwidth of the P2P root node was transferred integrally to the video root node (as it was its sole sibling), and vice versa.

bundled effort of both traffic engineering techniques enabled the real-time video traffic to more completely consume its allocated bandwidth share and as such prevented the temporary interruption of the delivery of the less significant video stream during periods of active file transfer.

8.4 Conclusions

This chapter has treated the NIProxy's support for the mediation of non-real-time network traffic. This type of traffic, which will for instance arise when exchanging file data between distributed hosts, exhibits properties that are widely divergent from those of real-time network streams. In particular, contrary to real-time traffic, a typical non-real-time connection has relatively relaxed latency restrictions and does not impose hard throughput requirements; conversely, it is likely to suffer considerably from unreliable or error-prone delivery. Given these discrepancies, a quite different methodology for shaping real-time and non-real-time network traffic is advocated. This is acknowledged by the NIProxy's NTS implementation, which provides two distinct classes of leaf nodes to represent network flows in the stream hierarchy. The aptitude of `discrete` leaf nodes for managing real-time network traffic was already established in chapter 7. This chapter on the other hand has confirmed that non-real-time transmissions can be effectively shaped by associating them with a leaf node of the `continuous` type in the stream hierarchy. In contrast to its `discrete` variant, a `continuous` leaf node provides a theoretical framework to modify the bandwidth consumption of network flows in a continuous manner (see section 4.3.2 for more information). To be able to actually enforce the calculated bit budgets (and hence to make `continuous` leaf nodes usable for managing non-real-time network traffic in practice), the behavior of the leaky bucket model was imitated, which involved the adoption of progressive buffering as well as rate control techniques. In particular, non-real-time content is buffered locally by the NIProxy and subsequently propagated to its destination at a rate that conforms to the exact amount of bandwidth which has been reserved for its encompassing network flow in the current NTS time interval.

The validity of the NIProxy's non-real-time network traffic shaping approach has been ascertained experimentally. More specifically, it was shown that the NIProxy succeeded in practically managing client downstream bandwidth in the presence of competing real-time and non-real-time network flows. In addition, the impact of the NIProxy's NTS operations on user QoE has been investigated by relating the produced bandwidth brokering results to the default situation (i.e., the setup in which the NIProxy is not included). This comparative study has revealed that the traffic engineering tasks performed by the NIProxy led to a considerable improvement in the QoE that was perceived by the end-user. This chapter has hence presented an important contribution, since being able to adequately cater to non-real-time network traffic considerably extends the QoE optimization effectiveness and the overall applicability of the NIProxy.

As a concluding remark, the discussion in this chapter might have sparked the impression that the `continuous` leaf node category is exclusively suitable for the management of non-real-time network traffic. This assumption will however be refuted in chapter 13, where it will be proven that `continuous` leaf nodes can just as much be exploited to regulate the dissemination of real-time data, on the condition that they are accompanied by appropriate (i.e., specialized) functionality to practically enforce the calculated bit budgets for the considered type of real-time network traffic.

Chapter 9

Efficient Transmission of Rendering-Related Data

The experimental results regarding network traffic shaping that have been presented thus far in this part of the dissertation were generated using "demonstrator" applications whose bandwidth management requirements could be satisfied via relatively straightforward strategies. In this chapter on the other hand, the NIProxy's suitableness to broker client downstream bandwidth in the context of a concrete, real-world distributed application will be investigated [Wijnants 08a]. The considered application supports real-time audiovisual communication between participants and in addition employs an advanced rendering scheme that implicates a number of specific requirements concerning the distribution of rendering-related data. The application's default communication model however fails to consistently guarantee that these requirements are met, with potentially adverse effects on the usage experience as a result. In contrast, through the presentation of representative experimental results, it will be corroborated that the NIProxy's network traffic shaping facilities are capable of effectively addressing the imposed network-related stipulations. As such, it will be shown that the introduction of the NIProxy to shape the network traffic that is induced by the considered distributed application likely led to an improvement of the QoE of its users.

9.1 Introduction

Support for custom or user-generated content in distributed multi-user applications is rapidly gaining popularity. One exemplification of this trend is the currently ongoing progression of the World Wide Web from a mostly static information source to a more dynamic environment where users are allowed to generate and distribute content themselves, for instance through blogs and wikis (the so-called Web 2.0). Another example is the emergence of social virtual worlds such as Second Life [Second Life 10], which enable their users to create virtual items and even to trade these self-made objects with others.

As a logical consequence of this evolution, there is an increasing need to transmit possibly large amounts of custom data to clients of distributed applications. Luckily, the emergence of broadband Internet subscriptions for residential users (such as xDSL or broadband cable) has made this feasible. Nonetheless, client downstream bandwidth is still limited and does not necessarily suffice to effectively receive all data that is produced by the distributed application(s) that the user is currently running. Like any scarce resource, client downstream bandwidth should consequently be managed intelligently and deliberately, with as ultimate goal providing the user with a maximal usage experience, given his current bandwidth constraints.

This chapter will again treat the NIProxy's network traffic shaping functionality which enables automatic and dynamic management of client downstream bandwidth. A significant difference with previous chapters however is the distributed application that will be used for evaluation purposes. In particular, the bandwidth brokering results that have been presented thus far were generated in "artificial" test environments and case studies which involved applications that had been designed and implemented specifically for investigating the NIProxy's network traffic shaping potential and performance. In contrast, this chapter will report on the findings of employing the NIProxy to manage client downstream bandwidth in an existing real-world distributed application supporting user-generated content, which in this particular case necessitates the dynamic distribution of rendering-related data to clients. Compared to the previous test setups, the distributed application requires a much more sophisticated bandwidth management strategy. The main contribution of this chapter is hence an illustration of the NIProxy's ability to effectively manage the bandwidth of clients of realistic distributed applications with potentially complex requirements. As a secondary contribution, the produced experimental results will as such also demonstrate the NIProxy's flexibility and broad applicability.

9.2 Considered Distributed Application

To assess the NIProxy's appropriateness to manage client downstream bandwidth in a non-artificial setting, it was integrated in a concrete (in-house developed) 3D Networked Virtual Environment (NVE) application. The considered NVE assigns great importance to inter-participant communication and hence supports both audio and video chat between users. Another important feature of the NVE is its rendering scheme. In particular, as will be described in section 9.2.1, the application incorporates hybrid 3D rendering techniques as well as sophisticated Level of Detail (LoD) methods. Finally, the NVE application enables users to populate the shared virtual world with custom, possibly self-made 3D content. Distribution of this user-provided data occurs through a dedicated file server, which disseminates the data to clients in a unicast manner and on an as-needed basis. As an example, when a user enters a certain part of the virtual environment which he has not visited before, the data that is required for the rendering of this region will need to be downloaded from the file server.

9.2.1 Rendering Scheme

The NVE's rendering scheme involves both *geometric* models and *Image-Based Representation* (IBR) data [Jehaes 04b][Jehaes 08]. In short, *Relief Texture Mapped Objects* (RTMOs) are used, which were first presented by Oliveira et al. in [Oliveira 00], as an efficient representation method for distant objects. These representations are made up of a collection of images with depth information, which are warped during run-time in order to get an appropriate view based on the current camera position. On the other hand, objects close to the viewer are rendered using Hoppe's *Progressive Meshes* (PM) approach [Hoppe 96], hereby assuring that geometric rendering does not cause the framerate to drop below a predefined threshold. The main advantage of the mixed rendering strategy is that it greatly reduces the time that is needed for visualizing complex virtual scenes without incurring too large a sacrifice in image quality. In other words, the scheme aims to maintain an interactive framerate while at the same time preventing the introduction of significant perceptual distortions during virtual world rendering.

Figure 9.1 depicts a particular virtual object that is rendered both geometrically and RTMO-wise. As can be seen, the geometric rendering contains much more detail compared to the IBR version. However, as the RTMO representation is mainly exploited for the rendering of distant objects, the considered NVE's hybrid model representation solution introduces only a lim-

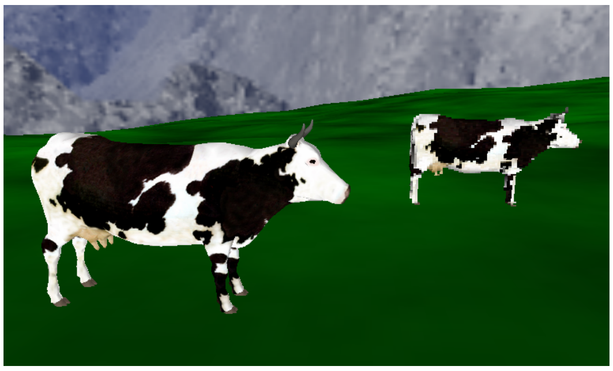


Figure 9.1: Comparative illustration of model quality yielded by respectively geometric rendering and RTMO representation [Jehaes 08].

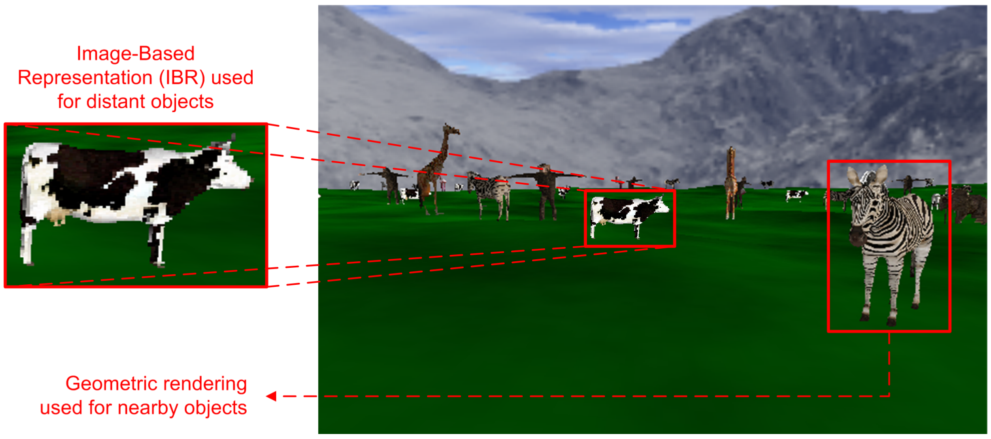


Figure 9.2: A relatively crowded virtual environment, visualized using the NVE's hybrid geometric/IBR rendering scheme [Jehaes 08].

ited amount of degradation in terms of image quality. This is confirmed in Figure 9.2, which shows the scheme in action for a relatively densely populated virtual scene.

In the context of the just described hybrid representation and rendering scheme, Jehaes et al. have also discussed a number of optimization techniques for the network dissemination of representational data in NVE applications [Jehaes 04a]. In particular, they state that by first streaming the low-detail

Table 9.1: Storage sizes (in KiloBytes) of two Progressive Mesh (PM) models in the NVE database.

Texture	Base Mesh	PM L1	PM L2	PM L3	Base Compression (base mesh & texture)
304	12	11	21	/	283
161	22	21	41	83	168

image-based representations of objects in the virtual environment, it becomes possible to quickly present the user a complete, albeit low-quality view of the world. The next step of their proposed solution involves the progressive downloading of more detailed representations (i.e., geometric information) on the basis of the current rendering need so that the visual quality of the view is gradually upgraded over time. Through analysis of the network traffic that is introduced by the distribution of the involved rendering-related data, it was determined that the reduced-detail representations strike an effective balance between perceptual fidelity and transmission time. As a result, the proposed transmission scheme for representational information significantly reduces the time that is needed for rendering, at an acceptable quality, an initial version of the virtual environment, with the visual accuracy subsequently being gradually improved over time.

9.2.2 Model Data

The NVE's model database stores representations in the form of both 3D progressive meshes and relief texture mapped objects. Focusing first on the progressive mesh data, Table 9.1 presents PM storage size information for two example models in the database. The first column indicates the amount of texture data for each model, stored in the JPEG format. Comparing this value to the data size of the base mesh in the second column reveals that the texture data significantly outweighs the latter in terms of storage (and hence transmission) requirements. This gap could be decreased by resorting to a multi-resolution texturing solution but, overall, texture data will generally take up a considerable fraction of the total model data. The columns labeled L1 to L3 specify the data size of each consecutive PM level. At each PM step, the triangle count is doubled, which in turn results in a more detailed and sharp model visualization. This is illustrated in Figure 9.3, where the base and full-quality mesh of an example 3D model are placed side by side. For comparison, the RTMO representation of the model is also depicted; it is

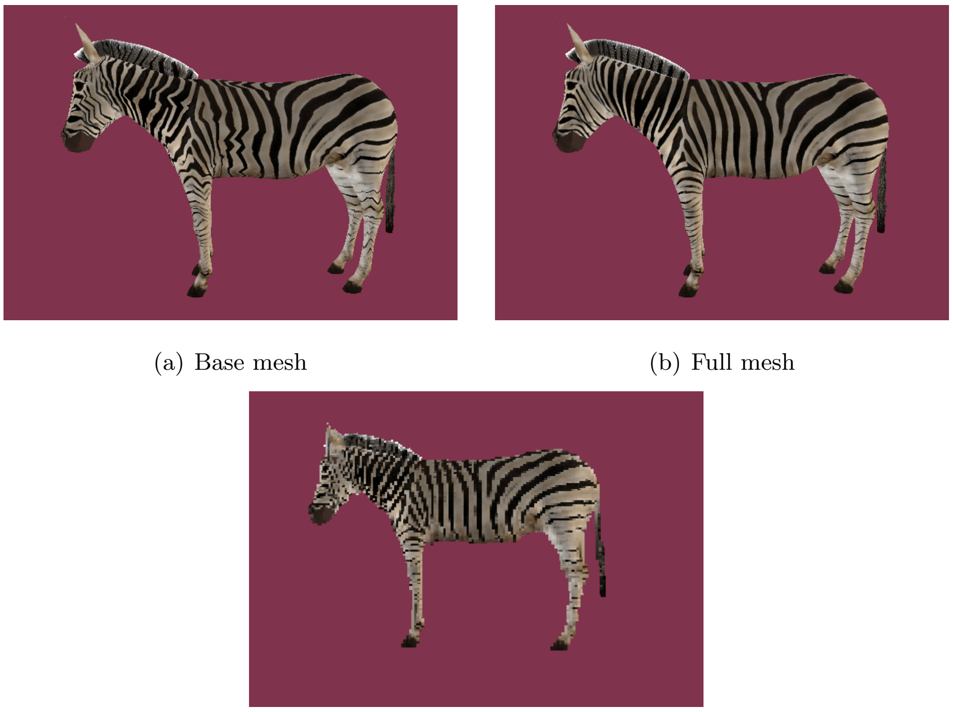

(a) Base mesh (b) Full mesh (c) RTMO

Figure 9.3: Model quality comparision [Jehaes 08].

clear that the geometric rendering of the base mesh still considerably surpasses the IBR approach in terms of perceptual accuracy. In order to transmit the PM models in an efficient way, they are stored in a compressed format at server-side. This implies that the models need to be unpacked at client-side before they can be used in the rendering stage. The final column in Table 9.1 indicates the size after compression of the texture data together with the base mesh. Since the texture data is already stored in the JPEG format, the additional compression does not significantly reduce the total file size.

A completely different situation is evoked by Table 9.2, which displays the storage sizes for the RTMO representations. The storage size for a RTMO mainly depends on the resolution of the relief textures (RTs) instead of on the model that is being represented. Each depth pixel is stored as a RGBA value, with the alpha component representing the quantized depth value. For rendering a RTMO, at most three relief textures are needed for a specific viewpoint. Comparing the compressed values for three relief textures to the data size of

Table 9.2: Storage sizes (in KiloBytes) for RTMO image-based representations.

Resolution	# RTs	Size	Compressed Size
32×32	1	4	2
	3	12	6
	6	24	13
64×64	1	16	7
	3	48	21
	6	96	45

compressed PM base representations (including textures) establishes that the RTMO transmission size is much smaller. In the NVE database, only relief texture resolutions of 32×32 and 64×64 are used, since these provide sufficient image quality for distant objects and for the initial rendering of nearby objects. Table 9.2 even points out that complete RTMOs (i.e., a bundle of 6 RTs of a certain model) still seriously outperform the PM base representations with regard to transmission time.

9.2.3 Network Communication Issues

The considered NVE application expects network bandwidth to be abundantly available (which is unfortunately still an unrealistic assumption given the throughput constraints of contemporary residential Internet connections) and therefore lacks an intelligent bandwidth management solution. In particular, the models of the objects that populate the virtual world are requested from the file server one at a time on the basis of their current scene priority. However, as priorities might alter rapidly during scene traversal, and especially during viewpoint rotation, a better solution was advocated that would enable more effective and interactive control over the downloading of model data that is required for rendering the world at client-side. Moreover, the NVE application's communication scheme does not adequately address the issues that are introduced when combining the transmission of non-real-time rendering-related data with real-time audiovisual network traffic.

Based on these shortcomings, the NVE application was considered an excellent candidate for testing the NIProxy's network traffic shaping functionality. Section 9.3 will therefore discuss how the NIProxy was employed to intelligently regulate the downloading of 3D model data to the client and to harmonize these transmissions with the real-time network traffic that is introduced by the NVE's audiovisual communication facilities.

9.3 Implementation

The discussion in the previous section has revealed that the NVE application requires the distribution of real-time as well as non-real-time data to clients. Especially the application's advanced rendering scheme and its support for user-generated content causes client bandwidth management to be a far from trivial objective and imposes a number of specific requirements and constraints. This section will report on the way the NIProxy was exploited to intelligently fulfill this task and will in addition present the main implementational issues which the NIProxy incorporation encompassed.

9.3.1 Stream Hierarchy Design

To be able to effectively orchestrate the bandwidth consumption of the NVE application, the requirements that are imposed by its distinct features needed to be translated into an appropriate layout for the client stream hierarchy. The structure that was decided upon is illustrated in Figure 9.4. As can be seen, an internal node of the `Percentage` type was selected as root of the hierarchy and was used to discriminate between respectively the real-time and non-real-time network traffic that is induced by the NVE application. In more detail, it was determined that the real-time network traffic received 30 percent of the client's available downstream bandwidth, while 70 percent was reserved for the reception of non-real-time network streams.

A `WeightStream` node served as root of the real-time subtree in the stream hierarchy. All real-time (i.e., audio and video) network flows were represented as `discrete` leaves and were made direct children of this root. The `discrete` leaf nodes supported two discrete bandwidth levels and were hence confined to turning their associated stream off and on (see section 4.3.1). To determine the weight values of the real-time network flows, the scheme that was introduced in section 7.1 was reused. To recapitulate, this scheme dynamically assigns weight values depending on virtual distance as well as virtual orientation information.

The root of the non-real-time branch of the stream hierarchy on the other hand consisted of a `Priority` node and had two children, which were both of type `WeightData`. The leftmost `WeightData` node was intended to group together all High-Priority (HP) rendering-related files that need to be delivered to the client, whereas the rightmost functioned as root for rendering-related

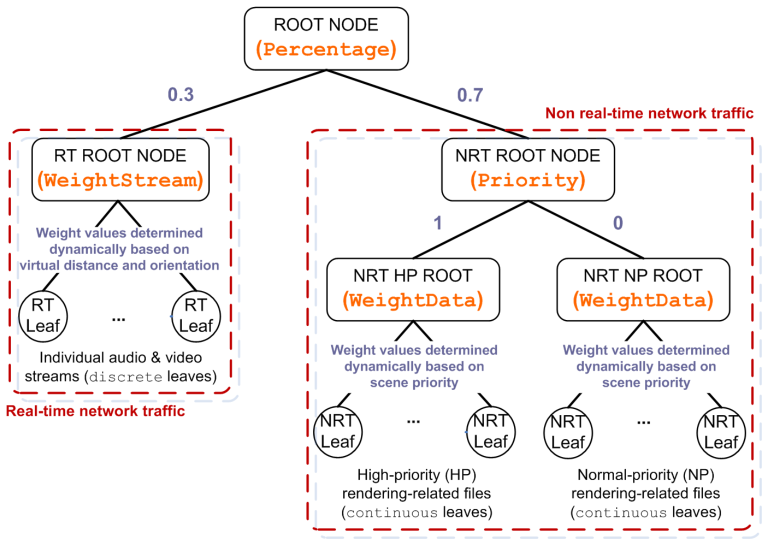

ROOT NODE
(Percentage)
0.3
0.7
Non real-time network traffic
RT ROOT NODE
(WeightStream)
Weight values determined dynamically based on virtual distance and orientation
RT Leaf
...
RT Leaf
Individual audio & video streams (discrete leaves)
Real-time network traffic
NRT ROOT NODE
(Priority)
1
0
NRT HP ROOT
(WeightData)
NRT NP ROOT
(WeightData)
Weight values determined dynamically based on scene priority
Weight values determined dynamically based on scene priority
NRT Leaf
...
NRT Leaf
NRT Leaf
...
NRT Leaf
High-priority (HP) rendering-related files (continuous leaves)
Normal-priority (NP) rendering-related files (continuous leaves)

Figure 9.4: General layout of the stream hierarchy used to manage the downstream bandwidth of clients of the considered NVE application.

data having Normal-Priority (NP). In particular, the HP part of the non-real-time subtree was used exclusively to receive IBR data and, more specifically, only the IBR data of those models that lay in the current viewing frustum and were therefore crucial for rendering a first view of the virtual environment. All other rendering-related files were clustered under the NP part of the non-real-time subtree. Each individual HP as well as NP rendering-related object was incorporated in the stream hierarchy as a `continuous` leaf node whose maximal bandwidth consumption equaled the amount of bytes that still needed to be delivered to the client. The allocation of weight values to these `continuous` leaf nodes was controlled by the scene priority of the rendering-related object with which they were associated. In turn, scene priority was determined by the NVE's LoD selection mechanism, which takes into consideration distance to the viewer as well as model display size. Furthermore, the NVE application performs pre-loading of IBR model data outside the frustum, but inside a circular Area of Interest (AoI) around the viewpoint. These data streams were given a low weight value to guarantee that their delivery did not hinder the reception of streams of higher priority. A schematic clarification of how rendering-related data was exactly categorized and hence incorporated in the stream hierarchy is provided in Figure 9.5.

The grouping HP root node had a higher priority value associated with it, compared to the NP root node. Consequently, the entire percentage of the

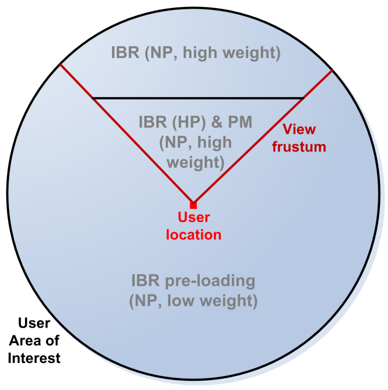


Figure 9.5: Categorization of rendering-related data.

client's available downstream bandwidth that is designated to the reception of non-real-time network traffic will first be exploited to receive the HP rendering-related files. Only if any bandwidth remains in excess, it will subsequently be granted to the NP branch of the non-real-time subtree.

Notice that previous findings regarding the bandwidth brokering skills and characteristics of the `WeightStream` node class were kept in mind when designing the layout of the stream hierarchy. In particular, section 7.3 has established the proficiency of this type of internal node in apportioning bandwidth among real-time network traffic. Section 8.3.2 on the other hand has demonstrated that leveraging a `WeightStream` node to shape non-real-time network traffic might yield counterintuitive results. Therefore, in the devised stream hierarchy, the real-time data was linked by means of a `WeightStream` instance, whereas `WeightData` nodes were relied on to cluster the non-real-time transmissions. Section 9.4 will confirm that employing the `WeightData` node type to coordinate the distribution of bandwidth over non-real-time network traffic does yield intuitive results.

9.3.2 Support for Stream Hierarchy Leaf Node Relocations

Since users can freely roam about the virtual world that is offered by the NVE application, the importance of individual real-time and non-real-time network flows is likely to alter dynamically over time. For the real-time network traffic, shifts in stream importance are captured by appropriately updating the weight value of the corresponding `discrete` leaf node. For non-real-time network traffic however, the situation is somewhat more complicated and possibly requires relocation of `continuous` leaf nodes in the client's stream hierarchy.

In particular, as a user moves to a new location or changes his viewing direction in the virtual world, non-real-time data that previously belonged to the high-priority category might now be classified as being normal-priority and vice versa. When this occurs, in addition to enforcing any necessary updates to their individual weight values, the involved `continuous` leaf nodes are automatically transferred between the HP and NP branches of the non-real-time subtree. Finally, besides parent switching, complete removal of leaf nodes from the client's stream hierarchy is also supported. A plausible cause are radical user relocations in the virtual environment, which could reduce the importance of certain network flows to such an extent that they should no longer be allocated any client downstream bandwidth. By (temporarily) pruning the corresponding leaf nodes from the stream hierarchy, this can easily be guaranteed.

9.4 Evaluation

The proposed bandwidth distribution strategy for the NVE application has been assessed via two distinct experiments in which the NIProxy was responsible for managing last mile data delivery to clients. The initial experiment involved only a limited number of models and did not include real-time network traffic. In contrast, the second test simulated a more realistic scenario which encompassed a densely populated scene, a less artificial client downstream bandwidth limit and contention from real-time traffic.

9.4.1 Minimalist Experiment

The initial experiment modeled a minimalist setup. By excluding real-time network traffic and by considering only a moderate number of virtual models, the client-side reception of non-real-time rendering-related data could be traced comprehensively and unambiguously. The results are shown as a stacked graph in Figure 9.6. Besides constraining the model count, it was also decided to limit the downstream throughput of the client's access connection to an unrealistically low 20 KiloBytes per second (KBps) to allow for an intelligible analysis of the produced results.

Looking at Figure 9.7(a), it can be seen that at the beginning of the experiment, only objects 0 and 1 were positioned in the user's field of view. Object 0 was located relatively far from the user in the virtual world. In contrast, the virtual distance between the user and object 1 was relatively small. The NVE's rendering scheme consequently determined that solely IBR data was required for object 0, whereas IBR data as well as all PM levels needed to be delivered

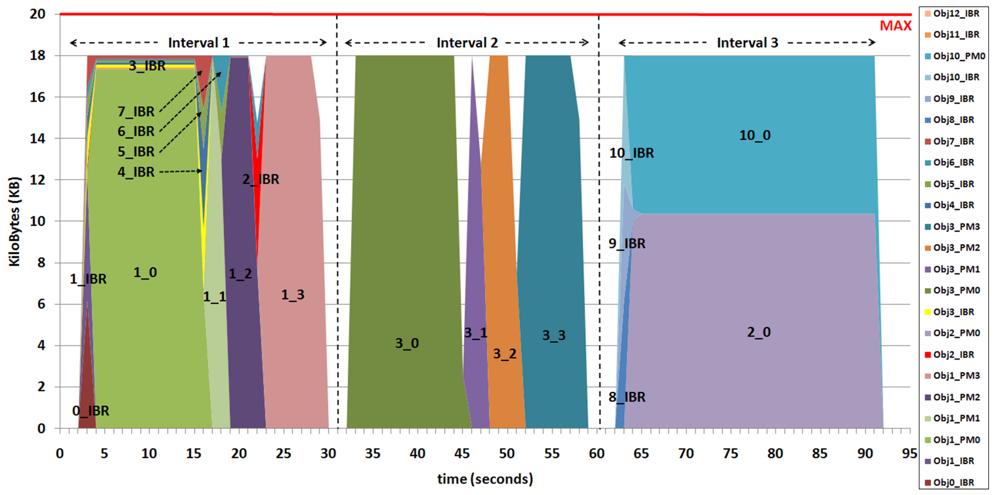
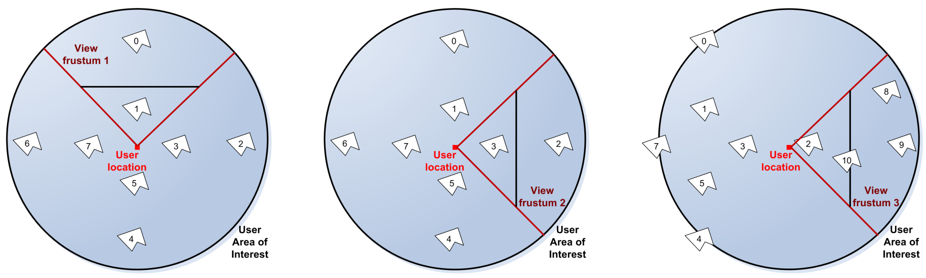


Figure 9.6: Simple scenario network traffic chart (stacked graph) demonstrating deliberate management of non-real-time (i.e., rendering-related) traffic (in KBps).

to the client for object 1. The network chart in Figure 9.6 discloses that for both objects, the high-priority IBR data was forwarded first, after which the delivery of object 1's PM levels was started. The network trace also illustrates that, as soon as the RTMO representations of the objects in the view frustum were received successfully, the pre-loading of IBR data of non-visible objects began taking up a very small fraction of the available bandwidth.

After approximately 30 seconds, the user performed a 90 degree rotation in the virtual world (see Figure 9.7(b)). As this rotation put both objects 0 and 1 outside the user's visible area, the weight values that were associated with both model streams were significantly reduced. On the other hand, the view frustum now contained objects 2 and 3, so their data needed to be propagated to the client at a higher weight and priority than before. Again, object 3 was close to the viewer and was given more weight, while only IBR data was required for object 2. Note from the network chart that, after the rotation, only the geometric data for object 3 was transmitted since the IBR files for both objects had already been received in the first interval of the experiment, during the NVE's IBR pre-loading phase for objects outside the view frustum. At the beginning of the second experiment interval, the renderer could therefore immediately present an initial view of the currently visible objects.

Finally, at around 60 seconds, a forward translation was performed to arrive at the scene layout which is depicted in Figure 9.7(c). The user's forward



(a) Interval 1 (b) Interval 2 (c) Interval 3

Figure 9.7: Scene layout for the minimalist experiment.

movement caused object 3 to fall outside the view frustum. At the same time, objects 2 and 10 were now marked for PM rendering, while objects 8 and 9 were added to the view at lowest resolution. The transmission of rendering-related data continued as before, with the exception that now objects 2 and 10 needed to split the available client downstream bandwidth based on the weight values that were assigned to them by the run-time LoD selection scheme. The RTMO data for object 2 was already received during the first experiment interval, so it first had to wait for the transmission of the IBR models of objects 8, 9 and 10 to complete before it could continue with upgrading its own representational fidelity (through the reception of PM data).

9.4.2 Representative Experiment

The second experiment involved the transmission of a complex virtual scene and hence demonstrates how the proposed bandwidth management scheme behaves in lifelike situations. Furthermore, besides rendering-related data, the experiment required the simultaneous distribution of real-time video traffic. The client's downstream access bandwidth was in this case set to the more common value of 100 KBps.

Figure 9.8 plots the yielded data reception at client-side. Due to the large number of virtual models that were involved in this experiment, the bandwidth usage of individual non-real-time data flows is no longer displayed. Instead, the graph visualizes the aggregate bandwidth which was consumed by each category of rendering-related data (i.e., IBR, compressed base mesh with texture data, PM L1, PM L2 and PM L3).

The network trace illustrates that at the start of the experiment, IBR data was given full priority so that the NVE application could quickly provide the user with an initial, low-quality view of the virtual world. Once these

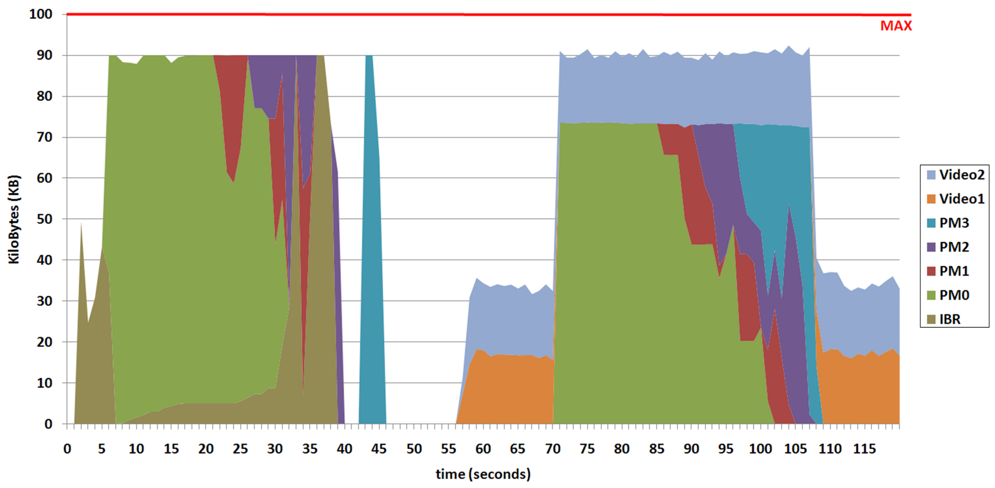


Figure 9.8: Stacked graph illustrating the client bandwidth distribution during a realistic scenario involving the reception of both 3D model data and real-time video traffic (in KBps).

RTMO representations were completely received, model data was incrementally upgraded based on the geometric resolution levels that were selected by the NVE's LoD manager. After approximately 35 seconds, most PM data that was needed for the higher-fidelity rendering of objects in the frustum was delivered to the client and, as a result, the bandwidth amount that was assigned to the IBR pre-loading of models outside the user's field of view was drastically increased.

At around 55 seconds, two video sources entered the user's area of interest. Each video stream required a bandwidth of approximately 17 KBps. Recall from section 9.3.1 that the NIProxy reserved only 30 percent of the total client downstream bandwidth for the reception of real-time network traffic. This amount did not suffice to simultaneously propagate both video streams, but since there was initially excess bandwidth from the non-real-time network traffic category, the real-time network flows were allowed to exceed their allocated bandwidth share (see section 4.2.3). Roughly 15 seconds later however, the user rotated 90 degrees in the virtual world, which caused model importance to alter due to the user's modified viewing direction. As a result, new geometric model data needed to be delivered to the client and hence competition for the available client downstream bandwidth between real-time and non-real-time network traffic was introduced. Due to this contention, the real-time network traffic now needed to conform to its appointed bandwidth percentage, which resulted in the NIProxy temporarily blocking one of the video streams to prevent the non-real-time network traffic from being denied its fair bandwidth share. In particular, during the contention period, the network flow for video 2 was retained to the prejudice of video 1 because of the former's higher scene importance and hence larger weight value in the client's stream hierarchy. Only after the transmission of the 3D model data had finished, the simultaneous forwarding of both video streams was resumed.

9.4.3 Discussion

The experimental results comprehensively demonstrate the benefits and capabilities of the NIProxy's network traffic shaping functionality. A first important observation is that over-encumbrance of the client's access connection was consistently prevented. As a result, packet delay and loss were minimized and an optimal data reception at client-side was achieved. Notice that a small fraction (i.e., 10 percent) of the client's downstream bandwidth capacity was even left unused in the experiments. As was discussed in section 4.6, this unallocated amount served as safety margin to guarantee a certain level of resilience to uncontrolled surges in the bandwidth consumption of the network flows which constituted the traffic mix that was forwarded to the client. In the second experiment, this overflow protection buffer successfully countered the slight variance in downstream throughput of the real-time video traffic and as such prevented these bandwidth consumption fluctuations from negatively impacting data dissemination efficiency and hence user QoE[1]. Secondly, the requirements of the considered NVE application and, in particular, of its rendering scheme were successfully captured by the bandwidth distribution strategy that was implemented by the NIProxy. For instance, the presented network traces prove that, by assigning priority to the transmission of IBR data, the NVE's aim of quickly providing the user with an initial view of the virtual world was satisfied. Finally, the second experiment has indicated the NIProxy's ability to correctly deal with situations in which real-time and non-real-time network traffic contend for the downstream bandwidth that is available to the client. In particular, by employing a `Percentage` node to differentiate between real-time and non-real-time network flows in the client's stream hierarchy, both traffic categories received their fair bandwidth share throughout the entire experiment.

[1]As was already mentioned in section 8.3.2, the NIProxy's rate control functionality for non-real-time data results in this type of network traffic behaving in a very controlled fashion in terms of downstream bandwidth consumption. The safety buffer consequently has no utility for non-real-time traffic.

Recall from section 9.2.3 that the results that were attained by the NIProxy's network traffic shaping framework are beyond the reach of the unmodified version of the NVE application:

- The standard implementation does not include constructs for throttling the downstream data transmission to clients; it merely appeals to the elastic behavior of non-real-time network flows to avoid congestion on clients' access links. As was already established in chapter 8, this approach is hardly effective and entails a number of significant pitfalls. As an example, it will lead to bandwidth capacity violations in case the combined bandwidth consumption of the inelastic (i.e., real-time) traffic exceeds the throughput of the access link. As another example, it might lead to the starvation of the non-real-time network traffic.

- Client downstream bandwidth is assumed to be plentifully available. Consequently, facilities for coordinating the distribution of network traffic are lacking, let alone for orchestrating the simultaneous dissemination of real-time and non-real-time content.

- The NVE's built-in communication scheme does not succeed in effectively fulfilling the application's requirements and demands concerning the dissemination of rendering-related non-real-time content (e.g., reserving bandwidth for IBR pre-loading, enabling quick visualization of an initial (low-quality) virtual world representation, etcetera). In more detail, whether these requirements will be satisfied largely depends on external factors such as scene complexity (since the probability of data dissemination issues is directly proportional to model count), the user's movement pattern in the virtual environment (quick movement will cause significant and rapid changes in model reception priority, which is not coped well with by the default communication scheme) and whether or not there is contention from real-time traffic.

Based on these observations, it is clear that the introduction of the NIProxy and its network traffic shaping operations positively influenced the QoE that is perceived by the users of the considered NVE application.

9.5 Related work

A survey of related work on network traffic shaping was already presented in section 2.4.1. This section will therefore only briefly review proposed techniques and solutions for the topic on which this chapter has primarily focused, namely the efficient and adaptive network transmission of 3D rendering-related data. Interesting related work in this subdomain of automatic bandwidth brokering includes the research by Ioana Martin [Schneider 99, Martin 00, Martin 02, Boier-Martin 03], the 3D models Transport Protocol (3TP) [AlRegib 05] and the generic middleware for the streaming of 3D progressive meshes described by Li et al. in [Li 06].

The cited approaches concentrate exclusively on optimizing the delivery of 3D models over transportation networks and hence provide advanced techniques which are targeted specifically at this type of multimedia content. This implies however that they do not cover scenarios which require the simultaneous transmission of model data and other types of multimedia, such as real-time video or P2P data. Some of these systems are even so concentrated that they consider only one particular type of 3D content. As an example, the 3TP approach is only applicable to progressively compressed 3D models and hence lacks support for streaming other types of rendering-related data, like textures or RTMO representations. In contrast, the NIProxy approaches network traffic shaping from a more generic perspective and supports bandwidth management in the presence of different types of real-time as well as non-real-time network traffic. As a result, the mechanisms for the dissemination of rendering-related data that are supported by the NIProxy are less sophisticated compared to those of the referred systems. At the same time however, the NIProxy's more general methodology guarantees a much wider applicability. In particular, by not focusing solely on orchestrating the delivery of rendering-related data, a large variety of distributed applications may exploit and benefit from the NIProxy's QoE optimization features.

9.6 Conclusions

This chapter has again discussed the NIProxy's ability to improve the network dissemination of multimedia content. In particular, findings and observations which stem from employing the NIProxy to manage the downstream bandwidth of clients of a real-world NVE application have been presented. The considered application demands efficient distribution of rendering-related data and in addition supports real-time streaming of audiovisual content. Via representative experimental results, the NIProxy's ability to capture the prerequisites which are imposed by this particular application and to translate them into an effective bandwidth brokering strategy has been demonstrated. As such, this chapter has confirmed that the NIProxy's QoE optimization potential is not limited to artificial demonstrator software but can just as well be exploited by realistic, concrete distributed applications with potentially non-straightforward requirements. Stated differently, this chapter has corroborated that the NIProxy's NTS provisions are sufficiently sophisticated and elaborate to enable the bandwidth management stipulations and expectations of complex applications to be satisfied. As a secondary contribution, the presented results have indirectly highlighted the amount of flexibility that is afforded by the NIProxy's generic approach to client bandwidth management, which enables it to be integrated in a multitude of distributed applications.

Chapter 10

Outbound Traffic Engineering

All practical evaluations that have already been presented in this part of the dissertation were solely concerned with optimizing the dissemination of data that is *destined for* users of distributed applications. According to the categorization that was introduced in section 3.3, such data follows the inbound flow direction. It is evident that the engineering of inbound network traffic has a substantial and direct effect on the QoE that is perceived by users. The NIProxy therefore initially focused its traffic engineering operations on this traffic category.

In the course of this PhD research, it was however determined that the opposite flow direction might also hold possibilities in terms of user QoE optimization. This chapter will consequently report on the incorporation of support in the NIProxy for managing data that *originates from* a connected client (instead of being destined for it) [Wijnants 09a]. In section 3.3, this traffic direction was labeled outbound. In particular, it will be described how the NIProxy's dual user QoE optimization tools, network traffic shaping and multimedia service provision, which had previously been applied exclusively in the inbound direction, were translated to outbound equivalents. Finally, via a multimedia streaming case study, the benefits and implications with regard to user experience improvement that are enabled by upstream network traffic shaping and the provisioning of outbound services will be validated.

10.1 Architectural Modifications

In the initial version of the refactored NIProxy implementation, the application of its supported traffic engineering techniques was confined to inbound network traffic. This limitation was caused by the NIProxy's software architecture, which initially included only an (inbound) packet processing chain for processing network packets that are destined for a NIProxy client. Based on the conviction that incorporating outbound traffic engineering functionality might increase the QoE optimization potential of the NIProxy, it was soon after the refactoring operation decided to introduce a similar construct for outbound data. As was already discussed in section 6.2.2, the outbound packet processing chain largely mimics its inbound equivalent in terms of composing components as well as operating procedure. In contrast to the latter however, it ought to be traversed by network packets that are transmitted by a NIProxy-connected client and hence implements network traffic shaping and multimedia service provision for data that follows the upstream/outbound flow direction.

Besides the introduction of the outbound packet processing chain, only minor adjustments were required to the implementation of the Bandwidth Manager and Service Manager software components to incorporate outbound traffic engineering support in the NIProxy. As was discussed in section 6.2.2, these components respectively encapsulate the NIProxy's bandwidth brokering and service delivery frameworks. The Bandwidth Manager previously maintained only a single stream hierarchy, based on which a connected client's inbound network traffic was shaped. To also enable the orchestration of the upstream bandwidth consumption of distributed applications, a supplementary stream hierarchy was introduced. This implies that the current Bandwidth Manager implementation simultaneously administers two separate stream hierarchy instances which form the basis for managing the client's downstream and upstream network traffic, respectively. Analogously, the Service Manager was redesigned so that it could distinguish between inbound and outbound services. By concurrently managing an independent service chain for both flow directions and by dynamically determining which service list to apply based on each intercepted network packet's flow direction, support for outbound service provisioning was achieved.

Since the outbound packet processing chain was modeled upon the inbound variant, it has inherited the collaboration interface that is provided between services and the network traffic shaping framework (see section 6.2.2). In particular, the Service Manager component exposes an interface through which outbound services can communicate with the Bandwidth Manager and Stream Manager instances that are included in the encompassing outbound packet processing chain. Analogous to the inbound implementation, it is hence enabled for outbound services to query and even adjust the upstream stream hierarchy which steers the shaping of the network traffic that is emitted by the associated NIProxy client. As has already been demonstrated a number of times in previous chapters and as will again be confirmed in section 10.4, supporting interaction between the provided traffic engineering tools enables the NIProxy to elevate its user QoE optimization capabilities to a performance level that surpasses the results that are attainable when both mechanisms are applied independently. As a final remark in this context, cooperation between traffic engineering techniques which concentrate on different flow directions is just as well supported. It is consequently possible for an outbound service to interoperate with the inbound network traffic shaping scheme (and vice versa). This is due to the Bandwidth Manager component simultaneously maintaining both the downstream and upstream variants of the associated client's stream hierarchy. At the time of writing, it has not yet been validated whether this option is advantageous or even appropriate. If at some point it would be identified as undesirable, for instance because it might jeopardize correct NIProxy operation, this possibility could easily be disabled.

10.2 Conceptual Implications

Recall from section 3.3 that the inbound/downstream and outbound/upstream classification is defined from the point of view of a NIProxy client. Supposing that the NIProxy is positioned on the logical borderline between the network core and the access network (which section 3.5 has identified to be an auspicious location for NIProxy deployment), the flow direction definition implies that

- inbound traffic engineering will be applied to data that is received by the NIProxy on its connection with the network core, with the purpose of streamlining the intercepted data's downstream delivery over the destined client's access link
- network traffic that is emitted by the NIProxy client itself will be subjected to outbound traffic engineering, which will regulate its upstream injection into the network backbone

The difference between inbound and outbound traffic engineering can also be explained from the perspective of the lifetime of network traffic. Assuming again a setup in which the NIProxy is deployed along the core/access network boundary, inbound network flows will be engineered near the end of their passage through the transportation network (i.e., at a time when the transported data has nearly reached its final destination); in contrast, the engineering of outbound data will occur close to its source (i.e., soon after it originated from the NIProxy client).

10.3 Use Case: Outbound Static Video Transcoding

This section will present an example outbound service that will serve as practical use case to demonstrate and evaluate the NIProxy's upstream network traffic shaping and outbound multimedia service provision facilities. The service can be regarded as the translation of the static video transcoding service which was introduced in section 5.3 to the outbound flow direction. In particular, the service introduces static outbound transcoding functionality in the NIProxy and hence enables it to reduce the bitrate of outbound H.263-encoded video traffic. Recall from section 5.3 that the static keyword refers to the fact that the video transcoding parameters need to be specified on service instantiation and will remain fixed until the instance is destructed. Given the fact that the outbound video transcoding service is a mostly straightforward adaptation of the inbound equivalent, implementation-wise both variants are largely resemblant. This section will therefore not reiterate the technical details of the outbound service's implementation but will instead focus on its general modus operandi and on its interoperation with the NIProxy's (outbound) bandwidth brokering actions.

10.3.1 Mode of Operation

The service expects as input H.263 video streams that are intercepted by the NIProxy on the outbound flow direction (i.e., frames which are emitted by the NIProxy client that is associated with this service instantiation). The output that is produced by the service equals either the Original Version (OV) of the outbound video stream or its Transcoded Variant (TV). To determine which video fidelity to output, the service consults the current upstream bandwidth distribution strategy which the NIProxy has calculated for the video source. More specifically, the service exploits its interface with the Bandwidth Manager to access the video source's upstream stream hierarchy and to verify whether the representation of the transcoded version of the video stream is currently assigned upstream bandwidth. The decision whether to perform transcoding is hence dictated entirely by the NIProxy's network traffic shaping scheme. As a result, unnecessary transcoding operations are eliminated, which is an important achievement since video transcoding is a computationally complex task which demands considerable amounts of scarce processing power.

10.3.2 Stream Hierarchy Manipulation and Awareness Extension

Section 4.7 has clarified that the bulk of the responsibility for composing and managing the stream hierarchy lies with the application software (i.e., the NIProxy client). This implies that the video source is responsible for ensuring that the video streams which it transmits are adequately incorporated in its upstream stream hierarchy The transcoded version of these streams however do not originate from the video source itself but instead are generated on-the-fly by the outbound video transcoding service. The service consequently introduces a new type of outbound network traffic, which should also be represented in the video source's upstream stream hierarchy. Therefore, analogous to its inbound counterpart (see section 5.3.1), the service executes the following actions on the detection of an outbound video flow:

- A new `discrete` leaf node is instantiated and linked to the transcoded variant of the discovered video flow
- The newly created node is included in the upstream stream hierarchy as direct sibling of the node which corresponds to the original video version

Once the transcoded quality of the outbound video flow has been incorporated in the stream hierarchy, the NIProxy's network traffic shaping framework will start considering it when managing the video source's upstream bandwidth capacity. Finally, besides updating the upstream stream hierarchy, the service also extends the NIProxy's network awareness repository by supplying the Stream Manager component with information regarding the bandwidth requirements of the transcoded version of outbound video flows.

10.4 Evaluation

In this section, the impact of the NIProxy's outbound traffic engineering functionality on its user QoE optimization capabilities will be evaluated by analyzing the outcome of two distinct experiments. Before presenting and discussing

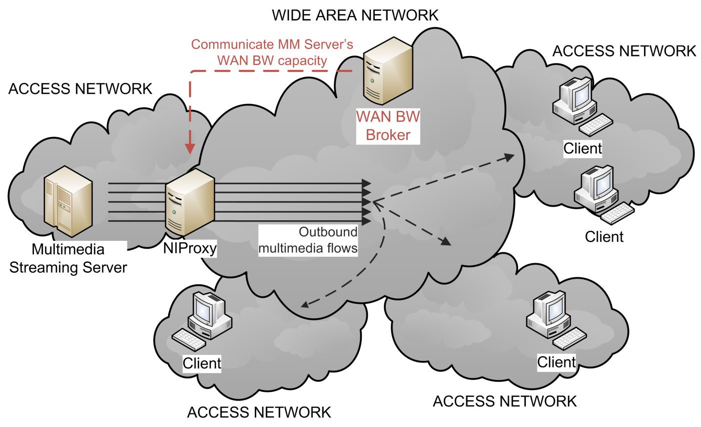


Figure 10.1: Emulated network environment in which the NIProxy's outbound traffic engineering support was evaluated.

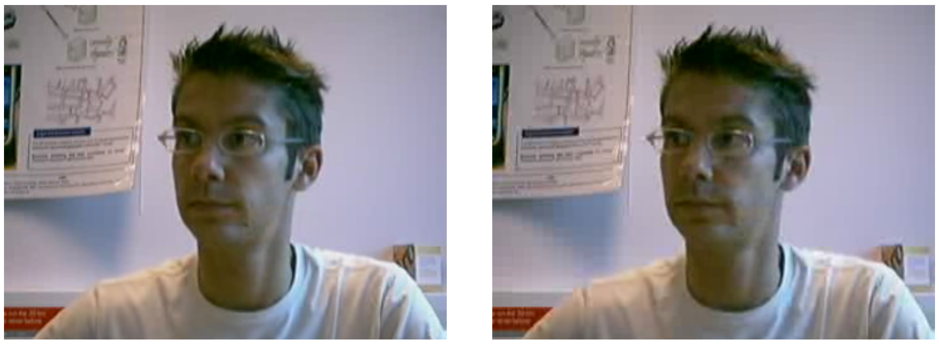
the actual results, a concise description of the setup in which the experiments were conducted will be provided.

10.4.1 Experimental Setup

To evaluate the conversion of the NIProxy's network traffic shaping and multimedia service provision functionality to the upstream/outbound flow direction, a limited multimedia streaming scenario in a Wide Area Network (WAN) setting was simulated. The experimental setup involved a single server and multiple clients. Based on incoming requests, the server unicasted multimedia data to clients using RTP. Besides server and clients, the setup also included a NIProxy instance. An important contrast with the experimental case studies that have been presented up to now is that the NIProxy instance was in this case not associated with the client(s) but was conversely responsible for shaping the multimedia traffic that was emitted by the server. Lastly, the existence of an entity was assumed that managed and regulated the allocation of the bandwidth capacity of the network backbone. This entity will be referred to as the *WAN bandwidth broker*. In the experiments, the bandwidth broker hence determined the amount of backbone bandwidth which the multimedia streaming server could maximally consume and subsequently communicated this information with the NIProxy instance. The WAN bandwidth allocation decision could, for instance, be grounded on a Service Level Agreement (SLA) that was negotiated between the content provider (i.e., the proprietor of the multimedia streaming server) and the operator of the WAN.

Table 10.1: Quality settings of the original and transcoded video versions.

	Original	**Transcoded**
Resolution (pixels)	352 × 288 (CIF)	352 × 288 (CIF)
Framerate (FPS)	25	15
Bitrate (bps)	160000	80000
Codec	H.263	H.263



(a) Original (b) Transcoded

Figure 10.2: Qualitative comparison of the original and transcoded version of a particular video fragment.

An overall picture of the experimental setup is provided in Figure 10.1. By conceptually positioning the multimedia streaming server and the clients in disjoint access networks, it was guaranteed that all of the server's outbound network traffic needed to traverse the WAN and hence consumed backbone bandwidth. Also notice that the NIProxy instance was deployed at the end of the server's access connection, this way enabling it to adapt and shape the server's outbound network traffic before it reached the WAN.

In the experiments, the NIProxy instance made use of the outbound static video transcoding service which was introduced in section 10.3. The quality settings of the video content that was stored on the multimedia server as well as the configuration parameters for the transcoding service are specified in Table 10.1. As can be inferred from this table, video transcoding resulted in a reduction of the video stream's temporal resolution; at the same time, the video compression ratio was increased (as the service employed a halved target bitrate). In contrast, the spatial resolution of the input video stream was left intact during transcoding. The video transcoding service hence outputted video fragments whose spatial resolution was identical to the original, but which were less fluid and which exhibited a lower visual quality. Figure 10.2 exemplifies the disparity in image quality between both video versions. As can be seen, the difference is marginal for low-complexity video fragments.

To conclude, remark that the setup could have easily been extended with additional NIProxy instances to perform downstream bandwidth management and inbound service delivery for the clients that were involved in the experiments. It was opted not to do so to ensure that the reader's attention is focused on the NIProxy's outbound traffic engineering tools and to enable clear and direct deduction of their impact from the generated experimental results. It is apparent however that such an extended environment would enable a number of additional options in terms of user QoE optimization and would hence most likely allow for further improvement of the achieved results.

10.4.2 Experiment 1: Simultaneous Audio and Video Streaming to a Single Client

Experiment Description

In the first experiment, the multimedia server simultaneously streamed an audio track and a video fragment to a single client. The objective for the NIProxy instance which was included in the experimental setup hence consisted of ensuring that the WAN bandwidth which the broker had reserved for the server was apportioned deliberately among these two network flows. The server's WAN bandwidth capacity was in addition made subject to considerable fluctuations, which conceptually partitioned the experiment into a number of discrete periods. This was done to introduce dynamism in the emulated network environment so that it could be ascertained whether the NIProxy was able to successfully cope with and respond to changing contextual parameters. In practice, such fluctuations could for instance be caused by the arrival of new network traffic that needs to be accommodated by the WAN.

The (upstream) stream hierarchy which steered the NIProxy's bandwidth brokering behavior during the experiment is illustrated in Figure 10.3. Due to the experiment's restricted scope, the stream hierarchy exhibited an elementary structure and contained only a limited number of nodes. In particular, the root of the hierarchy consisted of an internal node of type `Priority` and directly distinguished the audio and video flow which were emitted by the multimedia server from each other. Since there was no notion of multiple audio

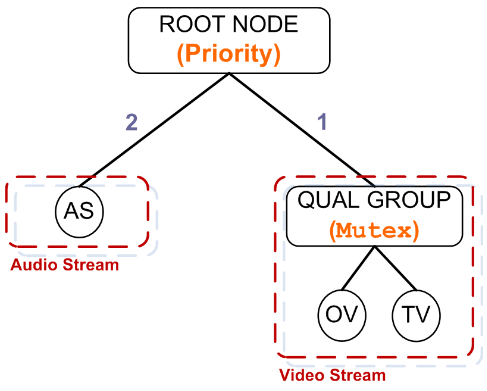


Figure 10.3: The multimedia streaming server's upstream stream hierarchy in experiment 1.

versions in the experiment, the Audio Stream (AS) was made a direct child of the root as a `discrete` leaf node. The video flow on the other hand was available in two distinct qualities (i.e., the Original Version (OV) as emitted by the multimedia server and a Transcoded Variant (TV) that was generated by the NIProxy's outbound static video transcoding service). Remember that both versions transport identical content and differ only in their quality parameters. In the experiment, it would have consequently been wasteful in terms of WAN bandwidth consumption in case the client would ever receive both video flows simultaneously. Following the approach that was proposed in section 5.3.1, their corresponding `discrete` leaf nodes were therefore grouped together using a `Mutex` internal node before they were added to the hierarchy root as a child. Also, as was described in section 10.3.2, the TV leaf node was not instantiated by the multimedia server but instead by the NIProxy's outbound static video transcoding service. As a final comment, each of the leaf nodes in the constructed stream hierarchy defined 2 discrete bandwidth consumption levels so that they could either completely disable the network flow which they represented or enable it at maximal bitrate (see section 4.3.1).

During the experiment's setup phase, the client indicated to the multimedia server that audio was preferred to video. The server in turn informed the NIProxy of this application-related information by ensuring that it was adequately captured in its upstream stream hierarchy. In particular, this was achieved by attaching a higher priority value to the leaf node which represented the audio flow. These priority values, and by extension the entire constitution of the stream hierarchy, remained constant for the entire duration of the experiment; only the bandwidth amount that was allocated to the root node

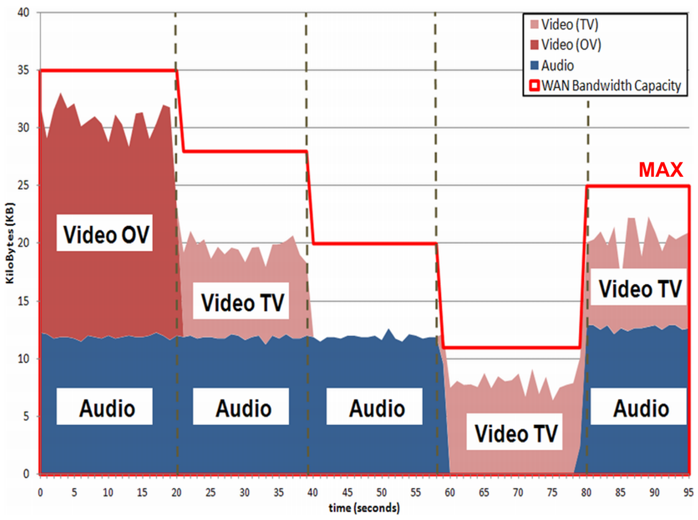


Figure 10.4: Plot (stacked graph) of the multimedia streaming server's WAN bandwidth consumption during experiment 1.

changed over time (i.e., at the beginning of each new experiment interval, based on the bandwidth capacity information that was provided by the WAN bandwidth broker).

Results

Based on the just described stream hierarchy, the NIProxy managed the multimedia server's upstream bandwidth consumption as is illustrated in Figure 10.4. In this network trace, the dashed vertical lines separate the different experiment intervals. As can be seen, interval transitions always coincided with an alteration of the amount of upstream bandwidth that was available to the multimedia streaming server. Also note from the trace that the original and transcoded versions of the video flow respectively demanded more and less bandwidth than the audio stream.

Analysis of the network trace unveils that the WAN transmission of the audio track was prioritized throughout the experiment. Only during the fourth interval, the audio stream was not forwarded over the WAN. This was however caused solely by a lack of sufficient upstream bandwidth, not because precedence was given to the streaming of the video fragment. Any upstream bandwidth that was not claimed by the audio flow was subsequently employed by the NIProxy to implement the video streaming. In case the residual WAN bandwidth was inadequate to stream the original video version, the switch to the lower-quality transcoded version was made. As a result, the client always received the video fragment at the highest quality possible, given the server's current upstream bandwidth capacity.

10.4.3 Experiment 2: Simultaneous Video Streaming to Multiple Clients

Experiment Description

In contrast to the first experiment, the second involved multiple clients which each requested a particular video fragment from the multimedia streaming server. No other type of multimedia content (i.e., audio) was included in this experiment because doing so would have complicated the produced results without however providing any additional insight in the added value of the NIProxy's outbound traffic engineering support. Also contrary to the first experiment, the multimedia server now disposed of a steady WAN bandwidth capacity of 50 KiloBytes per second (KBps). The experiment nonetheless exhibited a dynamic aspect, which was this time caused by the arrival and departure of clients during experiment execution. This resulted not only in the initiation and suspension of outbound server flows at run-time, but also in shifts in individual stream importance in the course of the experiment. The explanation for this latter effect is given by the assumption that contracts existed between the multimedia streaming server and its clients, causing clients to be categorized as either *regular* or *premium* users. The relationship between both client categories was such that, compared to regular users, premium users should receive an improved service and a preferential treatment from the server, whenever possible.

The dynamic joining and leaving of clients again divided the experiment into a number of consecutive intervals. In particular, in the first interval the server's client list consisted of two regular users, RU1 and RU2. The second period was initiated by the arrival of a third regular user (RU3), while the third interval commenced when a premium user (PU) requested a video fragment from the multimedia server. Finally, the transition to the fourth interval was triggered by the departure of regular user RU2.

Due to the variability of its client list, the multimedia server was forced to update its upstream stream hierarchy several times during the experiment. This stream hierarchy is depicted in Figure 10.5 and its construction scheme can be considered as a generalization, to a multi-client environment, of the approach that was taken in the first experiment. In particular, this time a root node of type `Percentage` was used, not to discriminate between the different network streams that were requested by a certain client but instead

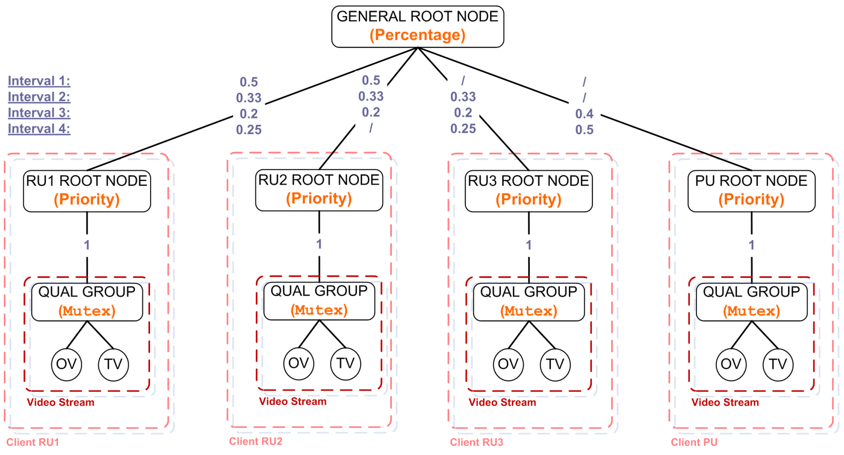


Figure 10.5: The multimedia streaming server's upstream stream hierarchy in experiment 2.

between the server's currently connected clients. Whenever a client joined the experiment, it was represented in the stream hierarchy analogously to the method from section 10.4.2. More specifically, on each new client connection, a subtree with a root node of type `Priority` was instantiated and added to the global root node; all ensuing requests for content that were issued by the client were incorporated in its designated subtree as children of this `Priority` node. Finally, Figure 10.5 also illustrates the percentage values that applied during the different periods of the experiment. These values were calculated by the multimedia server based on the contracts that were concluded with its clients. In particular, to favor them over regular users, a twice as high percentage value was appropriated to premium users.

A remark concerning the role of the intermediate `Priority` nodes in the stream hierarchy is in place. Since in this experiment all clients requested exactly one video fragment from the server, each such node had only a single child. As a result, their presence was in this case irrelevant to the shaping of the multimedia server's upstream network traffic. Stated differently, the role of the `Priority` nodes was limited to integrally relaying the bandwidth amount which they were granted by the global root node to their sole child. A similar observation applies to the method for priority value determination. This method was inherited from the first experiment (i.e., associate a priority value of 1 with video and 2 with audio) and, again due to the lack of peer child nodes, did not have any influence on the outcome of the bandwidth brokering process either.

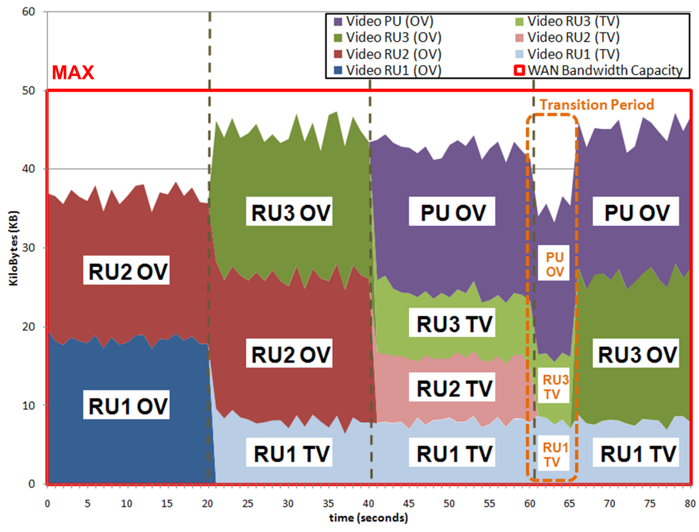


Figure 10.6: Plot (stacked graph) of the multimedia streaming server's WAN bandwidth consumption during experiment 2.

Notice that Figure 10.5 displays the multimedia server's upstream stream hierarchy in its most elaborate form. In particular, the depicted hierarchy corresponds to the instantiation during the third experiment interval, the stage at which all clients which were involved in the experiment were simultaneously active. In the first period, only the RU1 and RU2 subtrees were present. At the start of the second and third period, the multimedia server extended its upstream stream hierarchy with the representations for RU3 and PU, respectively. Finally, when the fourth experiment interval commenced, the RU2 subtree was removed.

Results

The server's WAN bandwidth consumption during the second experiment is plotted as a stacked graph in Figure 10.6. The boundaries of the experiment's discrete intervals are again visualized using dashed vertical lines. In the first interval, sufficient WAN bandwidth was available to stream to all currently connected clients (i.e., regular users RU1 and RU2) the video fragment which they had requested, at original quality. This ideal situation changed at the beginning of the second interval, when regular user RU3 joined the experiment and also issued a streaming request. The NIProxy therefore invoked the outbound static video transcoding service for client RU1 and commenced forwarding the transcoded version of the video stream that was destined for this client over the WAN. Notice that since all clients which were involved in this phase of the experiment had an equal importance (i.e., it were all regular users), the NIProxy had to arbitrarily select a client to "penalize". In this case, this turned out to be RU1. The third interval was initiated by the arrival of premium user PU. This resulted in the downscaling of the video streams which were intended for the three regular clients to transcoded quality, this way freeing up sufficient WAN bandwidth to accommodate the original version of the video fragment that was requested by PU. Finally, the departure of regular user RU2 at the transition to the fourth interval allowed the NIProxy to upgrade the video stream for RU3 back to original quality. The decision to favor RU3 instead of RU1 was again made randomly (i.e., since both clients were at that time assigned an identical percentage value, they were equally entitled to the freed WAN bandwidth). Also note the short period of time at the start of the final experiment period during which the server's upstream bandwidth distribution was non-optimal. In this small transition interval, which is highlighted in the network trace using a dashed rectangle, the available WAN bandwidth capacity was not exploited as completely and effectively as possible. This is explained by the fact that the NIProxy needed some time to ascertain that RU2 had actually left the experiment.

10.4.4 Discussion

The achieved results firstly confirm that the NIProxy-managed client (i.e., the multimedia streaming server) at all times respected its allocated WAN bandwidth capacity. In case this bandwidth amount would have been determined on the basis of a SLA, the financial repercussions that are typically associated with contract infringement would hence have been avoided. In addition, by accurately complying with the WAN bandwidth allocation strategy as devised by the bandwidth broker, the NIProxy contributed to WAN congestion prevention. This in turn is likely to have had a positive influence on the experience that was provided to the users for which the server's upstream network traffic was destined, since the absence of congestion increases the stability of the network and improves the predictability of its performance. Secondly, besides ensuring that the upstream bandwidth capacity was not exceeded, the NIProxy at the same time attempted to maximize its utility. In particular, guided by its application awareness, the NIProxy distributed the WAN bandwidth that was reserved for the multimedia streaming server in an intelligent and effective manner over the server's collection of outbound network flows. In the first experiment, for instance, the receiving user's preference for audio was successfully reflected in the server's upstream bandwidth consumption behavior. In the multi-user test case on the other hand, the NIProxy guaranteed that the server's premium user received a preferential treatment compared to its regular peers, whenever possible. It is intuitively apparent that the effort that was invested in the coordination and optimization of the WAN bandwidth consumption of the multimedia streaming server likely positively affected the QoE that was witnessed by the server's users. Third, the presented experimental results have exemplified the ability for outbound multimedia services and the NIProxy's network traffic shaping framework to collaborate and the interesting capabilities which it unlocks. In particular, the availability of the outbound static video transcoding service extended the NIProxy's range of network traffic shaping options by enabling the streaming of lower-quality video versions over the network backbone in the event of upstream bandwidth shortage. The anticipated positive effect hereof on user QoE has already been mentioned in section 7.3 for the inbound flow direction and was here achieved for outbound network traffic. A final minor observation from the practical evaluation is that the NIProxy succeeded in effectively coping with dynamic events, whether network- or application-related.

Based on the analysis of the experimental results, it becomes clear that the NIProxy's outbound traffic engineering functionality offers opportunities in terms of user QoE optimization on two fronts:

NIProxy-connected hosts In the presented case study, the NIProxy managed the outbound traffic of the multimedia streaming server. As a result, it was to be expected that this host would benefit from the NIProxy's functionality. In this case, the NIProxy for example prevented the server from violating its WAN bandwidth capacity and hence from incurring possible financial penalties that are imposed by the WAN operator.

Unmanaged hosts A less straightforward observation is that hosts which are not directly managed by the NIProxy might also be positively impacted by its support for outbound traffic engineering. In particular, the experimental findings have proved the engineering of the data that was streamed by the content server to be indirectly advantageous to the users for which the data was intended.

10.5 Conclusions and Future Work

Prior to this chapter, the NIProxy's network traffic shaping and multimedia service provision functionality had been applied exclusively on data that was destined for managed hosts. This chapter has described the extension of the coverage of these traffic engineering methods to data that is injected into the transportation network by NIProxy-connected hosts themselves. Care was taken to ensure that the collaborative interface that is provided between both mechanisms in the inbound direction was retained during the generalization process. Besides presenting the required architectural modifications, the conversion of the static video transcoding service from section 5.3 to the outbound flow direction was described. Stated more elaborately, this service enables the NIProxy to manipulate the quality and hence bitrate of video data that originates from NIProxy-managed hosts. Using the outbound video transcoding service as practical use case, the NIProxy's support for upstream network traffic shaping and outbound multimedia service provision was experimentally evaluated. Representative test results have clearly verified that the outbound traffic engineering functionality helps the NIProxy in delivering on its objective of improving the multimedia capabilities of IP-based networks.

Although support for upstream network traffic shaping and outbound multimedia service provision was obtained in the NIProxy by fairly straightforwardly mimicking their inbound precursors, a notable difference in their user QoE optimization reach has emerged from the presented experimental results. In particular, prior empiric findings have shown inbound traffic engineering to be mainly beneficial for the targeted user (i.e., the user for which the network traffic is destined). In contrast, the experimental results in this chapter have demonstrated that hosts which are not explicitly managed by the NIProxy might also (indirectly) profit from its outbound traffic engineering actions. As an example, in the presented media streaming case study, none of the end-users were connected to a NIProxy instance, yet all of them took advantage of the engineering of the outbound network traffic of the server from which they requested their content. Since the server itself also reaped rewards of the manipulation of its emitted network traffic, the NIProxy's outbound traffic engineering facilities offer user QoE enhancement opportunities on two fronts. Support for outbound traffic engineering has hence been proven to be a promising and very versatile tool in the quest for QoE optimization and therefore represents a useful extension of the NIProxy's feature list.

This chapter has merely scratched the surface of the QoE optimization options that are unlocked by outbound traffic engineering. Especially due to the fact that it allows for improving the experience of not only directly managed but also unmanaged hosts, outbound traffic engineering possesses a very extensive range of possible applications. An obvious topic of future work in the short term is a more thorough investigation of the prospects of outbound service provisioning through the implementation of additional outbound NIProxy services. An interesting candidate would be, for example, a service which adds encryption to outbound network flows, this way protecting the confidentiality of sensitive information in case it needs to be transmitted over a potentially untrustworthy interconnection network (e.g., the Internet). In the more distant future, it might be very valuable to experiment with a network setup which encompasses multiple NIProxy instances to simultaneously control on the one hand the upstream bandwidth consumption of multimedia sources and on the other hand the last mile downstream delivery of multimedia content to sinks. As was already mentioned in section 10.4.1, it is expected that such an extended setup will be able to push the results in terms of user experience optimization that were presented in this chapter to a higher level.

Chapter 11

FEC-Integrated Network Traffic Shaping

Data corruption incurred during network transmission is likely to result in user frustration and hence plays a significant role in the optimization of the satisfaction of users of distributed applications. As a countermeasure, Forward Error Correction (FEC) schemes complement source data with redundant information to enable data recovery at destination-side. This chapter will discuss the context-aware inclusion of adaptive parity-based FEC support in the NIProxy [Wijnants 09b]. Since FEC coding increases the load on the transportation network, it will be shown that an integrated approach with the NIProxy's network traffic shaping mechanism was adhered to to enable deliberate control over the FEC process and to guarantee that the bandwidth overhead which it introduces is justified. Representative experimental results will be presented which will confirm the validity of the selected approach. The results will in addition prove that, if applied in a well-considered manner, FEC coding is able to beneficially affect user experience and consequently forms a valuable addition to the NIProxy's toolset.

11.1 Introduction

Exchanging data over packet-switched computer networks can lead to corruption, which renders the data (partly) unusable for the receiver. The type and amount of errors that are incurred during transmission depend on multiple factors, including the kind of networking technology used, the current traffic load and the characteristics of the transported data. Broadly speaking however, data corruption can be caused by either the loss of entire packets or by the introduction of bit errors. Insufficiently capacitated network infrastructure (e.g., routers) is a frequent source of packet loss. Transmission errors on the other hand typically result from signal interference or noise on the communication channel, which are common issues in wireless networks. Irrespective of its cause, data corruption is likely to have a detrimental effect on user experience and hence effort should be made to minimize it. Lost or contaminated data can for instance lead to hitches in voice streams or visual artifacts during video playback.

Solutions for dealing with data corruption can be categorized into two groups. The first category consists of *retransmission-based* techniques, where the receiver monitors incoming network flows and requests the source to retransmit missing or corrupted data [Tanenbaum 02]. This process can in theory be repeated indefinitely until all data has been correctly received. In practice however, timing constraints might limit the number of acceptable retransmission attempts for a certain data item or might even render retransmissions completely unfeasible. Retransmission-based techniques are hence mainly suitable for recovering errors which are introduced in delay-tolerant network traffic like for instance FTP file transfers. In the *Forward Error Correction* (FEC) approach on the other hand, the sender accompanies the source data with redundant information which allows the receiver to repair, to a certain extent, errors that are introduced during transmission [Moon 05]. FEC schemes in other words enable the destination to recover lost or damaged data without incurring the round-trip time delay overhead of retransmission-based solutions. This is an important advantage in case the network traffic has real-time characteristics like for instance a live video feed. Finally, it is worth noting that for several media types techniques have been devised which do not attempt to repair compromised data, but instead try to hide the errors which the corruption introduces. While such *error concealment* techniques are certainly capable of alleviating the effects of data corruption, they will never be able to fully compensate them. These algorithms are therefore typically deployed on top of the data recovery scheme as an additional layer of protection and a measure of last resort.

Despite their radically different methodology, retransmission- and FEC-based schemes share a common disadvantage. They both intrinsically introduce overhead in terms of the amount of data that needs to be transmitted over the communication network. Stated differently, they raise the bandwidth requirements of network traffic and hence the load on the network. Because of this drawback, the surprising scenario might occur where the addition of error protection yields an increased instead of a decreased error rate. As a result, to prevent the positive effect of being able to recover from transmission errors from being nullified, deliberate reasoning regarding the amount of protection to add to network traffic is advocated.

Given its negative impact on user experience, techniques to counter lost or damaged data seemed like a promising and meaningful extension of the NIProxy's feature list. The contributions of this chapter are hence a description of how FEC-based error protection functionality was introduced in the NIProxy and an investigation of its influence on user satisfaction. The decision to opt for a FEC solution instead of a retransmission-based scheme is motivated by the NIProxy's focus on real-time network traffic. In the course of this chapter, it will be shown that FEC support was adaptively incorporated in the NIProxy to enable dynamic control of the error protection process on a per flow basis. More importantly, it will become apparent that an integrated approach with the NIProxy's network traffic shaping mechanism was adhered to, this way guaranteeing that error protection coding reckons with contextual information (e.g., the currently prevailing channel conditions) and in addition will actually result in an optimization of the user's experience.

11.2 XOR-Based Parity Coding

A large variety of schemes to enable receiver-side correction of transmission errors exists [Moon 05]. Each has particular characteristics in terms of error correction capabilities, computational complexity, information rate (i.e., the ratio between media data and redundancy), etcetera. Out of these alternatives, a parity-based technique which relies on bitwise XOR (eXclusive OR) encoding was selected for inclusion in the NIProxy. As input, this scheme accepts a group of n media packets and produces as output a single parity packet [Li 07]. This parity packet is constructed by applying the XOR operator on the collection of bits that are stored at identical locations in the n media packets and by subsequently saving the outcome at the corresponding location in the parity packet. Each bit in the parity packet in other words represents the parity of the equivalent bits in the input packets. When needed, padding is introduced to ensure that media packets which belong to the same input group have equal length. At decoding side, the parity packet can be used to recover a singly lost or corrupted media packet which contributed to its construction. This is achieved by XOR-ing the $(n - 1)$ correctly delivered media packets with the (also perfectly received) parity packet. In case the packet is corrupted instead of completely lost, the number of introduced bit errors is irrelevant to the recovery process; as long as all errors are confined to a single packet per input group, recovery will be perfect.

The choice for a XOR-based parity approach is motivated by the fact that it exhibits a number of assets and properties that make it very suitable for incorporation in a middleware system which aims at improving the end-user experience:

Versatility The scheme exhibits maximal usability as it is a generic technique that is not bound to a particular media type. In other words, XOR-based parity coding can be applied to any network flow to protect it against transmission errors.

Adaptability The information rate and the error recovery capabilities of this technique are respectively directly and inversely proportional to the number of media packets which contribute to a single parity packet. As a result, protection can at run-time be traded for bandwidth consumption by varying the size of the input packet group. Raising this value leads to less parity packets being generated and hence a higher information rate, but at the same time also a declined error recovery performance. Decreasing the input grouping size has exactly the opposite effect. XOR-based parity coding is hence an adaptive scheme which supports easy and on-the-fly configuration of its behavior and operation, for instance based on currently prevailing channel conditions.

Scalability A certain level of scalability is guaranteed due to the scheme being lightweight in terms of computational requirements. The necessary computation is confined to the application of the (lightweight) XOR operator on a collection of bits.

Standardized Transport RFC 5109 standardizes a Real-Time Protocol (RTP) payload format for the transportation of XOR-based parity redundancy [Li 07]. Standardized solutions are preferred over proprietary ones as they promote interoperability.

Backward Compatibility In case the FEC data is encapsulated and transported according to RFC 5109, compatibility with FEC-agnostic receivers is retained. In particular, RFC 5109 defines that the FEC redundancy needs to be transmitted as a separate RTP flow; as a result, the flow which carries the to-be-protected media content itself remains unmodified. This implies that FEC-incapable hosts can simply discard the received FEC data and work with the media packets in the state in which they are delivered (i.e., possibly contaminated by transmission errors).

The main disadvantage of the parity approach is that it is a relatively bandwidth consuming technique. The scheme generates additional packets that need to be transmitted over the transportation network, which introduces significant overhead since each packet requires its own protocol headers. This effect is further aggravated in case the sizes of the to-be-protected media packets exhibit high variability, as this imposes the need for extensive padding as well as increases parity packet volume [Moon 05]. In other words, parity coding is not the most efficient FEC scheme in terms of information rate.

11.3 FEC Integration in the NIProxy

11.3.1 Stream Hierarchy Incorporation

Like any other FEC technique, parity coding introduces overhead in terms of redundant data that needs to be transmitted over the communication network. Since this FEC-generated network traffic might consume significant amounts of bandwidth, it should be reckoned with by the NIProxy's network traffic shaping mechanism. This in turn necessitates its integration in the stream hierarchy. Redundant parity information is therefore represented as a `discrete` stream hierarchy leaf node which provides a discrete bandwidth consumption level for each supported input packet grouping size. The `discrete` leaf node in addition defines an extra level which corresponds to a zero bandwidth usage and hence indicates that parity coding should be disabled. Notice that this implies that the bandwidth consumption of the discrete level that is associated with the smallest grouping size determines the redundant FEC data's maximal bandwidth requirement.

Merely representing the FEC data does not suffice however, it also needs to be adequately related to the media stream which it protects since any bandwidth that is allocated to the FEC information can no longer be consumed by the media content itself. The objective is hence to split the bandwidth that has been reserved for FEC-protected traffic among the media data and its FEC overhead in such a manner that the reception at the destination is optimized. This process is often referred to as *Joint Source-Channel Coding* (JSCC) and its impact on user QoE is apparent. The experimental results that will be presented in section 11.4 were generated by relying on the `Percentage` node type to link a media flow with its FEC protection in the stream hierarchy. This approach allows straightforward control over the JSCC process via adjustment of the assigned percentage values. It should be noted however that this solution might prove to be too coarse and rudimentary to be practically usable in realistic environments. Investigating the integration of more fine-grained and advanced JSCC support in the NIProxy is an important topic of future work.

Finally, as the FEC redundancy is rendered useless in case the media data which it applies to is not received by the destination, a sibling dependency of type `SD_BW_ALLOC_CONSTRAINED` is defined between the stream hierarchy nodes which represent the media and its FEC protection in the stream hierarchy. As was described in section 4.5, doing so installs the condition that FEC traffic can consume bandwidth if and only if its associated media flow is currently enabled.

11.3.2 Implementation

FEC support was not incorporated in the NIProxy as an integral part of its general software architecture but instead as a NIProxy service. As section 5.1 has explained, the advantage of this design decision is that FEC-related functionality will only be loaded in case it is effectively needed and that any issues which it might introduce will be bounded by the service and will hence not contaminate the NIProxy's basic operation.

The service accepts as input original media data and supplements it with redundant parity information. Its exact mode of operation is as follows. First of all, the service registers interest for the class of network streams which carry data that is eligible for FEC protection[1]. Next, on the discovery of each such network stream, the service performs two initialization tasks. The first task consists of instantiating a FEC encoder for the newly detected flow, in this case a XOR-based parity coder. Secondly, as was discussed in section 11.3.1, the service notifies the bandwidth brokering process of the possibility to FEC protect the media stream as well as the thereby associated bandwidth requirements. On completion of this initialization phase, the network flow is finally added to the service's main processing loop. Each iteration of this loop starts with the service exploiting its interface with the network traffic shaping

[1]Recall from the discussion in section 6.2.2 that a NIProxy service is able to control which data it is handed over by registering interest for certain network stream types or even individual network flows; only data transported on these streams will be provided to the service as input.

mechanism to determine the discrete level to which the FEC data for the media flow that is being processed is currently set. The corresponding FEC encoder is subsequently switched to the input grouping size that is associated with this level, after which it is fed with the input media packet. In case this packet completed a protection group, the media packet is output together with the resulting parity packet. If not, only the media packet is returned as output of the service. As an exception, in case the currently enabled discrete level of the FEC stream's representation in the stream hierarchy corresponds with a zero bandwidth consumption, FEC processing is bypassed by simply not handing over the input packet to the parity encoder.

11.3.3 Supporting Additional FEC Techniques

It is important to note that the NIProxy's FEC support does not need to be limited to parity-based coding. Other FEC schemes such as the popular Reed-Solomon (RS) code [Moon 05] could just as well be incorporated. Analogous to the way parity coding was integrated, this would at least involve

- the introduction of a new FEC encoder in the NIProxy
- adequately representing the network traffic that is generated by the FEC encoder in the stream hierarchy
- relating the FEC data to the corresponding media content in the stream hierarchy

Being able to fall back on a catalog of FEC schemes would likely improve the NIProxy's performance. The effectiveness of individual FEC approaches namely tends to vary considerably depending on the features of the media that needs to be protected as well as environmental factors like the current characteristics of the data corruption process. In case multiple FEC codes would be supported, the NIProxy could exploit its contextual awareness to estimate the beneficial impact of each code on user QoE under the prevailing conditions. Based on these estimations, the optimal FEC technique could subsequently be selected. It would even be possible to enable collaboration between different FEC techniques through concatenation (i.e., applying FEC coding on an already protected media flow [Moon 05]) in case doing so would benefit the end-user experience.

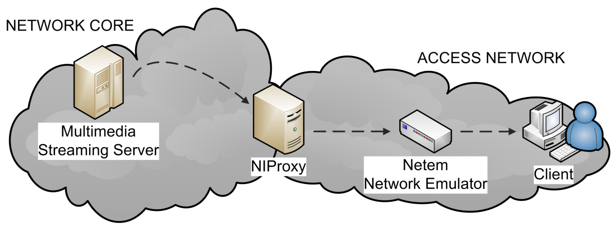


Figure 11.1: Network setup for the video streaming case study.

11.4 Evaluation

This section harbors experimental results which will comprehensively demonstrate the added value of the presence of FEC-based error protection functionality in the NIProxy. In particular, its advantageous influence on the NIProxy's QoE optimization capabilities will be evaluated by analyzing the outcome of a practical experiment that was conducted multiple times, under varying circumstances. The experiment, which will be described in detail in section 11.4.1, was deliberately kept simple to channelize the reader's attention toward the specific contributions of this chapter and to enable intelligible distillation of their impact from the produced results. Furthermore, despite the experiment being minimalist, the presented results are representative since they can be extrapolated to realistic contexts in a straightforward manner.

11.4.1 Experiment Description and Setup

The experiment simulated a video streaming scenario between a multimedia server and a receiving client. As is depicted in Figure 11.1, the server was conceptually deployed inside a high-capacity and relatively error-free network backbone. The client on the other hand was located at the periphery of the network and was connected to the backbone through a resource-constrained access link. In between these two end-hosts, a NIProxy component was interposed which was responsible for engineering the network traffic that was destined for the client. The objective was hence to determine whether this client benefited from the inclusion of the NIProxy in the experiment. The NIProxy instance was conceptually deployed on the boundary between the backbone and the access network. This location was chosen because it represents a crucial junction point in the simulated network topology where data originating from the multimedia server needed to transfer from the resource-abundant network core to the much less capacitated and possibly error-prone access network. Recall from section 3.5 that such transitional network locations are considered to be ideal positions for NIProxy inclusion. Finally, to be able to emulate packet loss on the client's access link, the experimental setup also included an instance of the open-source netem network emulator [netem 10][Hemminger 05]. The netem framework is included in recent distributions of the GNU/Linux operating system (i.e., distributions with version number 2.6 and higher) and enables the reproduction of Wide Area Network (WAN) dynamics by delaying, dropping, duplicating, re-ordering or corrupting network packets.

Within this experimental setup, the multimedia server maintained two simultaneous RTP video sessions with the receiving client, which will be denoted by VS1 and VS2 in the following discussion. The streaming server emitted video data in unprotected form (i.e., without FEC information) since the network core was assumed to be nearly error-free. The NIProxy instance however had its FEC service loaded and was hence able to add XOR-based parity protection to transiting network traffic. Parity coding could be performed per 3 or per 6 input packets. Recall that resorting to a smaller input grouping size increases the error recovery possibilities at the expense of elevating the amount of bandwidth that is required for transporting the parity information. To allow for the unambiguous demonstration of the effects of the FEC processing as well as the bandwidth requirements which it imposed, in the experiment only video session VS2 was marked as being eligible for receiving FEC protection. For the comparison with unprotected flow VS1 to be meaningful, an identical video fragment was streamed in both sessions. Besides the FEC service, the NIProxy made use of the (inbound) static video transcoding service which was presented in section 5.3. The NIProxy was consequently able to address downstream bandwidth shortage on the last mile by reducing the bitrate of incoming video streams. The quality parameters of the employed video fragment as well as the output settings for the video transcoding service are listed in Table 11.1.

Figure 11.2 depicts the stream hierarchy which steered the transmission of the two video flows over the receiving client's access connection. This stream hierarchy in other words specified how the downstream bandwidth capacity that was available on the last mile needed to be distributed over the generated video traffic. The root node was of type `Percentage` and had as children two subtrees which each represented one of the video sessions that existed between the streaming server and the client. Both subtrees were assigned a percentage value of 0.5 to specify that the bandwidth capacity of the access link should at all times be split perfectly fairly over both video connections. This was again enforced to enable meaningful comparison of the way both video streams were treated by the NIProxy as well as of their reception at

Table 11.1: Video encoding parameters.

	Original	**Transcoded**
Resolution (pixels)	320 × 240	176 × 144 (QCIF)
Framerate (FPS)	20	15
Bitrate (bps)	100000	60000
Codec	H.263+	H.263+

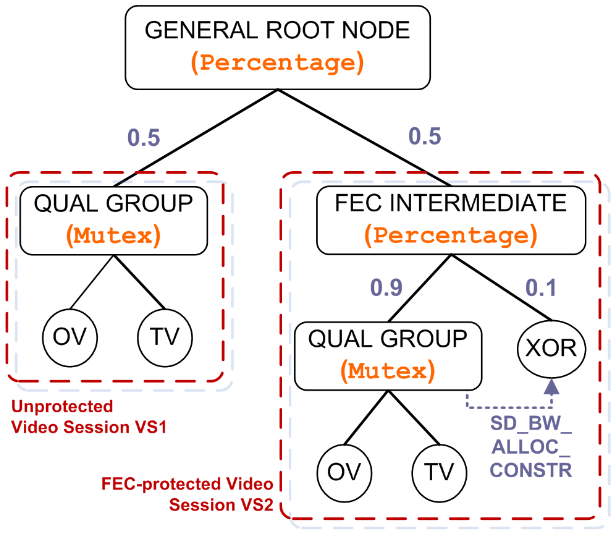


Figure 11.2: Stream hierarchy which directed the shaping of the network traffic that was destined for the client in the experiment.

client-side. The leftmost subtree corresponded with video session VS1 which was not qualified for receiving FEC protection from the NIProxy. This subtree comprised two `discrete` leaf nodes which respectively represented the Original and Transcoded Version (OV and TV) of the transported video fragment and which were differentiated from each other using an internal node of type `Mutex` (see section 5.3.1). Both leaf nodes supported two discrete bandwidth consumption levels which represented the extremes of respectively obstructing their associated network stream and forwarding it at its maximal throughput. An analogous construction can be found in the rightmost subtree, which however also included a number of additional nodes since it corresponded with video session VS2 which was potentially subjected to FEC processing by the NIProxy. The XOR-based parity FEC data was incorporated as a `discrete` leaf node which provided three discrete bandwidth levels, one for each supported input grouping size plus an additional level that was associated with a zero bandwidth consumption. As was discussed in section 11.3.1, JSCC was implemented by making the FEC representation a sibling of the quality grouping `Mutex` node using a node of type `Percentage` as parent. In this experiment, a static JSCC approach was employed where 90 percent of the available bandwidth was always assigned to the media stream itself, while the remaining 10 percent was reserved for its parity data. Also observe the sibling dependency of type `SD_BW_ALLOC_CONSTRAINED` that was defined between the quality grouping `Mutex` and the FEC leaf node.

The just described stream hierarchy was constructed entirely by the client that was managed by the NIProxy, expect for

- the TV nodes; as section 5.3.1 has explained, these nodes were added to the stream hierarchy automatically by the NIProxy's static video transcoding service

- the FEC Intermediate and XOR nodes; these were introduced in the stream hierarchy by the FEC service (see section 11.3.1)

- the `SD_BW_ALLOC_CONSTRAINED` sibling dependency; analogous to the integration of the FEC Intermediate and XOR nodes, the FEC service was responsible for the definition of this dependency between the stream hierarchy representiions of the multimedia content and its FEC protection

The described experiment was executed twice, once without and once with the netem component introducing packet loss in the access network. During each experiment execution, all conditions remained constant, with the exception of the downstream bandwidth capacity of the access link. Artificially modifying the last mile throughput at predefined points in time enabled the investigation of the effects of these bandwidth fluctuations on the way the NIProxy shaped the network traffic that was destined for the managed client. Five such bandwidth modifications were performed in the course of the experiment, which caused the experiment to be conceptually segmented into six distinct intervals.

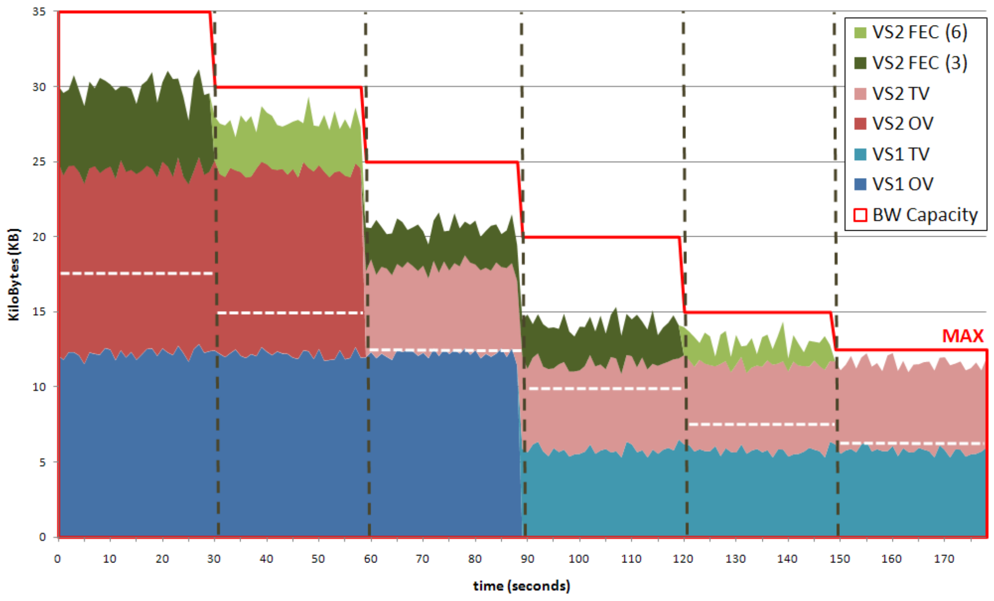


Figure 11.3: Stacked graph illustrating the network traffic received by the NIProxy-managed client during the error-free execution of the experiment.

11.4.2 Experimental Results

No Packet Loss

The experiment was first executed in an error-free environment to enable complete and perfect tracing of the way the video traffic transited the access network and arrived at the destined client. The outcome is plotted in Figure 11.3. In this network chart, the solid red line specifies the downstream bandwidth capacity of the access connection, whereas the dashed vertical lines separate the consecutive experiment intervals. As can been seen, the interval transitions occurred at interspaces of approximately 30 seconds and always coincided with an (artificial) reduction in last mile throughput. Finally, the dashed horizontal lines indicate the bandwidth percentages that were reserved for both video connections during the different experiment intervals. As each connection was assigned an equal percentage value in the stream hierarchy, the downstream bandwidth capacity of the access link was conceptually cut in halve and split perfectly equitably over both. Notice however that in the case of video session VS2, the allocated bit budget needed to be distributed over the video data itself and its FEC protection.

In every experiment interval apart from the first, the NIProxy was forced to apply network traffic engineering due to access bandwidth being insufficiently available to inject all involved network streams at their maximal rate into the last mile. In the third period, for example, the client received VS1 at full quality and the parity coding for VS2 also ran at maximal bandwidth consumption, but VS2 itself however was transcoded to a lower fidelity by the NIProxy before it was relayed to the access network. As the experiment progressed, downstream access bandwidth became gradually more constricted; consequently, increasing bandwidth reductions needed to be enforced for the involved network streams, up to the point where FEC coding was even completely disabled in the last experiment interval.

Also observe from the network trace that diminishing the bandwidth consumption of the media data yielded an equivalent reduction in the bandwidth requirements of its FEC protection. This is explained by the fact that as data packets decrease in size, so do the packets carrying parity information.

Finally, the network graph reveals that, perhaps somewhat unexpected, the available access bandwidth was not always fully exploited. This is most pronounced in experiment interval 4, where approximately a quarter of the downstream bandwidth capacity remained unallocated. This behavior was nonetheless justified since in these situations the NIProxy could not switch any of the involved network flows to a higher bandwidth consumption level, either because doing so would violate the current bandwidth constraints or because they were already operating at their maximal rate.

10 Percent Random Packet Loss

In the second iteration of the experiment, the netem component which was included in the experimental setup emulated 10 percent packet loss on the access connection. The netem entity was configured to discard packets randomly (i.e., the packet dropping process did not employ a correlation factor and hence no particular attempts were made to simulate burst errors). As can be derived from Table 11.2, the outcome was the loss of video data as well as FEC protection packets. On the other hand, it also resulted in the FEC data being put to meaningful use (i.e., to reconstruct lost packets at receiver-side). Table 11.2 therefore also includes packet recovery statistics for video stream VS2. It is shown that the redundant parity information enabled the receiving client to recreate 56.19 percent of the packets that were lost on video session VS2, this way yielding a residual loss of 92 packets instead of the original 210.

Besides collecting packet loss and recovery statistics, data reception at client-side was again also recorded. Unsurprisingly, the results perfectly re-

Table 11.2: Packet loss and recovery statistics (10 percent packet loss randomly introduced on access link).

	Experiment Interval						**Total**
	1	**2**	**3**	**4**	**5**	**6**	
# **Lost**							
VS1	47	45	52	32	21	27	224
VS2	56	31	38	34	27	24	210
FEC	15	6	11	11	3	0	46
% **Lost**							
VS1	11.03	10.23	11.79	9.07	7.09	9.64	10.02
VS2	12.33	7.11	10.83	11.53	9.12	8.39	9.92
FEC	10.27	6.38	12.09	11.22	4.69	0	9.33
# **Recovered**	40	18	20	22	18	0	118
% **Recovered**	71.43	58.06	52.63	64.71	66.67	0	56.19

sembled those from Figure 11.3, except for the bandwidth consumption of the monitored streams this time displaying occasional irregularities (i.e., drops) which were caused by the loss of packets. As the network trace does not convey extra information compared to 11.3 nor provides any additional insight, it is omitted from this discussion.

Executing the experiment in an error-free environment yielded perfect video playback at the destination. This was unfortunately no longer the case in this iteration of the experiment. Since the access connection now suffered from packet loss, the destination did not always have all video packets at its disposal. As a result, decoding issues arose which, as is illustrated in Figure 11.4[2], in turn caused (sometimes severe) perceptual artifacts to be introduced in the decoded video. The availability of FEC information for VS2 however

[2]The video fragment shows two moving balls which are respectively blue- and red-colored. Due to the simplicity of the scene, visual inconsistencies are immediately and clearly noticeable.

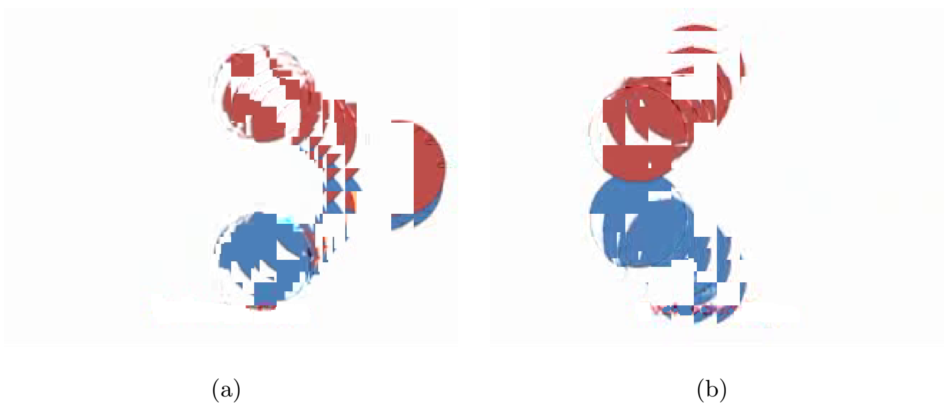


Figure 11.4: Two examples of imperfect video playback at client-side caused by the loss of video packets.

enabled the receiver to repair a substantial fraction of the packets that went missing on this session. Compared to video stream VS1, the playback of VS2 was hence significantly less distorted since decoding artifacts were much less pronounced.

11.4.3 Discussion

A number of significant findings can be deduced from the network chart that is presented in Figure 11.3. First of all, it proves that the NIProxy shaped the network traffic which was destined for the client in such a manner that the downstream capacity of the client's access connection was at all times respected. This is an important achievement since it substantially contributed to congestion avoidance in the last mile. Stated differently, the NIProxy delivered a direct contribution to the access network operating in a stable and predictable manner (which was for instance exemplified by a low variation in packet transmission latency). A second observation is that the bandwidth distribution solution which was delineated for this experiment was successfully put into effect by the NIProxy, since the available access bandwidth was shared equitably among the involved video sessions. Either network connection was only allowed to raise its bandwidth consumption beyond its "fair share" in case spare bandwidth that was originally reserved for the other connection was available. This is a direct consequence of the two-phase bandwidth distribution approach that is implemented by the `Percentage` node type (see section 4.2.3) and it resulted in a more complete exploitation of the access bandwidth capacity. Notice that this observation is not limited to the way bandwidth was shared among the video connections themselves, but that it also applied to the JSCC process for the FEC-protected video stream. This is exemplified in experiment intervals 3 and 4, where the FEC data stream was initially entitled to a bandwidth percentage that barely sufficed to perform parity coding per 6 input packets. As the video data which it protected however did not fully consume its reserved bit budget, the FEC stream was able to claim this excess bandwidth and exploited it to switch to a grouping size of 3 packets. Third, the presented experimental results are an interesting illustration of the potential of supporting interoperation between NIProxy services and its bandwidth brokering operations. In particular, the network trace demonstrates that the JSCC process (i.e., the amount of client bandwidth that was spent on FEC data) was directed entirely by the NIProxy's network traffic shaping mechanism. As a result, the processing that was implemented by the FEC service was at all times in tune with the contrived bandwidth brokering strategy. Finally, notice that JSCC may result in the need to reduce the quality of the multimedia data to allow for its FEC protection flow to be accommodated. In the described experiment, this for instance occurred in the third interval, where sufficient bandwidth was available to forward video stream VS2 at its maximal rate. However, as its associated FEC data was entitled to a fraction of this bandwidth volume, VS2 itself needed to be transcoded to a lower bitrate. In contrast, since video stream VS1 lacked FEC protection, it was possible for the client to receive VS1 in original quality during this experiment interval. Remark that all discussed findings also apply to the second iteration of the experiment, since the NIProxy enforced an identical traffic engineering policy in both experiment executions.

The discussion thus far has uncovered that FEC-protecting a multimedia flow does not come for free as it imposes a non-negligible bandwidth overhead. The bandwidth that is consumed by the redundant information however enables missing or corrupted data to be repaired at the destination. The beneficial impact hereof cannot be inferred from the presented network trace but instead is comprehensively highlighted by the loss and recovery statistics that are provided in Table 11.2. In the third experiment interval, for instance, recall that it was necessary to transcode video stream VS2 to a lower bitrate so that its FEC protection could be accommodated. Table 11.2 however demonstrates that the availability of the FEC data in this interval enabled the receiving client to reconstruct 52.63 percent of VS2's packets which were corrupted during their passage through the noisy access network. In contrast, in the same experiment interval the client received video stream VS1 at original quality, but since this stream lacked FEC protection, none of its 52 lost packets could be recovered. When comparing the playback of VS1 and VS2 on the end-user device, the latter was much smoother and less perceptually degraded. It is expected that a lower-quality yet only mildly distorted version of a video fragment will generally yield a more enjoyable (or, conversely, less frustrating) viewing experience than a high-quality video fragment which exhibits grave visual deformations and/or temporal interruptions that are caused by unaccounted for transmission errors. This is however a highly subjective matter about which individual users may hold differing opinions. It is therefore important to note that, for users which actually prefer distorted high-quality video to a less distorted lower-quality variant, it is possible to prevent their destined video traffic from receiving FEC protection from the NIProxy. In fact, FEC support was incorporated in the NIProxy in a sufficiently flexible manner so that such decisions can be enforced at an even finer granularity, namely on a per flow basis. Stated differently, the NIProxy allows for its FEC behavior to be controlled and configured per individual network flow.

A concluding remark concerns the versatility of the NIProxy's FEC support. In particular, by opting for a media-independent parity coding approach, it is possible for the NIProxy to apply its FEC functionality to arbitrary types of network traffic. Although the experimental evaluation has focused exclusively on video content, the NIProxy's FEC support can consequently just as well be exerted to protect network flows which transport other types of content against transmission errors.

11.5 Related Work

This chapter has specifically addressed the incorporation of FEC support in the NIProxy. The books by Moon [Moon 05] and Lin and Costello [Lin 04b] are both excellent reference works on error correction coding.

An important consequence of FEC-based solutions is the need to perform Joint Source-Channel Coding (JSCC). Examples of proposed JSCC strategies abound in the literature. For instance, Bolot et al. have presented a joint rate/error control algorithm for Voice over IP (VoIP) applications which aims to strike an optimal balance between original and redundant data such that the subjective audio quality as perceived at the destination is maximized [Bolot 99]. To FEC protect the voice data, the authors rely on the media-dependent Redundant Audio Coding (RAC) technique [Moon 05]. Their solution is hence not applicable to media data other than audio. As another example, Filho et al. have described an adaptive and media-independent FEC scheme which is based on interleaved XOR-based parity coding [Filho 06]. In an interleaved approach, multiple to-be-protected media flows and their FEC redundancy are interwoven to obtain additional error resilience. Their scheme is steered by a hierarchical model which attempts to predict the packet loss evolution. In particular, based on estimated future packet loss characteristics, the interleaving parameters of the FEC scheme are configured to ensure that the residual packet loss (i.e., the loss after reconstruction at receiver-side) is minimized. A final example is the algorithm that is proposed by Frossard et al. for computing the rate distribution between MPEG-2 video data and its FEC protection [Frossard 01]. The scheme is based on media-independent XOR parity coding and attempts to minimize the end-to-end perceptual video distortion by adapting its mode of operation to 3 factors: bandwidth availability, the complexity of the video scene (which affects source coding) and network state and performance (e.g., the currently prevailing error conditions).

The just described JSCC approaches apparently outperform the NIProxy's FEC support in terms of efficiency as well as sophistication. In this regard, it is important to note that the work that was presented in this chapter should be considered as merely an initial attempt at FEC integration in the NIProxy. This initial attempt is however built on top of a solid foundation, which implies that the NIProxy's FEC features and performance can readily be extended and improved in the future. As an example, any JSCC solution could theoretically be incorporated in the NIProxy by encapsulating its behavior and mode of operation in a new type of internal stream hierarchy node. In addition, a distinctive feature of the NIProxy and hence also its FEC functionality is the ability to exploit application-related knowledge, a type of context that is mostly overlooked in FEC-based systems. Finally, keep in mind that, contrary to the majority of the FEC solutions which are described in the literature, the NIProxy is a general-purpose user QoE optimization framework of which FEC coding merely forms one small facet. It is consequently not surprising that the NIProxy fails to match the performance and functionality of systems that are specialized in FEC processing.

11.6 Conclusion and Future Work

Multimedia data which is destined for clients of distributed applications might arrive in corrupted form or could even be partially lost during its propagation through error-prone transportation networks. The typical outcome is a deteriorated media presentation at receiver-side, which should be avoided as much as possible since it is a very likely source of user frustration. Forward Error Correction (FEC) schemes attempt to remedy this issue by adding redundancy to the transported data to enable receivers to repair compromised or lost information. Given its potential to negate or at least alleviate the detrimental effects of data corruption, it has been decided to introduce FEC coding functionality in the NIProxy. This chapter has shown that FEC support was incorporated in the form of an adaptive XOR-based parity coder whose operation is directed by the NIProxy's network traffic shaping mechanism. The objective of this integrated approach is to guarantee that the bandwidth overhead that is introduced by the FEC processing is justifiable and is adequately weighed against not only the multimedia stream which it protects, but also any other network traffic that is being exchanged as part of the distributed application. The FEC inclusion has been practically evaluated using a video streaming use case. The experimental results have comprehensively corroborated that adaptive FEC support was successfully incorporated in the NIProxy. Additionally, they have demonstrated that the FEC functionality represents a valuable addition to the NIProxy's feature list to improve the experience of users of distributed applications.

The incorporation of FEC coding in the NIProxy is still in its infancy. As a result, although the presented experimental results are already promising and encouraging, they are only tentative as it is apparent that there is still room for considerable improvement. Possible future research directions include extending the NIProxy's FEC support with techniques other than XOR-based parity protection (e.g., RS coding) and the introduction of more powerful and effective JSCC algorithms. Regarding this JSCC topic, a first important improvement would be to adapt the division of bandwidth among the media data and its FEC protection to the currently prevailing error characteristics. Secondly, it might turn out beneficial to design a new type of internal stream hierarchy node to direct the JSCC instead of relying on an existing type, as none of these might be capable of efficiently modeling the JSCC process. Finally, besides performing implementational adjustments, it might also be interesting to organize user studies to obtain qualitative feedback regarding the effects of the FEC processing that is performed by the NIProxy on the user experience.

Chapter 12

End-to-End QoE Optimization Through Overlay Routing Interoperation

As was already noted in section 3.6, the range of conceivable QoE optimization operations is so extensive that it is unrealistic to assume that any single framework will be able to address them all. Most QoE optimization components therefore specialize in a limited number of techniques for user experience manipulation. Some proposed systems are even entirely devoted to one particular facet of QoE improvement. The NIProxy is no exception to this rule since it primarily focuses on network traffic shaping and multimedia service provision. Although the latter mechanism ensures a certain degree of extensibility in terms of supported functionality, as a stand-alone entity the NIProxy will never be able to solve all possible QoE issues (or at least not as optimal or efficient as specialized systems).

A potential solution to this problem is collaboration between individual frameworks. By combining multiple QoE management components, it becomes possible to tackle a larger set of QoE issues. Moreover, an integrated solution might unlock additional QoE enhancement options that are not achievable when applying the constituting components separately (i.e., the whole is often greater than the sum of its parts). This chapter will discuss such an integrated platform in which the NIProxy is combined with a resilient overlay routing service to enable (near) end-to-end QoE optimization [De Vleeschauwer 08b][Wijnants 10]. The two-layer architecture exploits its routing functionality to enhance data dissemination in the network backbone by improving the backbone's resilience to issues like failing or congested links. The incorporated NIProxy instances on the other hand are deployed close to end-users, where their provided traffic engineering tools are exploited to control and direct the delivery of data over the last mile of the network connection. Through the presentation of experimental results, it will be demonstrated that the proposed architecture covers nearly the entire end-to-end data exchange path and succeeds in maintaining a high QoE despite routing issues in the network core and/or resource restrictions in the access network.

The overlay routing functionality that is included in the proposed platform has been developed at the INTEC Broadband Communication Networks (IBCN) research group of Ghent University. The work that will be described in this chapter hence represents a joint research effort on the topic of QoE optimization that combines expertise from Hasselt University and Ghent University.

12.1 Introduction

In recent years, a popularization of the networked access of multimedia applications has occurred. Compared to traditional services like file transfer and e-mail, these distributed applications impose much stricter requirements on the telecommunications network in terms of packet loss, delay variation, throughput guarantees, and so on. For instance, interactive applications such as Voice over IP (VoIP) and online gaming demand a low delay to guarantee a fluid operation. As another example, data corruption and packet loss negatively impact video conferencing and IPTV services since it will rapidly degrade playback at receiver-side due to the introduction of perceptual distortions. Complicating matters even further is the fact that, due to the recent trend towards mobile computing, service providers are increasingly targeting not only fixed but also mobile customers. Since the capabilities of fixed terminals and wired networks diverge significantly from those of their mobile and wireless counterparts, a highly heterogeneous usage environment is yielded. This diversity in turn results in growing dependability as well as adaptation requirements for distributed applications.

Empirical evidence has proven that current generation networks, and the Internet in particular, are not always capable of guaranteeing that the requirements which are imposed by multimedia services are satisfied. Failure to meet these requirements is likely to give rise to disruptions in application quality and can have several causes; some examples are enumerated below:

Suboptimal Routing The Internet provides only best-effort routing, which implies that no guarantees are given regarding the level of service that will be experienced by network packets. In particular, the standard routing protocols of the Internet focus on producing paths with a minimal number of intermediate hops, hereby disregarding the delay, throughput and packet loss characteristics of the individual links which constitute the constructed route. As a result, routing in the Internet might fail to find the optimal path between two end-points (i.e., the path with the highest performance and quality parameters).

AS Graph Issues The Internet corresponds to a composition of several *Autonomous Systems* (ASs). Each AS represents a stand-alone administrative domain in which every router adheres to a single, clearly defined routing policy. To achieve inter-AS routing, the Internet maintains an AS graph. Empirical evidence indicates that whenever a problem occurs in this graph, the routers may take considerable time to find a workaround, resulting in periods of connectivity loss that can even be in the order of minutes [Feamster 03].

Constrained Last Mile Bandwidth The access part of a client's network connection is another possible source of complications, as it likely forms the bandwidth bottleneck in an end-to-end path between communicating hosts. In particular, compared to the typically over-provisioned network core, the last mile connection is much less capacitated. As a result, insufficient (downstream) access bandwidth may be available to support all the client's active services (or even to receive all content that is being exchanged as part of a single multimedia service). An overwhelmed access connection will generally give rise to congestion and hence also an increase in packet loss and delay in case adequate techniques for the adaptation and engineering of network traffic are lacking.

These observations argue that current networks frequently fail to provide customers of distributed multimedia applications with an acceptable usage experience and that complications might be introduced anywhere along the network path between the communicating end-points. This chapter will therefore describe a two-tier platform which integrates the NIProxy with a resilient overlay routing service to achieve near end-to-end user QoE optimization. As such, this chapter will implicitly prove the NIProxy's ability to interface and collaborate with other QoE optimization systems.

12.2 Proposed Two-Tier Platform

This section will present the proposed two-tier QoE optimization architecture. First, an overall view of the platform will be provided and its constituting element types will be briefly enumerated. The next two subsections are devoted to a profound discussion of the responsibilities and modus operandi of both tiers. Finally, remarks regarding the degree of coverage of the end-to-end IP path between communicating hosts will be presented.

It is again repeated that the network architecture has been implemented in collaboration with Ghent University. In particular, the tier-1 functionality and component types stem from research on overlay routing that has been conducted at Ghent University.

12.2.1 Overview

The QoE enhancement platform attends to the separation of concerns paradigm and as such comprises three distinct types of components. The encompassed component types each have well-delineated responsibilities and are deployed at strategic locations in the Internet topology to enable (near) end-to-end QoE optimization:

Overlay Servers (OSs) Implement an overlay network on top of the network backbone. They act as overlay-layer routers in the proposed QoE architecture and as such provide a resilient and reliable packet routing infrastructure. In particular, the Overlay Servers enable the QoE platform to evade issues in the network core by routing packets around problematic backbone links.

Overlay Access Components (OACs) Regulate the (transparent) access to the overlay routing infrastructure that is provided by the Overlay Servers. In particular, they are responsible for determining, giving current environmental conditions, whether network packets would benefit from overlay routing. Contrary to the Overlay Servers, Overlay Access Components are deployed in close proximity to the application endpoints, possibly even on the end-user device.

NIProxy instances These are introduced at the periphery of the network, where their traffic engineering functionality is exploited to execute QoE improving operations before the network traffic reaches the last mile of the network connection.

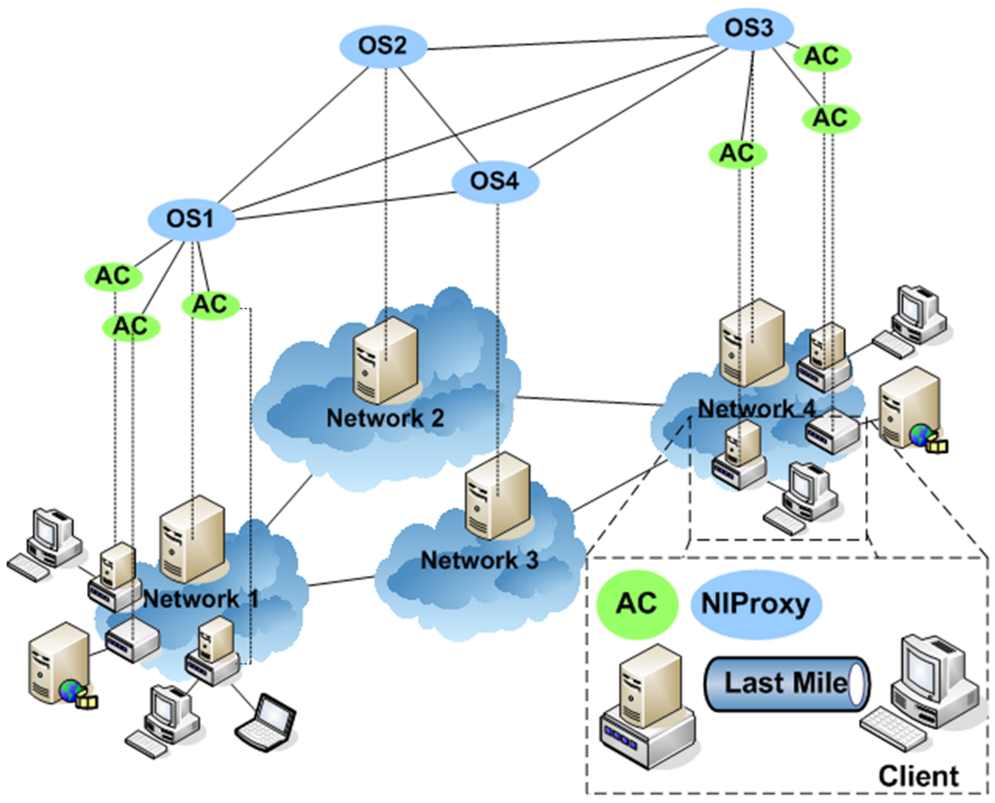


Figure 12.1: Architectural overview of the integrated QoE optimization platform.

Combined, the first two component types implement a robust overlay routing service in the network core and consequently constitute the first tier of the proposed QoE optimization architecture. The essential function of this layer is to offer a alternative route in case the direct IP connection is experiencing inferior performance. The second tier of the distributed architecture is formed by the NIProxy instances, which are responsible for addressing possible QoE optimization requirements that are imposed by the access part of the downstream content delivery path.

A schematic summary of the proposed end-to-end platform is depicted in Figure 12.1. For reasons of compactness, Overlay Access Components are denoted by AC instead of OAC. This approach will be adopted in all subsequent illustrations in this chapter.

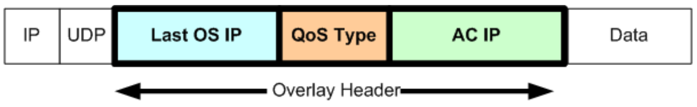


Figure 12.2: Overlay header format [De Vleeschauwer 08b].

12.2.2 Tier-1 Functionality and Constituting Component Types

Overlay Server

The Overlay Server (OS) components are incorporated in several Autonomous Systems in the IP backbone and organize into an overlay network [De Vleeschauwer 10, De Vleeschauwer 08a]. They maintain *overlay routing tables* which map target OS IP addresses to the next hop OS IP address. As is illustrated in Figure 12.2, an overlay packet encompasses a dedicated overlay header, which is introduced between the UDP header and the packet payload. When such a packet is received by an Overlay Server, it extracts the target OS address from the header and consults its routing table to determine the next overlay hop IP address, after which the packet is relayed to this next hop. In addition to information on the target Overlay Server, the overlay header contains a field which specifies the IP address of the Overlay Access Component that the packet needs to be delivered to as well as a field for the type of QoS which the packet expects. The latter allows for the specification of the treatment that a specific packet should receive. For instance, some services may be very delay-sensitive (e.g., VoIP), while others are highly intolerant to packet loss (e.g., streaming video). When the final Overlay Server receives the packet, it will forward it to the targeted Overlay Access Component, which in turn will make sure that the packet will be delivered to the destined client (as will be clarified later on).

The Overlay Servers maintain an overlay topology which contains information regarding the connectivity between pairs of Overlay Servers. OS interconnection quality is analyzed and estimated through active network probing. In particular, the OSs periodically exchange ICMP echo messages with their neighbors in the overlay topology. Based on the outcome of this probing, the quality of the overlay edges is determined and values for delay and packet loss are deduced. This information is disseminated among Overlay Servers to make sure that they all share an identical view on the connectivity in the entire overlay network. The accumulated topological knowledge is subsequently leveraged by the OSs to calculate their overlay routing tables. When a problem is detected in the topology, the overlay routing tables will be updated correspondingly, which in turn will cause the OSs to introduce one or more in-

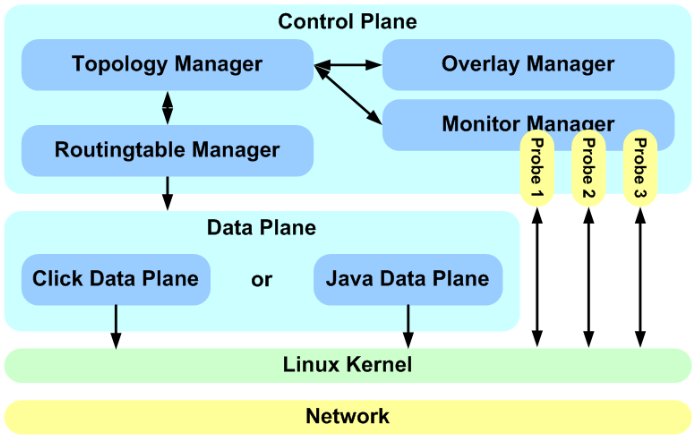


Figure 12.3: Overlay Server software architecture [De Vleeschauwer 08a].

termediate overlay hops so that packets will be relayed around the problematic parts of the network.

As can be deduced from Figure 12.3, the software architecture of the overlay routing server encompasses two separate yet interacting levels [De Vleeschauwer 08a]. The control plane maintains the overlay topology and overlay routing tables. In addition, it coordinates the monitoring of the overlay edges and performs overlay management tasks (e.g., undertaking the necessary actions when new Overlay Servers are added to the network). The data plane on the other hand is responsible for actual packet handling and for forwarding packets to the next hop. By decoupling the control plane from the data plane, it becomes possible to adjust the operation of one plane without impacting the other. The implementation of the data plane is based on Java UDP sockets and draws from the Click modular router software [Kohler 00].

Overlay Access Component

The Overlay Access Component (OAC) entities monitor the direct end-to-end IP path between pairs of communicating hosts and enable applications to access the overlay routing service that is provided by the Overlay Servers [De Vleeschauwer 10][De Vleeschauwer 08a]. Each end-point of a connection is associated with a single OAC. The OAC needs to be deployed on or close to the actual end-device. For instance, an OAC could be installed on a residential gateway or could take advantage of the recent evolution of the access node towards an intelligent platform with the ability to host a variety of services [Gilon - de Lumley 07]. The OAC detects when a new connection becomes active and is responsible for deciding whether packets belonging to this connection should follow the direct IP route or instead should be pushed to an OS for overlay dissemination. The overlay route will only be exploited in case the direct path exhibits inadequate QoS parameters. This policy is enforced to prevent needless encumbrance of overlay resources.

QoS issues on the direct IP path are again detected through active network probing. However, the connection quality could also be inferred through passive monitoring and by exploiting knowledge about the transport protocol. For instance, by sniffing packets of TCP- or RTP/RTCP-based services on an intermediate network node, it might be possible to deduce connection quality parameters [De Vleeschauwer 07b][Simoens 07]. As soon as a QoS problem such as packet loss, high delay or even disconnection is observed on the IP route, the OAC will start inserting an overlay header in the packets which belong to the affected connection and will push the modified packets to a nearby Overlay Server. This Overlay Server will subsequently deliver the packets to the OAC which is located close to the destination of the connection via an overlay route with higher QoS properties (i.e., by exploiting a number of intermediate overlay hops). As a final step, the target OAC will remove the overlay header from the received packets and will subsequently transmit them to their actual destination. In this way, the connectivity between source and sink is preserved, while access to the overlay network is abstracted and remains transparent to the benefiting application. The availability of the OACs in other words eliminates the necessity to modify the application software and hence makes it possible for legacy applications to also profit from the overlay routing service.

The different tasks that are involved in the operation of the OAC element can be enumerated as follows:

- Each OAC maintains a connection to one or possibly multiple Overlay Server instances, preferably ones which are situated nearby in the network topology. The OAC will forward all network packets which it intercepts from a source and which should receive overlay routing treatment to one of these OSs.

- As soon as the OAC detects that a monitored source starts communicating with another IP host, it will establish whether the remote party is also subscribed to the overlay routing service. If this is the case, the OAC will instantiate a probing thread that will periodically examine the QoS performance of the direct IP route to the OAC at destination-side. Based on the probing outcomes, the OAC is able to analyze the connectivity between source and destination and to maintain values for the delay and the packet loss that is occurring on the end-to-end IP path.

- In case the QoS parameters of a connection drop below a configurable threshold, the OAC will conclude that the connection could benefit from overlay treatment. As a result, any subsequent packets which belong to this connection will be encapsulated in an overlay packet and will be pushed to the overlay network, which will deliver it to the OAC that is associated with the destined host. The overlay header will be filled in so that its last OS field specifies the IP address of the Overlay Server to which the destination OAC is connected, while the field for the access component will be completed with the IP address of this remote OAC itself.

- On the reception of an overlay packet, an OAC will first perform decapsulation by removing the overlay header and will then forward the packet further to its final destination, as if no overlay handling had occurred.

- When the connection between a particular pair of hosts becomes inactive, this will be discovered via a time-out. The OAC will respond to such an event by suspending the QoS probing thread for the terminated connection and by purging the route from its connection table.

The software architecture of the Overlay Access Component is depicted in Figure 12.4. The (prototype) OAC implementation runs on the GNU/Linux operating system and makes use of the netfilter framework [Netfilter 10] to intercept packets and to reinsert them in the network. It also comprises modules to take care of the active monitoring of individual connections as well as the detection of new connections that are potential candidates for overlay dissemination. Finally, a management component is also present which is responsible for selecting the Overlay Server to which the OAC will forward intercepted packets in case they require overlay routing. Performance testing has shown that on commodity desktop hardware (i.e., a 2 MHz AMD Opteron 2212 processor) a total of more than 40000 packets per second could be handled for packets of size 256 bytes, which corresponds to a processing bandwidth that exceeds 80 Megabits per second (Mbps) [De Vleeschauwer 10]. For a packet size of 1460 bytes, a bandwidth of more than 500 Mbps was achieved.

Because it is possible to locate the OAC on the client device or on existing hardware such as residential gateways and access nodes, the economical cost of deploying this component type can be quite low. For instance, due to the recent trend towards providing more intelligence in the network [Gilon - de Lumley 07], no essential hardware modifications are required to

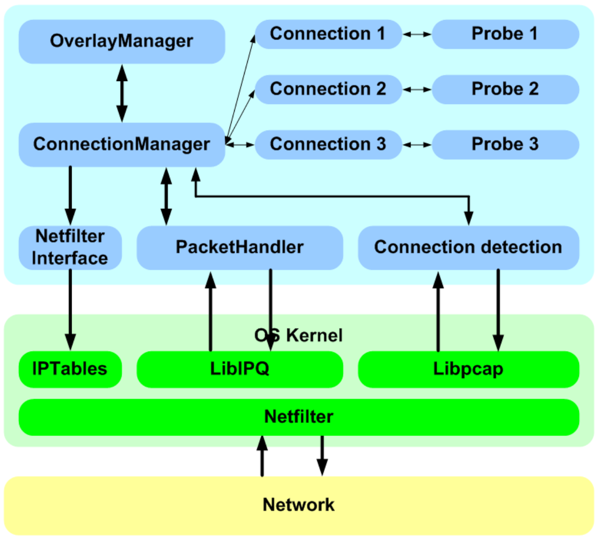


Figure 12.4: Overlay Access Component software architecture [De Vleeschauwer 08a].

the hosting equipment; instead, it suffices to develop new services for the access node's service platform which provide the OAC functionality.

Overlay Routing Exemplification

To clarify the tier-1 operation, consider the example scenario which is depicted in Figure 12.5. The simulated network topology consists of several ASs and includes 4 Overlay Servers, 2 Overlay Access Components and 2 hosts which are engaged in a real-time communication session. Both the routing tables of the Overlay Servers and the routing information that is maintained by the OACs are shown. The former relate target OS to next hop OS, while the latter maps destination IP address to OS and OAC at the remote site. At some point, an IP edge that is part of the direct route between Overlay Servers OS1 and OS4 starts exhibiting a degraded performance (e.g., it is suffering from congestion). This anomaly is responded to by the Overlay Servers by adjusting their routing tables accordingly. In particular, the routing table of OS1 now indicates that packets which are destined for OS4 should no longer follow the direct overlay link but instead need to be propagated to OS3 (see Figure 12.5). A similar entry can be found in OS4's routing table.

As the erratic edge lies on the direct IP path between the end-hosts, their communication session is impacted by the malfunction. The OACs which are

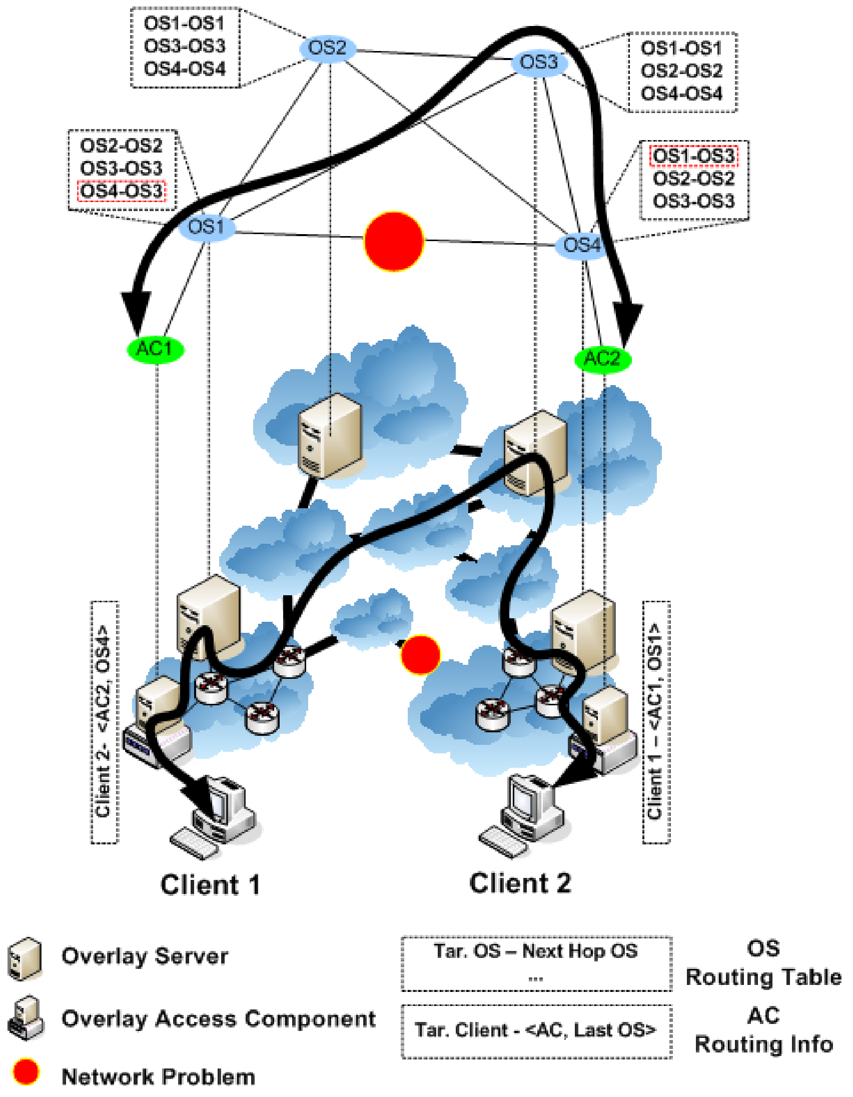


Figure 12.5: Illustrative overlay routing scenario.

associated with both clients will therefore start pushing host-emitted network traffic to the overlay network. The Overlay Servers will in turn disseminate the provided data via the optimal overlay path (i.e., in accordance with their routing table). In particular, the Overlay Servers will make sure that the failing IP link is circumvented by relaying network traffic between OS1 and OS4 through OS3 as an intermediate hop.

12.2.3 Tier-2 Functionality

The second layer of the platform is composed solely of NIProxy instances and hence requires little elaboration. The NIProxy components are deployed at the edge of the network (where they could for instance coincide with an OAC) and deliberately manage the delivery of content and data over the last mile connection of the destined host. In other words, tier-2 of the proposed architecture exploits the contextual knowledge and the dual network traffic engineering functionality of its constituting NIProxy instances to optimize the downstream transmission of content in the access network.

12.2.4 End-to-End Path Coverage

By integrating multiple types of QoE optimization entities, the proposed layered platform effectively combines their specific features and functionality. As a result, a synergy is achieved. In this particular case, the main advantage of the integrated approach is an extended coverage of the end-to-end IP path between pairs of communicating hosts. The tier-1 components focus on the network core and improve its routing performance in the event of disconnections or inconsistently performing IP links. At the same time, the second layer of the architecture concentrates on the last mile of the downstream content delivery path and addresses potential QoE optimization requirements that are introduced by the resource-constrained nature of this part of the network connection.

In case the content provider is deployed inside the network backbone, the proposed two-tier QoE optimization platform will cover the *complete* end-to-end route from source to sink. This scenario will for instance arise when a residential user accesses a multimedia streaming server. In case both endpoints of a network connection are however located inside an access network, the initial upstream part of the communication route will not be covered. More specifically, the transmission of data over the source's access connection will lack optimization. Only once the data reaches the boundary of the network core, the remainder of the data dissemination path will be managed by the proposed platform with the aim to improve the QoE that is witnessed by the destined user. In such environments, the proposed platform hence only achieves *near* end-to-end QoE optimization.

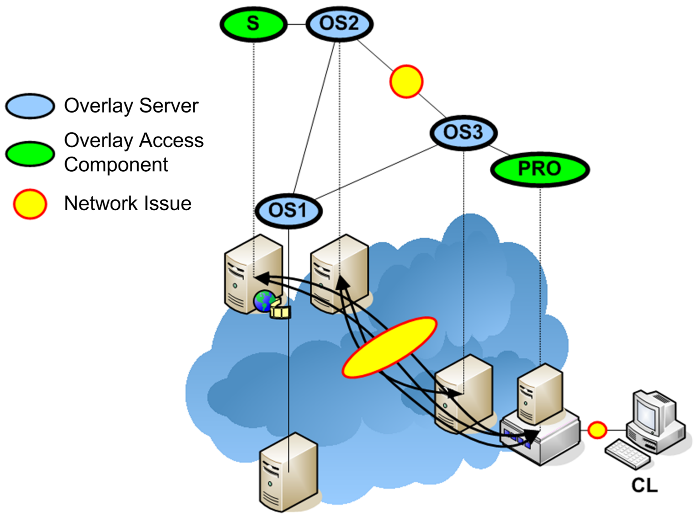
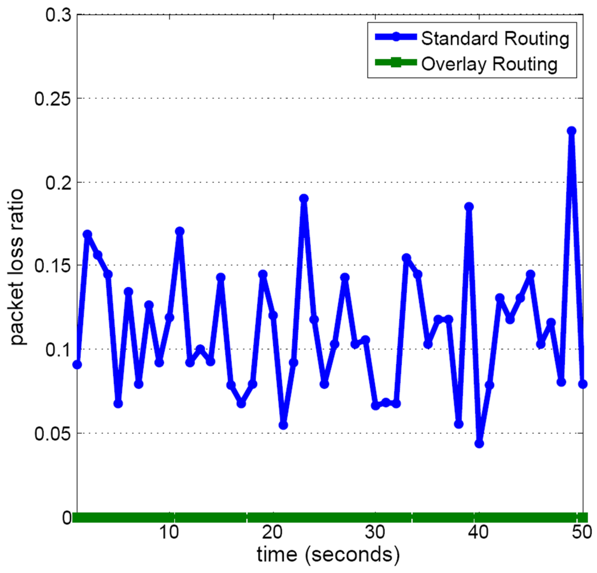


Figure 12.6: The testbed used for experimental evaluation.

12.3 Evaluation

12.3.1 Evaluation Testbed

To practically assess whether the integrated platform enables fruitful collaboration between its composing component types, a physical testbed was set up. The network topology that was emulated by this testbed is displayed in Figure 12.6. The testbed consisted of 8 off-the-shelf desktop PCs which all ran the GNU/Linux operating system. On these machines, three Overlay Servers, two Overlay Access Components, a NIProxy instance (PRO), a multimedia streaming server (S) and a client (CL) were installed. The NIProxy instance was logically positioned on the transition point where the client's access link connected to the network core and hence fulfilled the role of access node. The OAC instances were deployed on the hardware which hosted the NIProxy and the multimedia server. To be able to influence the performance of the emulated network topology, the testbed in addition included two impairment nodes which were based on Click technology [Kohler 00]. The first Click node artificially introduced random packet loss to simulate failing or malfunctioning links in the backbone of the network. In particular, one of the IP edges between OS2 and OS3 was subjected to an average packet loss rate of 10 percent. The second Click node on the other hand allowed the throughput of the last mile of the client's network connection to be manipulated.



Overlay link	Packets
OS2-OS1	7545
OS2-OS3	0
OS1-OS3	7545

OVERLAY LINK USAGE

Figure 12.7: Packet loss ratio witnessed on the route between the multimedia streaming server and the NIProxy instance, with and without overlay routing.

12.3.2 Experimental Results

Using the just described evaluation testbed, an experiment was conducted which involved the multimedia server simultaneously streaming 4 video fragments to the client machine. Each fragment was transmitted over a separate connection. The objective of the experiment consisted of ascertaining whether the composite platform succeeded in addressing the packet loss in the network core and whether it was at the same time able to adequately react to the throughput restrictions that were imposed by the last mile.

Mitigating Network Core Impairments

Figure 12.7 plots the packet loss ratio per second that was experienced in the network core during a 50 second time period, with and without overlay routing. Since the shortest route from server to client included the impaired IP link, the standard routing service propagated the video data through the lossy part of the network backbone. As can be deducted from the graph, the outcome was the loss of a considerable amount of network packets. At the destination, this packet loss in the network core yielded video decoding issues, which in turn led to the introduction of perceptual inconsistencies during the playback of the received video data. It is intuitively evident that visually distorted video playback intrinsically entails a detrimental impact on the QoE of the viewing user.

The graph in Figure 12.7 reveals that, in case overlay routing was exploited, the packet loss in the network backbone was completely eliminated. The overlay infrastructure automatically detected the packet loss issue on the direct path between source and destination and, in response, offered a backup route which exhibited superior QoS characteristics (i.e., an overlay route which was not affected by packet loss). This is confirmed by the table in Figure 12.7, which displays the overlay link usage statistics during the 50 second experiment interval. In particular, the table clearly indicates that the direct overlay edge between OS2 and OS3 was evaded; instead, the server-to-client network traffic was consistently routed through OS1 as an intermediate overlay hop.

Last Mile QoE Optimization

By optimizing the network backbone's routing performance, the Overlay Servers and Overlay Access Components succeeded in reliably delivering the video packets to the NIProxy instance that was incorporated in the testbed. At this stage, the second tier of the proposed QoE optimization platform came into action to direct the transmission of the video packets over the last mile of the client's network connection. Due to varying last mile conditions and altering user focus, this part of the experiment conceptually consisted of a succession of 5 discrete intervals. In particular, the transitions from the first to the second and from the fourth to the final interval were initiated by a modification of the available downstream access bandwidth. These throughput fluctuations were applied by the Click node that was incorporated in the access part of the emulated network topology. The remaining interval transitions were caused by user-initiated shifts in stream importance[1]. The last mile bandwidth alterations were automatically discovered by the NIProxy thanks to its network awareness. In contrast, the relative video importance knowledge was propagated to the NIProxy instance by the client software in the form of application awareness.

Figure 12.8 depicts the stream hierarchy which steered the last mile content streaming. As can be seen, an internal node of the `Priority` type was used to discriminate between the 4 involved video flows. In the illustration, the priority values that were assigned to the different video sessions are grouped horizontally per experiment interval(s). These values were calculated so that they directly reflected the user's input regarding relative stream significance.

[1]In the experiment, users could specify video stream preference through the GUI of the client software.

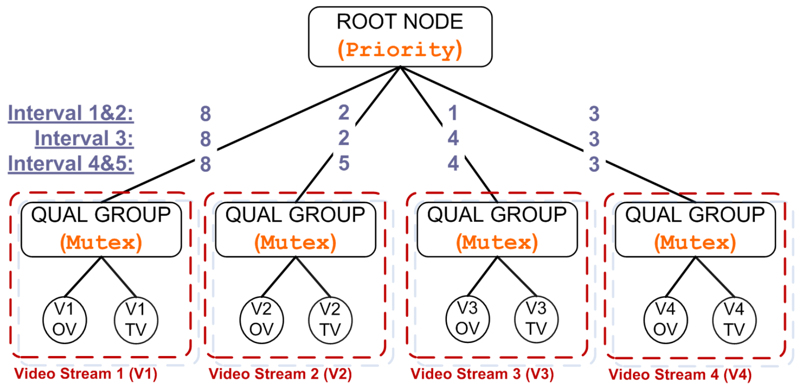


Figure 12.8: Stream hierarchy based on which the NIProxy managed client downstream bandwidth.

Also note that each video flow was incorporated in the stream hierarchy as a subtree instead of a single leaf node. This is explained by the fact that the NIProxy leveraged its (inbound) static video transcoding service during the experiment. As has been described in section 5.3.1, each individual video flow was therefore represented by two `discrete` leaf nodes, of which one corresponded to the Original Version (OV) of the video stream (i.e., the video fragment as it was emitted by the streaming server), whereas the other was associated with the stream's Transcoded Version (TV) (i.e., the lower-quality variant that was generated by the static video transcoding service). Both video qualities were grouped together using an internal node of type `Mutex` to guarantee that at all times at most a single quality variant would be assigned bandwidth.

The network trace depicted in Figure 12.9 illustrates the downstream access bandwidth allocation that was enforced by the NIProxy during the experiment. In this stacked graph, the dashed vertical lines separate the consecutive experiment intervals, while the red horizontal line indicates the downstream bandwidth capacity of the client's access link. The trace first of all indicates that the downstream last mile throughput was neatly respected throughout the test. This result can be attributed to the NIProxy's network awareness and the outcome was an optimal reception of the forwarded traffic at client-side. Stated differently, the video data that was actually streamed over the last mile connection was received by the client under ideal circumstances (i.e., with minimal delay and packet loss). A second important observation is the influence of the NIProxy's application awareness on the network traffic shaping outcome. By mapping relative video flow importance to node priority val-

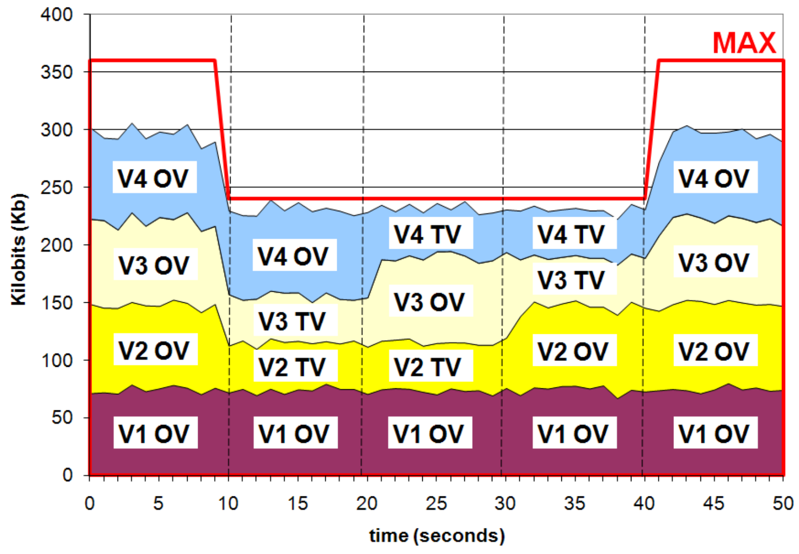


Figure 12.9: Stacked graph illustrating all video traffic received by the client.

ues, application-related information was successfully captured in the client's stream hierarchy. This in turn resulted in the bitrate of the least important video streams being reduced (by transcoding them to a lower quality) to preserve downstream access bandwidth for the forwarding of more relevant video flows. At any time during the experiment, the client consequently received the video traffic which it considered most significant at that moment at the highest possible quality. Both achievements are intuitively expected to have had a beneficial influence on the QoE that was provided to the end-user.

12.3.3 Discussion

The experimental results confirm that the resilient overlay routing service and the NIProxy's functionality were successfully combined in a layered QoE optimization platform. It has also been established that the platform's constituting entities focus on different aspects with regard to user QoE optimization and as such neatly complement each other. In particular, the findings from the video streaming case study demonstrate that

- its overlay routing infrastructure enables the platform to relay network packets around links in the network backbone which (temporarily) exhibit low QoS properties (e.g., which are suffering from packet loss)
- the incorporated NIProxy instances enable the optimization of last mile

content delivery; in the presented experiment for example, the NIProxy prevented the destined client's access network connection from being overwhelmed with data on the basis of its network awareness, while it exerted its application-related context to enforce an intelligent allocation of the downstream bandwidth that was actually available

Due to the complementarity of the proposed architecture's composing element types, a synergistic solution is yielded. More specifically, the video streaming case study illustrates that, in case the content origin is located inside the network backbone, the two-tier platform achieves complete coverage of the end-to-end network route from source to sink and is hence able to mitigate network-related problems that might arise anywhere along this path. The expected outcome is an improved QoE for the end-user: although no formal qualitative user inquiries were conducted, the results that were attained in the discussed experiment are very likely to have had a beneficial impact on user QoE.

12.4 Related Work

In previous research, the usage of overlay network technology for enhancing packet routing has already been looked at. The Resilient Overlay Network (RON) project describes a system for routing around network failures in which all involved parties are required to deploy an overlay routing server [Andersen 01]. The main difference with the overlay infrastructure of the two-tier platform that has been presented in this chapter is that the availability of the Overlay Access Component type relieves end-users from the necessity to host overlay routing servers themselves. This results in a more scalable solution than the RON methodology, whose applicability is limited to relatively small-scale overlay domains (i.e., environments which consist of about 50 distributed sites at most). De Vleeschauwer et al. have discussed in [De Vleeschauwer 04] a number of algorithms for determining optimal deployment locations for overlay routing servers. In [De Vleeschauwer 06], they in addition have introduced algorithms for the management of the overlay topology. Overlay technology can also be exploited to provide an overlay multicast service in case no native (i.e., network-layer) multicast support is available. In this context, De Vleeschauwer et al. have proposed an algorithm for solving the bounded diameter minimal cost Steiner tree problem [De Vleeschauwer 07a].

The composite platform which this chapter has introduced aims to improve the end-to-end QoS performance of transportation networks. As has already been established in section 2.1, frameworks for QoS provision abound in the literature. However, while these approaches uniformly support QoS provision in a single domain, only a minority is able to offer a full end-to-end solution. Typical impediments include scalability concerns and the severe cost that is associated with the requirement of implementing the proposed technologies in all the Autonomous Systems of the Internet. Furthermore, the problem of defining and enforcing Service Level Agreements (SLAs) between individual ASs has not yet been adequately solved either. In contrast, the proposed layered architecture is readily deployable in the Internet (and other IP-based networking substrates), at minimal cost and effort. The platform in addition does not restrict itself to QoS provision but instead addresses the larger issue of end-user QoE optimization.

12.5 Conclusions and Future Work

Accessing multimedia services via fixed and wireless networks has become common practice. Compared to traditional Internet applications such as web browsing, these services are much less tolerant to network anomalies like packet loss, bandwidth restrictions, congestion, delay and jitter. Such network complications can occur at arbitrary locations along the content dissemination path and are likely to significantly deteriorate the QoE that is perceived by the user of the multimedia service. This chapter has therefore proposed a two-tier platform which consists of QoE optimization components in the network core as well as at the edge. Thanks to its integrated design, the network architecture is able to implement (near) end-to-end user QoE optimization; only the upstream access segment of an end-to-end network route (if present) is not covered. Tier-1 of the platform is composed of overlay routing entities whose functionality achieves an optimization of the default routing service in the network backbone. In particular, through deliberate packet rerouting, this component layer enhances robustness to QoS and connectivity issues, which could for instance be caused by network congestion. Tier-2 of the platform encompasses NIProxy instances and is responsible for fine-tuning network traffic to the resource capacity of the access network connection of the destination. The NIProxy components in other words exploit their network traffic shaping and multimedia service provision tools to regulate the last mile data delivery in such a manner that the receiving user's QoE is optimized. Using a physical testbed, a WAN video streaming case study between a multimedia server and a receiving client has been conducted. The produced experimental results have corroborated that the proposed platform succeeds in maintaining a high user QoE when the network core experiences packet loss as well as in the event of access link bandwidth restrictions.

The work that has been presented in this chapter is important as it has demonstrated the possibility to fruitfully merge the NIProxy's functionality with the capabilities of other, independent QoE enhancement frameworks. In this particular case, the collaboration resulted in a more complete coverage, in terms of QoE monitoring and optimization, of the end-to-end route between content source and sink. Although this is already a significant achievement, the proposed distributed architecture is still susceptible to improvements, extensions and future research. A first topic of potential future work is scalability investigation. The platform is expected to scale since the inclusion of additional instances of its constituting component types (i.e., Overlay Server, Overlay Access Component and NIProxy) will normally result in a capacity increase. Actual scalability testing is however yet to be performed and the number of users that can be supported by a single instance of each of the composing component types has not yet been determined. Secondly, the platform could be extended with service hosting and resource discovery mechanisms. Third, it might be interesting to exploit the possibility to apply the NIProxy's network traffic shaping and multimedia service provision functionality not only in the downstream but also the upstream direction and to examine the implications hereof on the platform's QoE optimization options. Finally, organizing user studies might yield valuable qualitative feedback regarding the user QoE optimization operations that are executed by the integrated platform.

Chapter 13

Increasing Bandwidth Brokering Flexibility via Dynamic Video Transcoding Support

In all prior case studies that involved the real-time exchange of video data, the NIProxy's static video transcoding service has been exploited. To recapitulate, this service enables the NIProxy to on-the-fly transcode H.263-encoded video streams to a lower quality and hence to reduce their bandwidth requirements. As was discussed in section 5.3, the video transcoding service is labeled *static* because the desired transcoding parameters need to be specified on service instantiation and cannot be modified at run-time. All bitstreams that are produced by a particular instance of the video transcoding service will hence exhibit identical quality characteristics.

Previously presented experimental results have clearly exemplified the static video transcoding service's added value. In particular, the service provides the NIProxy with additional latitude when performing network traffic shaping. Thanks to the existence of the service, additional bandwidth brokering options are unlocked since it for instance allows for the bitrate of specific video flows to be reduced so that bandwidth is conserved for the dissemination of more important network traffic. Similarly, the service enables the NIProxy to produce network traffic shaping results in which the distributable bandwidth is consumed more completely (i.e., a smaller portion of the available capacity remains unallocated). As a final advantage, the service under certain circumstances prevents the NIProxy from being forced to completely discard individual video streams. In particular, during times of bandwidth deficiency, the video transcoding service unlocks the possibility to forward a lower-quality video version which, contrary to the original stream, might still fit in the current bit budget. In terms of user QoE, receiving a lower-quality variant of a video stream is often preferred to receiving nothing at all.

To summarize, an important contribution of the static video transcoding service is that it introduces a certain amount of flexibility in the network traffic shaping process. This flexibility is however constrained by the service's static nature. Requiring the video transcoding configuration to be fixed during service execution inherently yields rigidity. This chapter will therefore present a novel plug-in for the NIProxy that supports *dynamic* video bitrate adaptation [Wijnants 08c][Wijnants 10]. Contrary to its static counterpart, the dynamic video transcoding service is targeted at H.264/AVC-encoded video bistreams. This is motivated by the fact that the H.264/AVC specification is more elaborate than its H.263 predecessor and incorporates additional options for fine-scale bitrate modification. Another advantage of the H.264/AVC standard is that it achieves a much higher coding efficiency: for equivalent quality settings, a H.264/AVC encoder will economize approximately half the bitrate compared to H.263. It will be shown that, by leveraging these additional features and benefits, the proposed NIProxy plug-in succeeds in transforming video flows to an arbitrary bitrate in real-time. To corroborate this claim, representative experimental results will be provided that were realized in the context of the two-tier QoE optimization platform which has been described in chapter 12. The experimental findings will also illustrate that bundling the NIProxy's network traffic shaping functionality with the novel H.264/AVC transcoding module enables highly dynamic and flexible bandwidth brokering results to be generated. In other words, it will be validated that the availability of the H.264/AVC service considerably extends the QoE optimization features and effectiveness of the NIProxy and hence, through extrapolation, of the two-tier platform.

This chapter describes the continuation of the research collaboration between Hasselt University and Ghent University that had previously resulted in the realization of the integrated QoE optimization architecture (see chapter 12). In particular, the H.264/AVC video transcoding technology that will be leveraged in this chapter has been developed at the Multimedia Lab (MMLab) research group of Ghent University. It will be shown that this technology could easily be incorporated in the QoE optimization platform in the form of a plug-in for the NIProxy. Note that this implies that the results that will be presented in this chapter were achieved without requiring any modification to the design or implementation of the integrated platform or any of its constituting component types.

13.1 Background

As video is presumably the most challenging type of multimedia traffic, especially in terms of throughput demands, the ability to alter the bitrate of video data plays an essential role in an architecture that aims to deliver end-to-end QoE. Many different approaches exist to actively modify the bandwidth requirements of a coded video fragment. A straightforward solution consists of cascading a full decode and successive re-encode step to ensure that the output bitstream exhibits the desired properties (i.e., conforms to a certain target bitrate). As was already mentioned in section 5.3.4 however, the major drawback of such a *cascaded pixel-domain strategy* is that it performs a number of calculations unnecessarily and that it is hence computationally inefficient as well as expensive. This is for instance exemplified by the undesired computational overhead that results from the needless re-calculation of the motion field. Given the complexity of the specification, the applicability of H.264/AVC transcoders that are implemented according to this paradigm is limited to off-line scenarios (if deployed on present-day commodity hardware). In contrast, a second class of techniques directly alters the properties of video streams in the compressed domain (i.e., without first performing video decoding). As these methods perform their calculations directly on transformed video coefficients, only entropy decoding and encoding have to be performed. Such methods are generally referred to as *transcoding algorithms*. Compared to cascaded pixel-domain solutions, they are less straightforward to implement but are at the same time able to achieve a much higher computational efficiency.

In general, single-layer video flows can be transformed along three axes: resolution, framerate and visual quality. This is typically termed spatial, temporal and quality modification, respectively. Transcoding algorithms for quality adaptation can be classified as being based on either *Dynamic Rate Shaping* (DRS) or *requantization*. The former category achieves video bitrate reduction by selectively eliminating transform coefficients. Requantization-based schemes on the other hand do not eliminate but instead modify transform coefficients by coarsening the quantization process. As a result, methods from the latter category require recoding-like implementations and hence incur a considerable computational penalty compared to the DRS approach.

Conceptually, DRS-based solutions provide an interface (or filter) between the video source and the communication channel, by means of which the encoder's output can be matched to the currently available network bandwidth [Jacobs 98]. To lower the bitrate of video flows, this class of transcoding algorithms sets one or more coefficients of a transformed and quantized block of pixels to zero. This is achieved by either dynamically adjusting a cut-off frequency or by enforcing a transform coefficient level threshold. These alternative approaches are respectively denoted by *Frequency-Dependent DRS* (FD-DRS) and *Level-Dependent DRS* (LD-DRS).

A detailed study of the FD-DRS and LD-DRS methods, including a comparison of their performance, can be found in the published article on which this chapter is based [Wijnants 10] and will not be repeated here. It suffices to know that LD-DRS nearly always outperforms FD-DRS in terms of resulting visual quality. In other words, given a certain target bitrate, the LD-DRS method will typically yield perceptually less distorted video fragments. It has been decided to omit the technical details for two reasons. First, the focus of this chapter is on the usability and added value of dynamic video transcoding with regard to QoE optimization, not on the video transcoding functionality itself. Second, the H.264/AVC transcoding technology has been developed by fellow researchers from Ghent University and hence does not represent own work.

13.2 Implementation

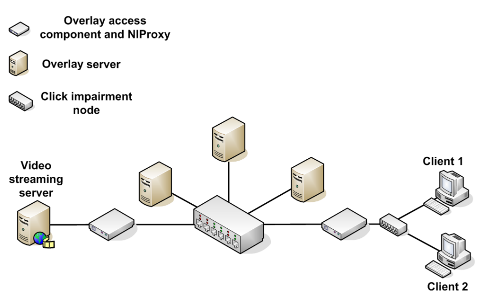
The dynamic video transcoding service for the NIProxy is targeted specifically at H.264/AVC quality reduction. Stated differently, the service does not modify the resolution nor the framerate of input video fragments. As streaming scenarios represent one of the NIProxy's most important and prominent application domains, a major design goal for the service was real-time operation. Given the real-time requirement, the aim was to minimize the computational complexity and to achieve a low delay, while at the same time maintaining an acceptable visual quality. This in turn motivated the choice for a true transcoding algorithm instead of a cascaded pixel-domain solution. Section 13.1 has pointed out that two fundamental categories of transcoding algorithms for quality transformation exist, namely DRS-based methods and approaches that rely on requantization. Of these alternatives, the former category was preferred due to its lower computational footprint. In particular, based on a comparative study of the performance of the frequency- and level-dependent DRS variants (see [Wijnants 10]), it was decided to opt for a LD-DRS implementation. Besides the LD-DRS functionality itself, the dynamic video transcoding service incorporates a rate control algorithm which steers the transcoder to ensure that the desired video bitrate is achieved. Technical details regarding the operation of the rate controller can again be found in [Wijnants 10].

13.3 Implications on the QoE Optimization Platform

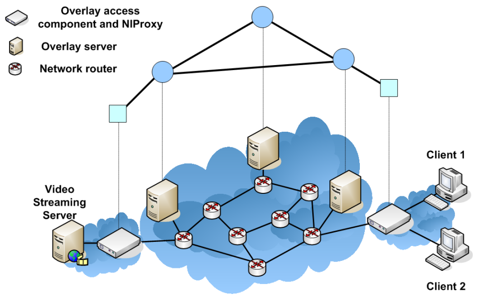
The availability of the dynamic video transcoding service enables the NIProxy to dynamically set the desired bitrate for H.264/AVC-encoded bitstreams during their propagation through the transportation network. As such, the service by itself already forms a valuable addition to the NIProxy's feature list as it could, for instance, be exploited to map a video stream to the current bandwidth capacity of a residential user's access connection in a Video on Demand (VoD) setting. More importantly however, the service extends the network traffic shaping options of the NIProxy since it unlocks the possibility to represent H.264/AVC video flows in the stream hierarchy via `continuous` leaf nodes. Recall from sections 4.3.2 and 8.2.1 that an important requirement for the correct functioning of this leaf node category is the ability to effectively and accurately enforce the bandwidth budget which they compute for their associated network stream. By linking the `continuous` leaf node to the dynamic video transcoding service, this requirement can be satisfied for H.264/AVC network traffic. Since `continuous` leaf nodes add a considerable amount of flexibility to the NIProxy's bandwidth brokering process, the H.264/AVC transcoding service significantly contributes to the calculation of highly dynamic network traffic shaping results that can react both promptly and adequately to context changes.

13.4 Evaluation

This section will evaluate the added value, in terms of user QoE optimization, of the novel dynamic video transcoding functionality. In particular, this section will examine whether the service allows additional flexibility to be introduced in the NIProxy's network traffic shaping process by enabling the adoption of `continuous` leaf nodes for the management of H.264/AVC video traffic.



(a) Physical layout of the testbed



(b) Emulated network topology

Figure 13.1: Testbed for the dynamic video transcoding evaluation.

13.4.1 Evaluation Testbed and Experimental Setup

Instead of leveraging the NIProxy as a stand-alone system to investigate the dynamic video transcoding service, it was decided to stage the validation in the broader scope of the two-tier distributed QoE optimization architecture which has been introduced in chapter 12. Similar to the assessment method for the platform's general characteristics (see section 12.3), a physical testbed was deployed to enable thorough investigation of the influence of the novel H.264/AVC transcoding functionality on the QoE optimization capabilities of the NIProxy and hence, by extension, of the layered platform. Figure 13.1 displays the physical layout of this testbed as well as the network topology that it conceptually corresponded to. In total, the testbed consisted of 10 PCs running GNU/linux and encompassed three Overlay Servers, two nodes on which both an Overlay Access Component and a NIProxy instance were hosted, two multimedia clients and one streaming video server. To enable the emulation of heterogeneous network conditions, two Click impairment nodes were also incorporated in the testbed [Kohler 00]. These nodes served a dual purpose since they were employed to artificially introduce arbitrary packet loss in the core network as well as to enforce downstream bandwidth restrictions in the clients' access network.

To evaluate the QoE optimization platform and, in particular, its novel dynamic video transcoding support, H.264/AVC streaming sessions were established between the video server and both multimedia clients. However, the end-to-end network connection of only one of the clients was managed by the platform. In the following discussion, the unmanaged client will represent the reference scenario against which the results that were achieved by the distributed platform will be offset. This will allow for the effects of the

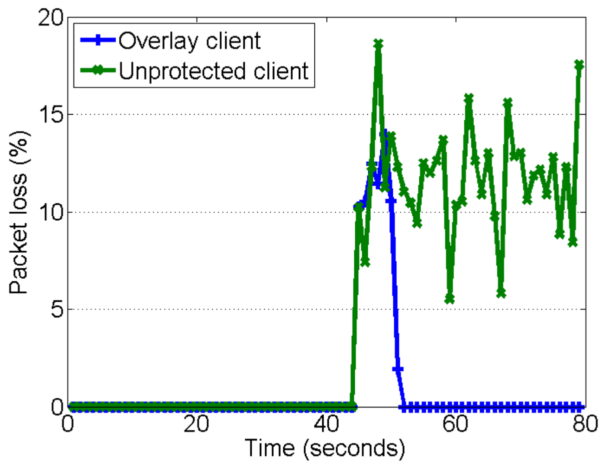


Figure 13.2: Packet loss ratio per second, with and without overlay routing.

platform's operations to be comprehensively visualized and for the impact of the availability of the H.264/AVC video transcoding service on user QoE to be underscored.

13.4.2 Experiment 1: Optimizing Network Core Routing

In a first test, an arbitrary H.264/AVC video sequence was streamed from the multimedia server to each of the two clients. Approximately 43 seconds after the beginning of the experiment, 10 percent random packet loss was activated in the core network by the Click impairment node. As a result, the communication session of both clients started suffering from lost packets. This in turn yielded video decoding issues at destination-side and hence a degraded video playback on the clients' screens. After a few seconds however, the QoS issue in the network backbone was recognized and reacted against by the QoE optimization platform. In particular, the platform began to bypass the compromised network segment by routing network packets that were destined for the managed client via an alternative overlay path.

The packet loss ratio that was witnessed by both clients during the experiment is plotted in Figure 13.2. The graph clearly illustrates that as soon as the QoE optimization platform actuated its rerouting functionality, packet loss was expelled from the managed client connection. The unprotected client on the other hand continuously suffered from a degraded performance, since its destined packets were not rerouted and hence continued to pass through the problematic (i.e., lossy) part of the network core. These results confirm

Table 13.1: Quality parameters of the video sequences employed in the second experiment.

	Video1 (V1)	**Video2 (V2)**
Resolution (pixels)	352 × 288 (CIF)	352 × 288 (CIF)
Framerate (FPS)	25	25
GOP size	25	25
Bitrate (KBps)	82	90

the findings from section 12.3.2, namely that the first tier of the platform is capable of successfully countering QoS impairments such as packet loss in the network backbone. Since this chapter is specifically dedicated to the investigation of the impact of the dynamic video transcoding service's incorporation in the QoE optimization architecture and since this service only influences the operation of the architecture's second tier, the tier-1 experimental results have only been provided for the sake of completeness and will not be elaborated upon.

13.4.3 Experiment 2: Exploiting Dynamic Video Transcoding on the Last Mile

To evaluate the NIProxy component and its H.264/AVC transcoding plug-in, a second experiment was performed. This experiment staged the simultaneous streaming of two H.264/AVC-encoded bitstreams (V1 and V2) to both multimedia clients while their last mile network connection was artificially impaired in terms of available downstream bandwidth. The principal characteristics of the employed video sequences are listed in Table 13.1. The goal of this experiment was twofold:

- Investigate the NIProxy's reaction to the dynamically varying downstream throughput of the client's access connection

- Verify whether the NIProxy succeeded in improving the user QoE by exploiting its H.264/AVC video transcoding service to dynamically and intelligently adapt the involved video flows so that their bitrate matched the last mile bandwidth capacity

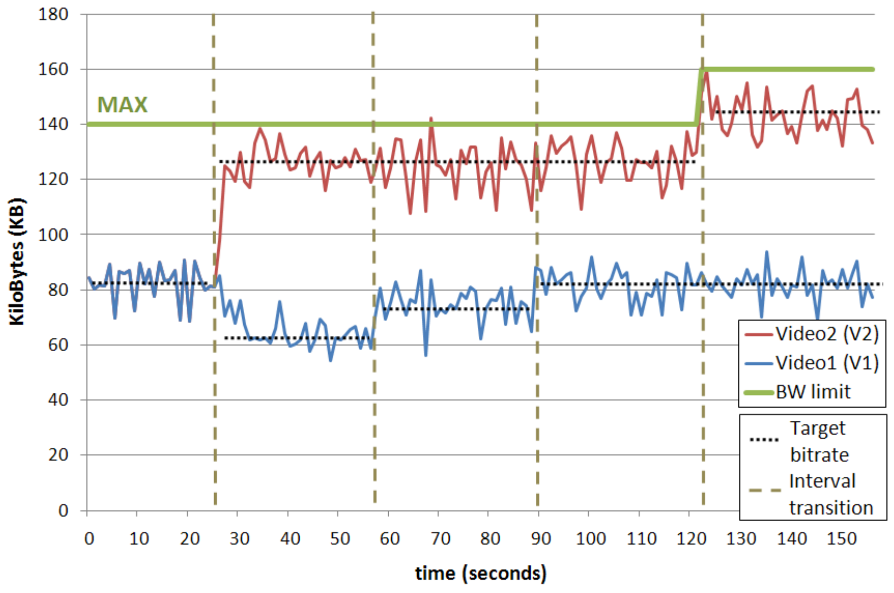


Figure 13.3: H.264/AVC data received by the QoE-managed client (stacked graph, in KBps).

The experiment itself conceptually spanned 5 successive intervals. Each interval transition was triggered by a single state change. In particular, the switch from the first to the second period was caused by the introduction of the second H.264/AVC flow (i.e., V2) in the experiment, whereas the two subsequent interval transitions were initiated by shifts in H.264/AVC stream importance (which users could indicate via the GUI of the client software). Finally, in the fifth interval, the last mile impairment was moderated so that the clients suddenly disposed of an increased amount of downstream bandwidth. To be more precise, whereas the Click node thus far had restricted each client's downstream access bandwidth to 140 KiloBytes per second (KBps), this impairment was at the beginning of the final experiment interval relaxed to 160 KBps.

Figure 13.3 plots all H.264/AVC network traffic that was received by the platform-enhanced client during the experiment. The stream hierarchy by means of which the NIProxy calculated this bandwidth distribution is displayed in Figure 13.4. Due to the constrained nature of the experiment, the stream hierarchy entailed only 3 nodes and had a very straightforward layout. In particular, the root of the hierarchy consisted of an internal node of type `WeightStream`, which was used to differentiate between the two H.264/AVC

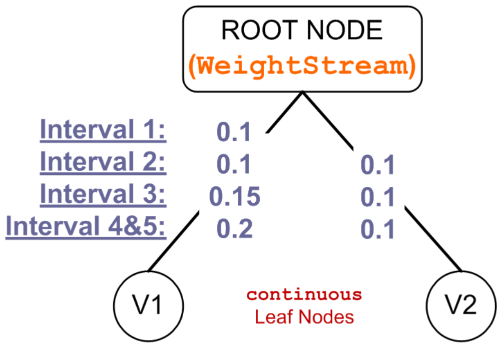


Figure 13.4: Stream hierarchy that directed the downstream bandwidth brokering.

flows that were involved in the test. Both these flows were represented by a `continuous` leaf node. Weight value determination for these leaf nodes occurred according to the relative importance of the corresponding H.264/AVC flows, as specified by the user. The exact weight values as they applied during the different experiment periods are provided in the stream hierarchy illustration. Finally, recall from section 13.3 that the NIProxy's H.264/AVC transcoding service was exploited to dynamically transcode the video streams to the target bitrates that were calculated by their associated `continuous` leaf node in each iteration of the NIProxy's bandwidth brokering process.

Analysis of the presented network trace teaches that, in the initial experiment interval, only one H.264/AVC flow was present (i.e., V1). Since sufficient last mile downstream bandwidth was available to forward this flow to the destined client at its maximal (i.e., original) quality, no video transcoding operations were performed by the NIProxy. This situation changed with the introduction of V2 in the second experiment period. Since both flows were assigned identical weight values in the stream hierarchy and had comparable maximal bitrates (which can respectively be deduced from Figure 13.4 and Table 13.1), they were granted a comparable bandwidth budget by the NIProxy's network traffic shaping mechanism. In particular, each stream was allocated a bandwidth amount of approximately 62 KBps. At the beginning of the two subsequent intervals, the user incremented the importance value of the first H.264/AVC flow (see Figure 13.4). This resulted in V1 being apportioned an increased share of the available downstream bandwidth at the expense of the second H.264/AVC flow, which now needed to be transcoded to a lower bitrate. The exact progression of the target video bitrates during the experiment is provided in Table 13.2 and is visualized in the network graph

Table 13.2: Target bandwidth evolution for the H.264/AVC bitstreams during the second experiment, in KBps.

	Experiment Interval				
	1	**2**	**3**	**4**	**5**
Video1 (V1)	82	62	72	82	82
Video2 (V2)	/	62	52	42	62

by means of horizontal dotted lines. In the final interval, the additional 20 KBps last mile bandwidth that became available was exclusively exploited to upgrade the quality of the second H.264/AVC stream. Given its lower importance value, this behavior might seem surprising. An explanation can however be found in the two-step operation of the `WeightStream` node (see section 4.2.4) which controlled the distribution of the available bandwidth over both video flows. In the initial bandwidth allocation phase, a fraction of the newly available downstream capacity was indeed reserved for V1. Due to its higher weight value, this amount even exceeded the additional bandwidth that was initially granted to the second H.264/AVC bitstream. However, as V1 was already being forwarded to the client at its maximal bitrate, it could not capitalize the extra bandwidth that was assigned to it. In the second phase of the `WeightStream`'s bandwidth distribution process, this excess capacity was transferred integrally to sole sibling V2.

Remark from Figure 13.3 that the transcoded video flows exhibited a relatively high variability in their bandwidth consumption and that they sometimes quite significantly deviated from their target bitrate. This behavior fell outside the control of the NIProxy. In particular, it was caused by the rate controller that is incorporated in the dynamic video transcoding service, which only succeeds in guaranteeing that the *average* video bitrate conforms to the target value. The NIProxy's overflow prevention buffer (see section 4.6), whose size was set to 10 percent of the client's downstream throughput, absorbed the vast majority of the bitrate peaks that resulted from the rate controller's suboptimal operation; only approximately 70 seconds after the start of the experiment, it could not prevent the downstream capacity from briefly being violated.

A number of important findings can be distilled from the presented experimental results. A first observation is that the NIProxy, by leveraging its network awareness, ensured that the downstream capacity of the last mile was roughly[1] respected throughout the entire experiment. This behavior has already been encountered a number of times during the discussion of preceding experimental evaluations of the NIProxy. To recapitulate, the outcome was an avoidance of access link congestion, which in turn resulted in a minimization of packet loss and delay and consequently an optimal reception of the forwarded network traffic at client-side. Secondly, it appears that the enforced bandwidth distribution correctly captured relative H.264/AVC stream importance information. More specifically, bandwidth was apportioned among the involved flows directly proportionally to their significance, from the user's point of view. This result can be attributed to the NIProxy's application-related contextual knowledge. Lastly and most importantly, as is demonstrated in the network trace in Figure 13.3, the H.264/AVC transcoding service succeeded in sufficiently accurately approximating the bandwidth amounts that were reserved for the H.264/AVC-encoded bistreams by their `continuous` leaf node representations in the stream hierarchy. The availability of the service in other words allowed the NIProxy to adapt the involved video streams to a continuous range of bitrates. As such, it introduced extra flexibility in the network traffic shaping process and enabled an optimal and complete exploitation of the bandwidth that was available on the last mile.

The findings described in the previous paragraph did not apply to the video streaming session of the client that was not managed by the QoE optimization platform. Due to the lack of facilities to shape network traffic prior to its injection into the access network and due to the inability to profit from H.264/AVC transcoding functionality, a far from optimal usage experience was yielded. To be more precise, the user did not dispose of any mechanisms to discriminate between the incoming H.264/AVC bitstreams (in terms of relative significance), let alone to enforce differential flow treatment. In addition, since the client's access connection was flooded with more data than it could transfer, a substantial amount of last mile packet loss was detected during the experiment. Parts of the encoded video data were consequently not received by the client, which in turn caused the appearance of perceptual artifacts in the decoded video sequences. The platform-managed user did not suffer from such deteriorated video playback and was hence provided with a far superior usage experience.

[1]Only approximately 70 seconds after the start of the experiment, a brief violation of the downstream access capacity occurred. It has however already been explained that this deficiency is attributed to the rate control algorithm of the H.264/AVC transcoding service and hence not to the operation of the NIProxy itself.

13.4.4 Discussion

The results that were achieved during the described experiments comprehensively demonstrate the advantages of the two-tier QoE optimization network. In particular, largely analogous to the findings from chapter 12, they confirm that the platform is capable of

- successfully mitigating impairments and QoS anomalies such as congestion and packet loss in the network backbone
- allocating the downstream bandwidth that is available on the last mile in a deliberate manner (i.e., in harmony with contextual factors)

For reasons of clarity and intelligibility, these features were illustrated via independent experiments. It is important to remark however that they are definitely not mutually exclusive and can hence just as well be achieved simultaneously (see the experimental results in section 12.3 for confirmation). This implies that the proposed platform supports user QoE optimization along the (nearly) complete delivery path from content provider to consumer.

A significant difference from the practical evaluation of the two-tier platform that has been provided in section 12.3 is that the platform this time incorporated functionality which enabled the bandwidth consumption of H.264/AVC bitstreams to be modified in a fine-grained manner. The experimental results confirm that the operation of the first tier of the platform was not affected by the availability of this additional functionality. The second tier on the other hand did substantially benefit from it, since the platform was now able to transcode H.264/AVC bitstreams to a continuous range of bitrates before relaying them to the client's access network. In the performed experiment, the involved video sessions were transcoded to exactly the amount of bandwidth that, according to the calculations of the NIProxy's bandwidth brokering scheme, was available for their last mile transmission. As a result, the involved H.264/AVC network traffic was meticulously tailored to the last mile throughput restrictions, without losing sight of other contextual factors in the process. The outcome was an optimal and complete exploitation of the downstream throughput of the client's access connection, which is likely to be appreciated by the end-user.

13.5 Conclusions

This chapter has extended the two-tier QoE optimization architecture that has been introduced in chapter 12 with real-time dynamic transcoding technology for H.264/AVC bitstreams. This functionality has been integrated in the form of a plug-in for the tier-2 NIProxy components. Given the platform's focus on streaming and real-time content dissemination settings, a crucial design goal for the transcoding plug-in was to minimize the computational complexity and delay introduction. Based on thorough investigation and appraisal of existing H.264/AVC transcoding techniques [Wijnants 10], it has been decided to implement the plug-in according to the Dynamic Rate Shaping (DRS) approach. The plug-in in addition incorporates a dynamic rate control algorithm to steer the DRS operation; as such, it is guaranteed that the transcoded video conforms to its target bitrate.

The salient characteristic of the H.264/AVC transcoding plug-in is that it enables the transformation of H.264/AVC bitstreams to an arbitrary bandwidth. The service which it provides has therefore been labeled *dynamic* video transcoding. It is evident that it excels the static video transcoding service which has been introduced in section 5.3 in terms of functionality. The discussion of representative experimental results has however revealed that this superior performance also applies to the QoE optimization prospects that are entailed by both services. In particular, it has been validated that the availability of the dynamic video transcoding service allows the NIProxy to adopt `continuous` leaf nodes to represent H.264/AVC-encoded video flows in the stream hierarchy. The idea is to leverage the service to dynamically transcode H.264/AVC bitstreams to the target bandwidth budget that was calculated by their embodying `continuous` leaf node during network traffic shaping. As a result, the dynamic video transcoding service enables highly dynamic network traffic shaping results to be computed which can adapt both promptly and effectively to context changes. Notice the substantial contrast with the static precursor, which interfaces with the bandwidth brokering process by means of a `discrete` leaf node and hence forces the NIProxy to choose between a limited number of alternatives with regard to the dissemination of video content (instead of enabling the content to be transcoded to a continuous range of bitrates). It is hence justified to state that the dynamic video transcoding service adds a considerable amount of flexibility to the NIProxy's bandwidth brokering process. As such, it extends the NIProxy's range of QoE optimization options as well as improves the QoE optimization results and effectiveness that can be attained by the NIProxy (and hence, through extrapolation, by the enclosing two-tier platform).

As a secondary contribution, this chapter has exemplified the advantages that are associated with two of the NIProxy's distinctive features. First of all, the positive implications of the plug-in-based implementation of the NIProxy's multimedia service provision functionality have been implicitly illustrated. To furnish the NIProxy with dynamic H.264/AVC video transcoding support, it sufficed to develop an additional plug-in. Since implementation-wise plug-ins are stand-alone modules, this process did not require any modification whatsoever to the NIProxy's general software architecture. Stated differently, the dynamic video transcoding service has demonstrated that its multimedia service provision methodology enables rapid and clean extension of the NIProxy's feature list. Secondly, the developed plug-in is another interesting demonstration of the interoperation interface that is defined between services and the NIProxy's network traffic shaping mechanism. In this concrete case, the dynamic video transcoding service has been employed to transcode H.264/AVC bitstreams to the bandwidth amounts which they had been assigned during network traffic shaping by their `continuous` leaf node embodiment in the stream hierarchy. The presented experimental results have highlighted the beneficial influence of this collaboration on the user QoE optimization process (i.e., it allows for the generation of highly dynamic, flexible and effective bandwidth brokering outcomes).

Chapter 14

Conclusions and Directions for Future Research

14.1 Conclusions

The ultimate goal of each application should be user satisfaction and optimal usage experience. While this is already a far from trivial task in stand-alone settings, it is an even more complicated assignment for distributed applications due to the extra constraints and potential pitfalls that are associated with communicating data via a transportation network.

This dissertation has described the Network Intelligence Proxy (NIProxy), a network intermediary which has been developed to enable Quality of Experience (QoE) optimization in IPv4-based telecommunications networks. As its name suggests, the NIProxy's methodology is centered around the principle of introducing *intelligence* in the networking infrastructure. This is achieved by querying three distinct sources of contextual information, namely the transportation network itself, the distributed application and the end-user. Based on its accumulated contextual knowledge, the NIProxy engineers the network traffic that passes through it and as such improves the traffic handling capabilities of the transportation network in which it has been incorporated. The specific focus hereby is on multimedia content due to the stringent requirements that are associated with its network dissemination.

The first traffic engineering technique that is supported by the NIProxy is *network traffic shaping* (NTS), which implies that the NIProxy enables the in-network orchestration of the bandwidth consumption behavior of distributed applications. This entails two general objectives. First of all, the NTS process needs to guarantee that the distributed application does not violate the bandwidth volume that is available for the dissemination of the network traffic which it induces. Failure to do so may result in network congestion and all the detrimental effects that it evokes (e.g., unpredictable network performance, increased latency, data corruption and packet loss, etcetera). The second objective consists of exploiting the available bandwidth budget as effectively as possible or, stated differently, in such a manner that the QoE of the user of the distributed application is maximized. The NIProxy's NTS scheme operates by capturing the relationships that exist between network traffic types (or even individual network flows) and other contextual knowledge in a tree-like hierarchy. The layout and the composition of the hierarchy will determine how the available bandwidth will be apportioned.

The validity of the NIProxy's NTS approach as well as its positive implications in terms of QoE optimization have been established in numerous experimental evaluations. As an example, it has been experimentally confirmed that the NIProxy succeeds in managing network bandwidth in the presence of competing real-time and non-real-time network traffic. These traffic types exhibit widely dissimilar properties and each consequently imposes distinct bandwidth brokering requirements. The NIProxy's NTS scheme has proven to be sufficiently elaborate and sophisticated to be able to regulate the bandwidth consumption of both categories of network traffic, even simultaneously. This has been demonstrated by means of demonstrator software that was developed specifically for evaluation purposes, but also by leveraging the NIProxy to coordinate the bandwidth utilization behavior of a real-world Networked Virtual Environment application that demands efficient distribution of rendering-related data and in addition supports real-time streaming of audiovisual content.

The NIProxy's second tool to manipulate network data dissemination, and hence user QoE, is *service provision*. The NIProxy incorporates a substrate for the hosting and execution of *services* on intercepted network traffic (especially traffic which transports multimedia content). Analogous to the NTS scheme, the service provision framework is context-aware. As a result, it is possible for services to achieve context-adaptivity by attuning their operation to the current context of use, which will in turn ensure that the processing which they implement will actually lead to an improvement of the end-user experience. The implementation of this QoE optimization technique is plug-in-based. Perhaps the most important advantage that is conferred by this design is that it allows services to be dynamically installed during operation and that it hence guarantees run-time extensibility of the NIProxy's supported functionality. As such, it becomes possible for the NIProxy to cope with heterogeneous conditions in dynamic networking environments without requiring a reboot (i.e., without interrupting the service for currently managed hosts).

The range of services that can be hosted by the NIProxy is theoretically boundless. Some notable examples have been described in the course of this thesis and their QoE optimization potential (and hence, by extension, that of the NIProxy's service provision framework) has been confirmed by means of experimental evaluation. As an example, a service has been developed which introduces real-time transcoding functionality in the NIProxy to enable it to on-the-fly reduce the bitrate of H.263 video. The service is said to implement "static" transcoding since all bitstreams that are output by a certain instantiation will exhibit identical video quality parameters and will have widely comparable bandwidth requirements. In nearly all the presented case studies that involved the real-time exchange of video data, the static video transcoding service has been exploited. The resulting experimental findings have corroborated that the service holds interesting possibilities in terms of QoE optimization, since it enables the user to be provided with a lower quality variant in situations where the NIProxy would otherwise be forced to mercilessly discard the original video and where the user would hence not receive any video at all. At a later stage, a dynamic counterpart of this service has been developed that supports bitrate transcoding of H.264/AVC network traffic to an arbitrary and at run-time adjustable target value. The practical evaluation of this service has revealed that it introduces additional flexibility compared to its static precursor and that it allows for a more complete and efficient utilization of the available network bandwidth. A final example is the Forward Error Correction (FEC) service, which adds XOR-based parity protection to network traffic to enable the receiver to remedy data corruption and even partial data loss that is induced during network dissemination. It has again been experimentally verified that this service represents a promising and versatile asset in the quest for QoE optimization and that it hence forms a valuable addition to the NIProxy's feature list. Important to note is that the functionality that is provided by all described services has been incorporated in the NIProxy without requiring any modification whatsoever to the NIProxy's general software architecture.

A defining characteristic of the NIProxy is that it synthesizes its dual traffic engineering facilities in an interactive and collaboration-enabled manner.

Instead of implementing the bandwidth brokering and service provision frameworks as isolated entities, an integrated design has been adopted that enables both techniques to cooperate during QoE optimization. The outcome is a holistic solution in which the individual components supplement each other and improve their QoE optimization potential through collaboration. This is for instance illustrated by each of the three example services that were mentioned in the previous paragraph. Since these services each introduce a new sort of network flow, they leverage the interface with the NTS mechanism to inform the latter of the existence of this network traffic type and of its requirements in terms of network bandwidth. From this point on, the provided information will be taken into account by the NIProxy's bandwidth brokering operations. Conversely, each of the described services interpellate the NTS framework to determine whether and, in the case of the dynamic video transcoding and FEC services, exactly how they should apply their functionality to transiting data. This implies that the QoE optimization benefits that have been experimentally established to be associated with the mentioned services are not solely the merit of the functionality that they implement but, just as importantly, also of their correct interaction with the NTS process. As such, they have exemplified that the NIProxy's integrated approach with regard to traffic engineering results in QoE optimization performance and prospects that transcend potential achievements when both mechanisms are applied independently.

The range of conceivable QoE improvement operations is so extensive that it is unrealistic to assume that any single system will be able to address them all. A potential solution to this problem is to bundle individual frameworks, preferably frameworks which concentrate on complementary forms of QoE manipulation. In collaboration with colleagues from Ghent University, a two-tier platform has been developed in which the NIProxy is combined with overlay routing technology. The platform leverages the NIProxy's traffic engineering features to optimize last mile content dissemination, whereas the routing functionality is exploited to achieve resilience to infrastructural issues in the core of the network such as malfunctioning or congested network links. Representative experimental results have confirmed that near end-to-end QoE optimization is realized since only the upstream access segment of an end-to-end network route (if present) is left uncovered by the integrated platform.

14.2 Directions for Future Research

A number of possible topics of future research has already been suggested in the course of this dissertation. These (mostly) correspond to relatively modest technical adjustments which embellish and extend already available functionality instead of radically changing the NIProxy's methodology and general mode of operation. The NIProxy's FEC support could for instance be improved by incorporating techniques other than XOR-based parity protection (e.g., RS coding) as well as by introducing more powerful and effective JSCC algorithms. Another example is scalability investigation. Submitting the NIProxy to a profiling study is likely to yield valuable information regarding the overhead of various system components and might reveal the upper bound of the number of users that can be simultaneously supported by a single NIProxy instance. Next, although the NIProxy supports the collection of information regarding terminal capabilities and user preferences by means of the MPEG-21 UED standard, the added value and potential of this type of context with regard to QoE optimization has thus far been exclusively investigated in isolated test environments. Considering and exploiting this context category in realistic, non-dedicated settings has yet to be done. Finally, the implementation of additional types of internal nodes for the stream hierarchy is expected to extend the NIProxy's bandwidth brokering options, whereas the development of additional services would improve the NIProxy's versatility.

The just described adjustments would require only relatively limited modifications and could hence be achieved in the short term. There is however also room for more drastic (and therefore longer-term) technological improvements and extensions. For instance, the NIProxy could be exploited to develop a *hierarchical* or *layered* solution for QoE optimization. The idea is to construct an overlay network in which the constituting NIProxy instances are not just peers of each other, but instead are hierarchically structured according to parent/child relationships. In the envisioned solution, the NIProxy nodes in the undermost echelon manage end-users and their network traffic and hence assume the role and perform the tasks that have been described in this dissertation; in contrast, nodes higher up the hierarchy govern a number of NIProxy elements from the layer directly underneath. Such a layered approach is expected to open up additional possibilities in terms of QoE research. Higher layers could, for instance, pre-process popular content to facilitate and accelerate QoE operations as the content fans out through the hierarchy. Example issues that will need to be addressed in a hierarchical setup include the determination of optimal content caching locations and the fine-tuning of the operations that are performed by higher layers. Assessment of the optimal distribution of QoE features among the different echelons (i.e., customization of the functionality on a per level basis) is also highly advocated.

QoE management in wireless communication networks and pervasive environments is another example of a technically-oriented challenge that would involve relatively drastic changes and extensions. Thus far, the NIProxy has primarily been applied in wired settings and this thesis has proven that it is a valuable solution for QoE optimization in such a context. There however exist significant differences between wired and wireless environments in terms of characteristics and performance. In addition, wireless scenarios introduce a number of issues that are not (or at least less pronouncedly) present in their wired equivalents. Consequently, straightforwardly leveraging the NIProxy in a wireless context is expected to yield suboptimal results at best. Via the inclusion of additional functionality, this inadequacy might however be remedied. For instance, a common challenge in wireless environments is user and node mobility. To enable the NIProxy to efficiently deal with this issue, it might be necessary to accompany its pure QoE optimization functionality with explicit session mobility support and handover technology. As another example, the inherent broadcast nature of wireless channels is likely to necessitate a number of modifications to the NIProxy's current NTS scheme. Also, the wireless environment might include (large numbers of) sensors that capture environmental data. It might therefore be interesting to include support in the NIProxy to transform such sensor input into a fourth type of contextual information (besides network-, application- and user-related context) so that it could also be considered during QoE mediation. Other examples of characteristics of wireless and pervasive environments that are likely to require special attention include the high level of heterogeneity in wireless access network technology and input/output capability variation.

A final potential direction for technical future research in the long term is *mobile QoE management*. In all the experimental evaluations that have been presented in this dissertation, the NIProxy was deployed at a certain location inside the transportation network and remained stationary from that point on. Introducing mobility into this equation is expected to unlock a plethora of additional research options. In particular, it is envisioned that QoE mediation is executed on mobile hardware such as laptops or powerful smartphones. As the mobile node roams around, it performs QoE arbitration in each of the environments in which it ends up by allowing devices to pair up with it. The most basic type of service that could be delivered by a mobile QoE manager is connectivity provision for devices which are not compliant with the wireless technology that is employed in their present environment. As an example, Bluetooth-equipped cellular phones without a 802.11 radio could be provided WiFi connectivity by a mobile arbitrator that supports both networking standards. It will be interesting to scrutinize whether other, less straightforward mobile QoE operations also make sense. The mobile manager could, for instance, perform the important optimization task of adapting and fine-tuning content according to contextual knowledge prior to its delivery to the requesting device. Challenges and issues regarding mobile QoE management that are worthy of investigation include service discovery, connectivity bridging, potential incentives and revenue models for mobile managers, energy consumption optimization for the mobile arbitrator and the transfer of QoE arbitration state between (mobile) devices. The mobile QoE topic could also be linked to the idea of hierarchical QoE management. As an example, resource-intensive adaptations and operations could be implemented in (fixed) nodes that are deployed inside the (wired) access network, while more trivial and last-minute adaptations would be left over to mobile arbitrators in the wireless environment.

Finally, besides technical topics of future research in both the short and long term, there also lie possibilities in terms of evaluation and validation. All experimental results that have been presented in part II were captured objectively, typically by tracing the traffic that traversed the transportation network, and were subsequently investigated analytically to infer conclusions regarding expected improvements in user experience. QoE is however a multidimensional and inter-disciplinary concept which, besides technically-oriented and objective aspects, for a large part involves human dimensions and subjective remarks. It might therefore be very valuable to subject the QoE optimization attempts that are implemented by the NIProxy to formal user studies and other usability research methods so that qualitative feedback regarding their impact could be collected. Stated differently, although the results that have been presented in this dissertation were of such a magnitude that their potential positive impact on QoE was often intuitively apparent, confirmation by means of qualitative user input is still advocated. On a related note, it is again repeated that the NIProxy is a *framework* for QoE optimization, not an absolute solution for it. The NIProxy merely provides a set of tools via which the QoE of users of distributed applications *may* be improved. This implies that simply integrating the NIProxy in a transportation network is by no means a guarantee for success; after incorporation, its actual operation needs to be fine-tuned to the characteristics and requirements of the considered application, user expectations and potentially a myriad of other contextual parameters and this will typically require user feedback.

Appendices

Appendix A

Early Results

This appendix bundles experimental results that stem from the inceptive phase of this PhD research. In particular, all results that will be presented in this appendix have been generated using the initial instantiation of the NIProxy, which was rendered obsolete by the refactoring operation that was described in chapter 6. As a result, the achievements themselves have also become somewhat outdated since they are no longer representative of the NIProxy's current capabilities and mode of operation. They will therefore be mentioned and discussed only superficially. For the same reason, implementational as well as other details will largely be omitted from the discussion. For each of the described results, the interested reader can find detailed information and necessary technicalities in the articles in which the results were originally published.

A.1 Experimentation in the ALVIC Framework

The netfilter-based implementation of the NIProxy (see section 6.1) has chiefly been evaluated in the context of the ALVIC framework. This section will commence by introducing this framework and by concisely discussing the NIProxy's incorporation in it. Next, a chronological overview of the conducted experimental validations and their outcomes will be presented. It will appear that the complexity of the evaluations became increasingly advanced over time due to the evolution of the NIProxy's QoE optimization potential.

A.1.1 ALVIC

ALVIC (Architecture for Large-scale Virtual Interactive Communities) is an in-house developed framework for the realization of Networked Virtual Environments (NVEs) and in particular Virtual Interactive Communities (VICs) [Quax 07]. Like standard NVEs, a VIC allows potentially large numbers of users to simultaneously participate in a shared interactive (3D) virtual environment. VICs hereby however specifically focus on social and community-related aspects. The high-level objective of a VIC can hence be expressed as providing geographically dispersed individuals who share common interests with a virtual platform where they can meet, converse, socialize, etcetera. Probably the most popular commercial VIC is Linden Lab's Second Life [Second Life 10].

Given its emphasis on virtual community creation, ALVIC has invested considerably in user communication facilities. Besides the typically supported options of textual chat and verbal communication, the framework encompasses extensive video conferencing support [Quax 03]. Instead of separating the video conferencing functionality from the virtual world experience, an integrated and more immersive approach is adopted which allows participants to be represented in the simulated environment in real-time by the video input that is captured by their webcam device (see Figure A.1).

In addition to extensive video conferencing support, the ALVIC architecture is characterized by its strive for scalability, both in terms of number of simultaneous users and spatial extent of the offered virtual environment. To achieve this objective, responsibilities and supporting functionality are devolved from the server infrastructure to individual clients as much as possible. Furthermore, the framework encompasses an advanced multicast-based awareness management module to dam in information dissemination. The awareness manager operates by spatially dividing the virtual environment into adjoining square regions. Each region has a well-defined multicast address associated with it that serves as communication channel for the events that occur in that region. As an example, to communicate their state updates to others, clients simply transmit them to the multicast group of the region in which they are currently residing. The awareness manager at each client constantly determines which regions the local user should be aware of and only the multicast

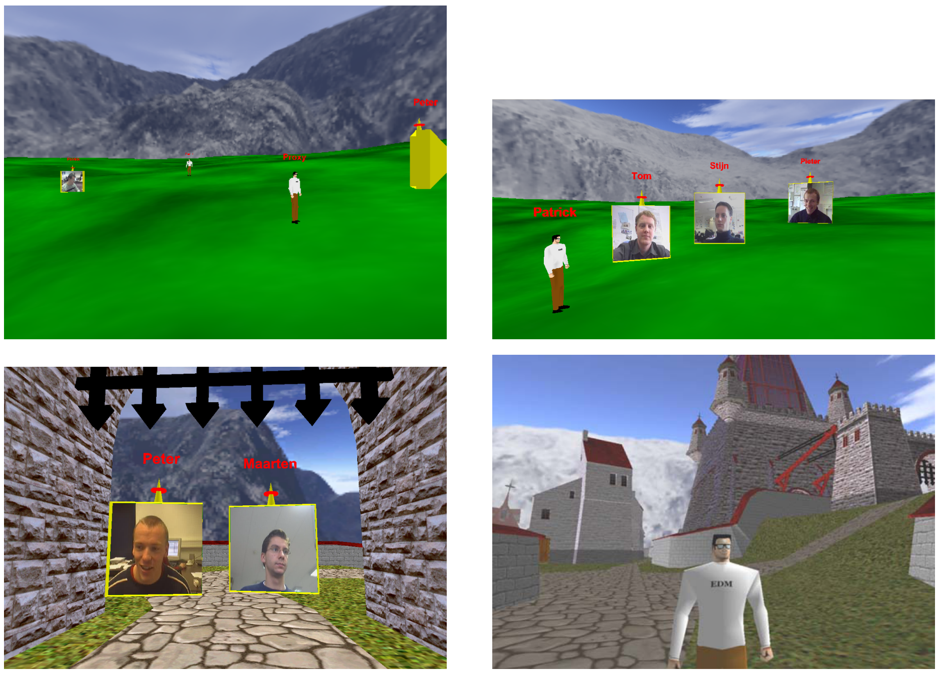


Figure A.1: Screenshots of applications that have been realized on the PC platform using the ALVIC framework. Notice that ALVIC's video conferencing support enables users' faces, recorded by their webcam, to be textured on their avatar in the virtual world.

addresses that correspond to these regions are joined. In other words, as users move around the virtual environment, multicast groups will be dynamically joined and left, this way restricting the amount of information that clients will be required to download and process.

The ALVIC architecture recognizes the considerable bandwidth requirements that are imposed on the transportation network by video-based communication. To prevent these requirements from undermining its scalability objective, the framework defines three separate video qualities which consume widely divergent amounts of bandwidth. The exact quality parameters and the bitrate characteristics of each video version are enumerated in Table A.1[1]. As can be deduced from this table, the High Quality (HQ) video setting re-

[1]It is worth noting that the encoding parameters and bandwidth requirements of the different video versions are completely configurable in the ALVIC framework. The majority of the experimental results that will be presented later on in this section were however generated with the settings presented in Table A.1.

Table A.1: ALVIC's multiple video qualities.

	High Quality	**Medium Quality**	**Low Quality**
Resolution (pixels)	352×288	352×288	352×288
Framerate (FPS)	25	15	15
Bitrate (bps)	200000	100000	50000
Codec	H.263	H.263	H.263

quires double the amount of bandwidth compared to the Medium Quality (MQ) setting, which in turn consumes twice the bandwidth of Low Quality (LQ) video.

Complementary to the multicast group for exchanging event information, the ALVIC architecture attaches three multicast addresses to each spatial region for the purpose of video data distribution. Each of these multicast groups corresponds to a particular video quality setting (i.e., HQ, MQ and LQ). Analogous to the way their state information is disseminated, video-based clients transmit the different encodings of their webcam feed to respectively the HQ, MQ and LQ video multicast groups of their current region. To limit the reception of (bandwidth-intensive) video traffic at client-side, the awareness manager determines which video quality should be received from every region which the local user is currently interested in, after which the corresponding video multicast addresses are subscribed to. A likely strategy would be to enter the HQ multicast group of the virtual region in which the local user is presently located, and the MQ or even LQ multicast group of adjacent regions. Quax et al. have investigated several strategies for video quality selection in [Quax 04]; especially frustum-dependent solutions appear to scale well to growing user figures.

Finally, audio/voice communication is implemented through yet another set of multicast addresses. In contrast to video, the ALVIC framework does not support multiple audio qualities. This design decision is grounded on the fact that audio streams containing voice data will typically require considerably less bandwidth than video. As a result, there is only a single audio-related multicast group associated with each spatial region and clients either receive an audio stream that is emitted by another client or they do not receive the stream at all.

A.1.2 NIProxy Integration

To allow ALVIC users to benefit from the NIProxy's QoE optimization potential, the ALVIC client software had to be modified. The required modification effort however turned out to be fairly modest, as it sufficed to integrate the NILayer auxiliary library (see section 3.4) in the client software and to interface it with ALVIC's client-side awareness management module. The NILayer was instructed to periodically extract application-dependent context from the awareness manager and to subsequently relay the accumulated knowledge to the managing NIProxy instance. In this particular case, the application-related knowledge took the form of information regarding the spatial regions in which the monitored user was at present interested (e.g., the multicast groups that corresponded to these regions). Additionally, the NILayer dynamically registered the virtual distance between the local user and each remote peer that virtually resided in any of these regions. The calculated positional information was employed to inform the NIProxy of the relevance of the individual multimedia flows that were being exchanged as part of the NVE application. The rationale was to attach flow importance directly proportionally to user proximity in the virtual environment, since a VIC user is most likely to interact with remote participants whom he is virtually located nearest to. Consequently, if the objective is to achieve efficient and convincing interactions, multimedia streams originating from virtually proximal peers should be received by the managed user at a maximal fidelity (without hereby violating the user's current downstream bandwidth constraints).

Besides the NILayer incorporation in the client software, a service for the NIProxy was implemented that enabled it to act as a *multicast reflector* for managed ALVIC clients. At the time of integration, multicast-based communication was often disallowed for residential users. Moreover, even to date, several Internet Service Providers (ISPs) obstruct the dissemination of multicast traffic on their access network. To prevent users from being denied access to ALVIC-based distributed applications, the service therefore eliminated the need for multicast facilities at client-side as follows:

- Instead of multicasting a packet themselves, clients could unicast the packet to their NIProxy instance, which would subsequently exploit its application awareness to transmit the packet to the appropriate multicast group on the wide area network

- The service dynamically subscribed to multicast addresses on behalf of the NIProxy's clients and converted multicast traffic that resided on the inter-proxy network to unicast packets to enable their last mile delivery; the service guaranteed that information contained in a multicast stream only reached clients that were actually interested in it, this way preventing downstream access bandwidth of uninterested clients from being wasted

A.1.3 Video Quality Selection Through Filtering

The very first practical assessment of the NIProxy's user QoE improvement potential concentrated exclusively on the filtering of video data. The focus on video was motivated by the obvious bandwidth predominance of this type of network traffic in ALVIC-based distributed applications. The findings and conclusions of this initial experimental evaluation have been published in [Wijnants 05b]. In summary, the assessment encompassed two separate experiments which were both repeated twice, once without and once with involving the NIProxy. In the first experiment, the downstream access bandwidth of an ALVIC client was theoretically varied over time in a static[2] scenario involving 4 remote participants. The results, expressed as network traces, are recapitulated in Figure A.2. In particular, the topmost graph plots the complete set of video traffic that was exchanged over the network during the experiment; notice the HQ, MQ and LQ variants of each remote participant's webcam feed. The next two graphs on the other hand illustrate the video-related data that was received by the monitored client when it was not and when it was managed by the NIProxy, respectively. The second experiment semantically emulated a dynamic conversation between a variable number of participants, which is a regularly recurring scenario in VIC sessions. In contrast to the first test, the traced client's downstream bandwidth capacity remained fixed at 320 Kilobits per second (Kbps) throughout the entire case study. Again, a promiscuous capture of the disseminated video traffic was performed and the video-related data delivery to the monitored client was registered when the NIProxy was respectively excluded from and included in the experiment. The results are plotted in Figure A.3.

The findings that were deduced from the conducted experiments are briefly recited in the following enumeration:

- Comparison of the first two network traces in Figure A.2 as well as in Figure A.3 reveals that ALVIC's awareness management module succeeded in drastically reducing the amount of video traffic that needed

[2]The term "static" refers to the fact that all involved clients remained stationary in the virtual world for the entire duration of the test; consequently, the relative importance of their video feed also remained constant over time.

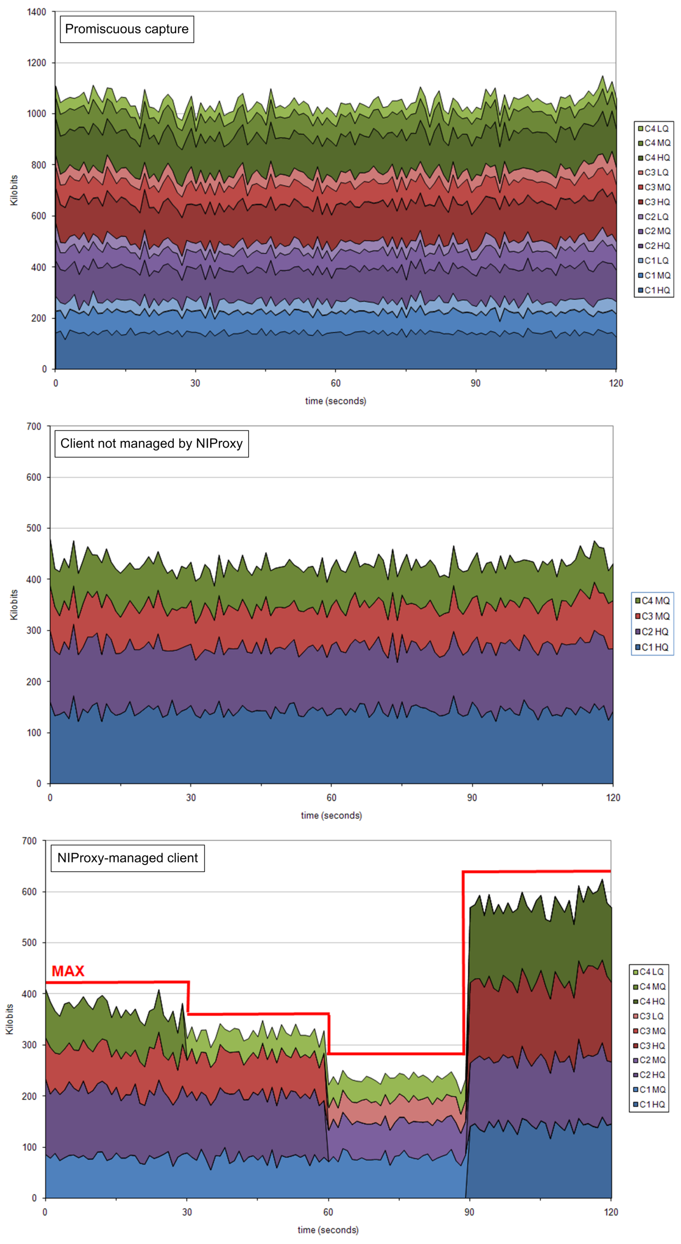


Figure A.2: Stacked graphs of the network traffic that was captured during an ALVIC video filtering experiment in which a client's downstream access bandwidth was altered over time.

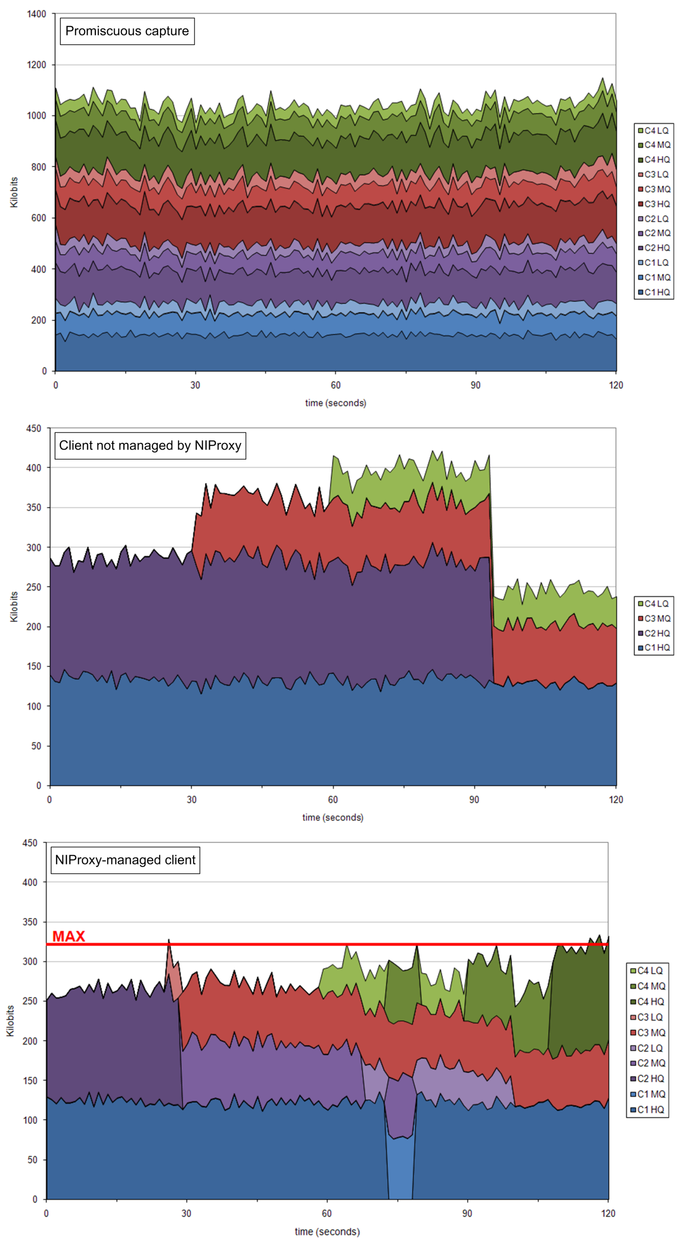


Figure A.3: Stacked graphs of the network traffic that was captured during an ALVIC video filtering experiment which simulated an interactive conversation between VIC participants.

to be delivered over the last mile network connection of the monitored client

- ALVIC's awareness manager lacks network awareness and consequently does not take the capacity, let alone the current condition, of the client's access link into account when deciding which regions the client should subscribe to; as a result, the theoretical bound of its downstream access bandwidth was exceeded numerous times in the iterations of the experiments in which the NIProxy was not involved (see the middle graphs in Figures A.2 and A.3)
- The client-side awareness manager is completely region-based and therefore relatively coarse-grained: run-time adjustment of the delivered quality of individual video feeds is not supported; instead, the awareness arbitrator is only flexible enough to modify the quality of the video dissemination on a per region basis (i.e., of entire regions and hence of all video sources that are virtually residing in it)
- The NIProxy leveraged its network awareness to guarantee that the managed client's downstream access bandwidth capacity was honored (see the final graphs in Figures A.2 and A.3)
- Its application awareness allowed the NIProxy to control the last mile video delivery in a fine-grained (i.e., per flow) and intelligent manner.

In summary, the network traces confirm that the NIProxy prevented the monitored client's access link from becoming congested and dynamically adjusted the distribution of the downstream bandwidth capacity over the involved video streams, based on their altering relative significance, such that the user's QoE was improved. Recall from section A.1.2 that video traffic importance was determined exclusively on the basis of virtual proximity.

A.1.4 Static Video Transcoding

A considerable drawback of the multi-quality video philosophy of the ALVIC architecture is that it compels sources to transmit three distinct versions of their video data. This is a computationally burdensome requirement for the end-user device, since the source's video feed will need to be encoded thrice. More importantly, this approach also demands a large amount of client upstream bandwidth. The majority of the broadband Internet subscriptions for residential users are based on an asymmetric model in which the upstream capacity consists of merely a fraction of the provided downstream bandwidth.

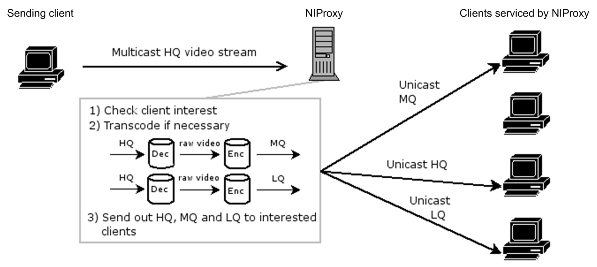

Sending client
NIProxy
Clients serviced by NIProxy
Multicast HQ video stream
1) Check client interest
2) Transcode if necessary
HQ
Dec
raw video
Enc
MQ
HQ
Dec
raw video
Enc
LQ
3) Send out HQ, MQ and LQ to interested clients
Unicast MQ
Unicast HQ
Unicast LQ

Figure A.4: Operation of the application-aware static video transcoding service for ALVIC-based distributed applications.

Typical values at the time of evaluation were an upstream throughput of 128 or 192 Kbps versus at least 1 Mbps in the downstream direction. This impediment refrained the ALVIC framework from being leveraged to develop distributed applications that are targeted at residential users since the collective bandwidth requirement of the three video qualities exceeded the available upload capacity. Even nowadays, despite the fact that the upstream throughput for residential Internet users has risen notably, many ISPs still enforce restrictions in terms of the volume of data that their customers are allowed to upload. This implies that while ALVIC's multi-quality video approach might currently be technologically feasible, obligating residential users to emit separate qualities of their webcam feed might very well still not be an affordable requirement for commercial applications.

This consideration formed the incentive to include application-specific video transcoding functionality in the NIProxy [Wijnants 05c]. In particular, a static video transcoding service was developed that was tailored particularly to the ALVIC framework and its distinctive video requirements. As is illustrated in Figure A.4, the service was implemented so that HQ-encoded ALVIC video traffic was intercepted and transcoded to medium and/or low quality counterparts in case at least one of the NIProxy's connected clients was interested in it, after which each managed client was served the appropriate video fidelity. The availability of this service hence relieved ALVIC video sources of the necessity of streaming multiple versions of their webcam input since it now sufficed for them to transmit only the highest quality variant of their video feed. Also notice from this discussion that, as was enunciated in section 6.1, services in the netfilter-based NIProxy implementation corresponded to global entities which serviced all clients simultaneously.

The impact of the ALVIC video transcoding service was evaluated experimentally. The experiment involved 6 ALVIC clients in total; the following

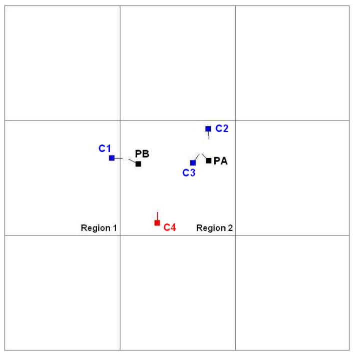


Figure A.5: Client dispersion in the virtual world during the ALVIC video transcoding experiment.

conditions applied:

- Clients C1 to C4 were not managed by a NIProxy instance, whereas clients PA and PB were
- Only clients C1, C2 and C3 acted as video source
- All clients remained stationary in the virtual environment throughout the complete experiment
- The clients were virtually positioned as depicted in Figure A.5
- The downstream access bandwidth of clients PA and PB was artificially modified at intervals of approximately 30 seconds (i.e., from 320 Kbps initially over 250 and 170 Kbps to the final value of 400 Kbps)

The experimental results are visualized in Figure A.6 in the form of network traces. Unsurprisingly, the results largely resemble the outcome of the experiment with variable client downstream bandwidth that has been described in section A.1.3. The findings and conclusions that can be drawn from the achieved results are consequently also nearly identical and will therefore not be explicitly repeated here. Important to realize however is the considerable methodological style breach with the previous video filtering approach. In particular, video quality selection and dissemination was in section A.1.3 controlled by the NIProxy by simply filtering the optimal quality variant of each video feed or, stated differently, by obstructing the last mile dissemination of inappropriate video versions. In contrast, the video transcoding approach encompassed the on-demand and on-the-fly generation of required video qualities by the NIProxy, hereby offloading this responsibility from ALVIC clients.

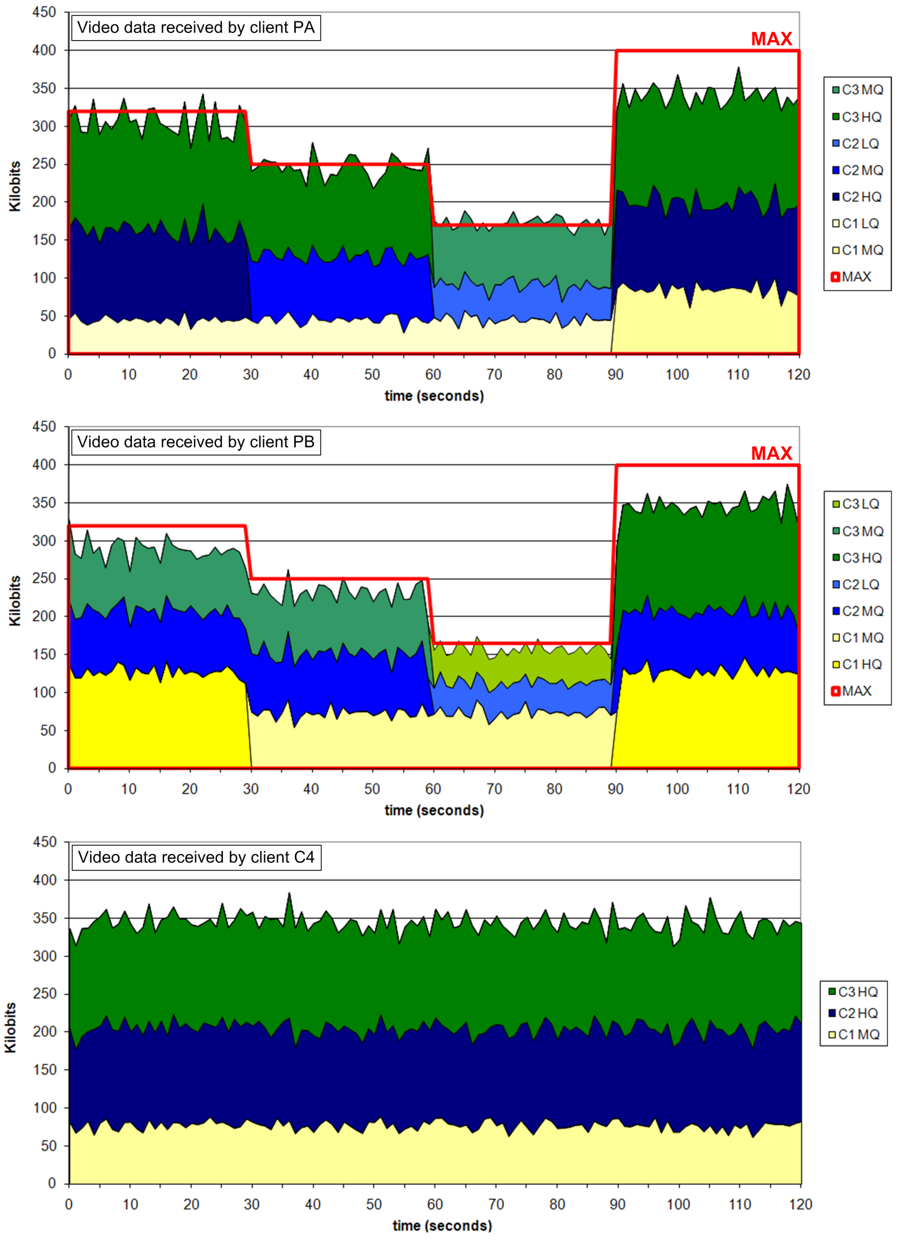


Figure A.6: Stacked graphs of the network traffic that was captured during an ALVIC video transcoding experiment in which the downstream access bandwidth was varied over time.

The principal contribution of the results presented in this section is that they formed the first practical evidence of the added value of the NIProxy's service provision support. In particular, they corroborated the possibility of leveraging the NIProxy as a platform for the delivery of potentially application-specific services. As a result, the versatility of the NIProxy and its viability as QoE optimization framework for heterogeneous types of distributed applications was for the first time experimentally confirmed. Moreover, the experience with real-time video transcoding that was acquired in this preliminary study and the determination of its beneficial implications in terms of QoE optimization options led to a continual interest in this topic throughout the course of this PhD research. In other words, the video transcoding service for the ALVIC framework motivated the development of the more generally applicable static H.263 video transcoding service from section 5.3 and its dynamic H.264/AVC counterpart that has been presented in chapter 13.

A.1.5 Mobile Access

While the ALVIC framework was initially designed exclusively for the PC platform, it was at a later stage ported to handheld devices. The objective of this effort was to achieve universal (i.e., anyplace, anywhere) access to ALVIC-based distributed applications while at the same time preserving the framework's extensive support for fixed users. Before actually initiating the porting operation however, the networking issues that would need to be resolved when simultaneously including wired and wireless devices in NVE sessions were theoretically investigated [Quax 05b]. In this preparatory study, it was established that in-network traffic engineering entities such as the NIProxy could play a crucial role in enabling mobile access to NVE applications over truly large distances. In the specific context of the ALVIC architecture, a network setup like the one depicted in Figure A.7 was envisaged. As can be seen, the NIProxy would in the envisioned topology be deployed at the periphery of the multicast-enabled Local Area Network (through which fixed users could connect to the VIC), where it would provide a number of specific services to support mobile users. In particular, two major tasks were identified for the NIProxy instance in the proposed setup:

Act as multicast-to-unicast gateway for mobile clients As there is very little multicast support in the Internet at present, traffic originating from the multicast-enabled LAN would need to be forwarded towards remote mobile clients through unicast connections. However, simply duplicating all multicast network traffic that resides on the LAN would not scale well with a growing number of connected users, neither would this approach

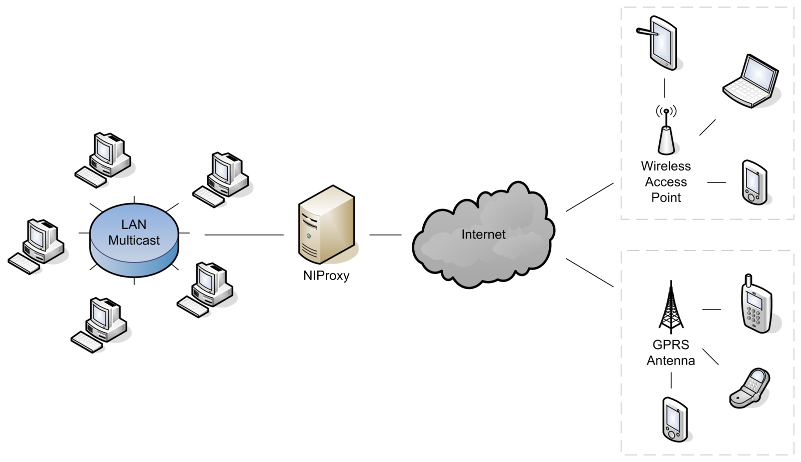
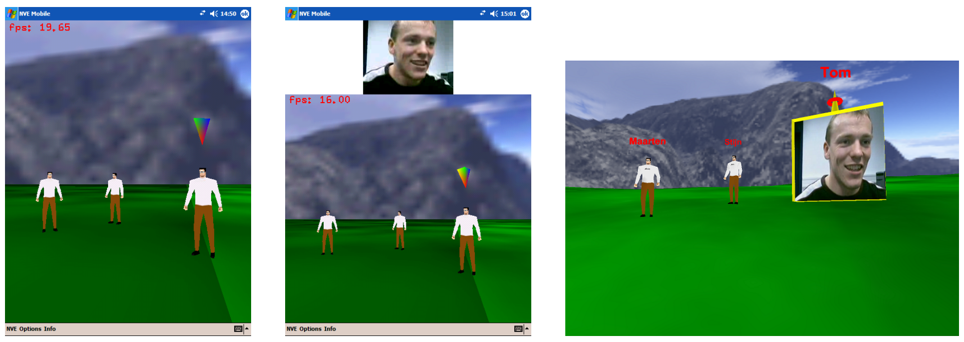


Figure A.7: Proposed network architecture for long-range mobile access to ALVIC-based applications.

be suited for low-bandwidth wireless technologies such as GPRS. For these reasons, it was determined that the NIProxy would need to leverage its compound network- and application-related contextual knowledge to intelligently select the multicast traffic that should be unicasted to each individual remote client. Conversely, the NIProxy would also be charged with the transmission of data, received from mobile clients in a unicast fashion, to interested wired clients by disseminating it on the correct multicast group.

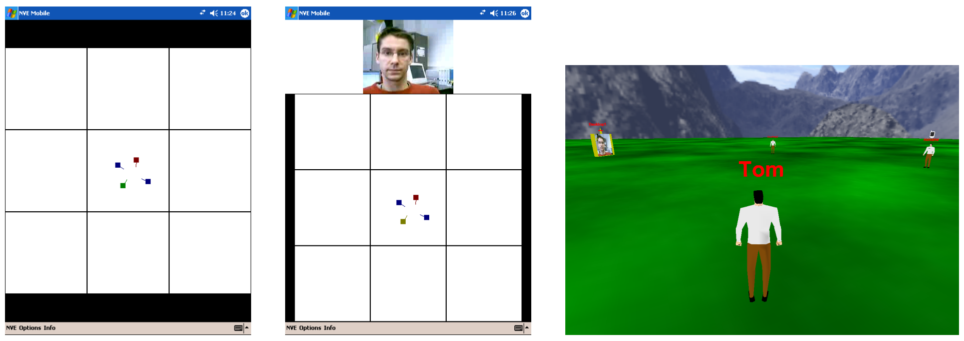
Transcode audio and video traffic emitted by fixed users Wired LAN clients will most likely be exchanging high-quality audio and video data whose bandwidth requirements by far exceed the available throughput of a long-range wireless link. The NIProxy would therefore be responsible for at run-time transcoding multimedia flows to lower-quality variants which better fit the bandwidth limitations imposed by wireless connections. Again, the NIProxy's network and application awareness were envisioned to be indispensable assets in the execution of this task. In particular, its contextual knowledge would enable the NIProxy to decide which multimedia streams should be transcoded and to what quality exactly.

Based on the findings of the exploratory study, the ALVIC framework was subsequently actually adapted for use on handheld terminals with wireless network interfaces [Quax 05a]. Some results are visualized in Figures A.8 and



(a) Without video visualization (b) With video visualization (c) Corresponding 3D view on a desktop client

Figure A.8: Screenshots of an example ALVIC-based NVE application with support for both fixed and mobile users (3D rendering on the mobile terminal).



(a) Without video visualization (b) With video visualization (c) Corresponding 3D view on a desktop client

Figure A.9: Screenshots of an example ALVIC-based NVE application with support for both fixed and mobile users (2D rendering on the mobile terminal).

A.9. As these screenshots reveal, the mobile instantiation of the ALVIC framework supports both 2D and 3D virtual world rendering. In case their terminal is sufficiently powerful (e.g., is equipped with 3D hardware acceleration), mobile users are provided with an immersive three-dimensional visualization that is largely comparable to the ALVIC virtual world representation on the PC platform. Conversely, on mobile devices which lack the necessary processing power to display 3D graphics at interactive framerates, only a top-down two-dimensional overview of the shared virtual environment is displayed. Pictures of example ALVIC multi-platform setups are provided in Figure A.10.

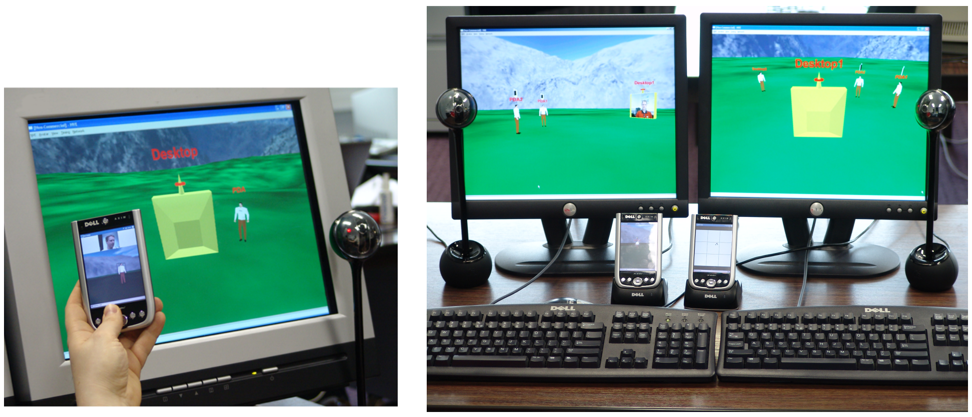


Figure A.10: Example ALVIC multi-platform setups including standard desktop clients as well as mobile Dell Axim x51v PDA devices.

Figures A.8 and A.9 illustrate that the NIProxy successfully facilitated video communication between wired and wireless users of the example ALVIC-based application. Due to the small form factor of handheld devices, simultaneously displaying multiple video feeds on such terminals will often be infeasible. For the developed demonstrator application, this issue was further aggravated by the fact that the available screen space on the mobile device could not exclusively be employed for video visualization purposes (i.e., it needed to be divided among the representation of the virtual environment and the visualization of received video traffic). Based on this observation, which can be considered as a form of application awareness, the NIProxy enforced the policy that at all times at most a single video feed should be forwarded to mobile clients of the demonstrator. Moreover, prior to its delivery, video traffic that was destined for mobile users was on-the-fly converted by the NIProxy to a lower spatial as well as temporal resolution. At the same time, the video traffic's bitrate was adjusted so that it matched the throughput constraints of the mobile device's wireless network connection (e.g., to accommodate mobile users with a low-bandwidth GPRS interface).

In summary, by drawing from its contextual knowledge, the NIProxy succeeded in providing mobile users of the multi-platform ALVIC demonstrator with an enjoyable video conferencing experience. This was achieved by harmonizing the video traffic that was destined for these users with the characteristics of their mobile device as well as their network connection type.

A.1.6 Audio Filtering

Video traffic typically dominates voice data in terms of network resource requirements. This explains the focus on this type of network traffic during the initial NIProxy evaluations. At a later stage however, ALVIC's audio communication facilities and their associated bandwidth requirements were also reckoned with. In particular, the experimental findings presented in [Wijnants 05a] corroborate that the NIProxy succeeded in simultaneously managing the audio and video traffic that is induced by the ALVIC framework. While definitely being a significant achievement at that time, this result has been rendered obsolete by more recent NIProxy validations. Therefore, the audio filtering evaluation and its outcome will not be explicitly repeated in this dissertation; the interested reader is kindly referred to the article in which the results were originally published.

A.1.7 Audio Mixing

Section A.1.5 has discussed the porting of the ALVIC framework to handheld devices. Compared to the desktop platform, such devices are much more constrained in terms of processing power. In addition, the wireless interfaces through which they connect to the network are characterized by a relatively limited throughput. Therefore, some sacrifices had to be made during the porting process. Probably the most drastic design decision was to confine the delivery of audio data to mobile ALVIC clients to exactly one flow. Stated differently, while a desktop user will receive the audio traffic from all remote clients that are currently residing in the regions which he is interested in, a mobile user will only receive the audio data of the object (e.g., avatar) that he has selected on his screen. This restriction is motivated by the high bandwidth and processing requirements that are associated with receiving and decoding multiple audio streams simultaneously. Furthermore, received audio traffic needs to be mixed together locally to obtain the final output signal, which again is a processor-intensive operation.

It is apparent that a substantial difference exists between the audio experience that is provided to respectively desktop and mobile users of ALVIC-based distributed applications. To bridge this divide, however without increasing the bandwidth or processing requirements for mobile terminals, an application-aware sound mixer service for the NIProxy was developed [Wijnants 06]. A high-level overview of the operation of this service is illustrated in Figure A.11. As can be seen, the sound mixer plug-in maintained a separate mixing unit for every serviced client and subscribed to the correct multicast groups to ensure that it received all necessary input audio data. Each incoming audio

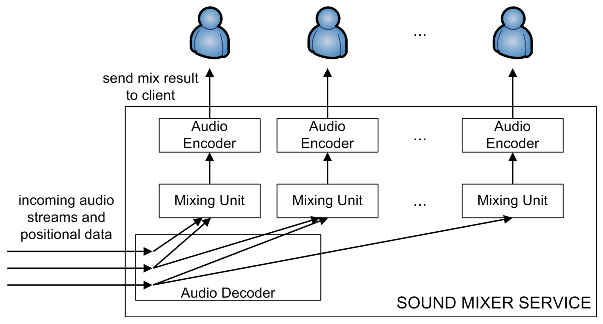


Figure A.11: Design and operation of the application-aware sound mixer service for ALVIC-based distributed applications.

flow was decoded and subsequently only transferred to the mixing units for those clients that were actually interested in this stream. This implies that a unique audio mix was created for every client that was serviced by the plug-in. Furthermore, the mixing units also took positional information into account, this way producing an output stream in which the contributing audio sources were correctly localized in 3D space. Notice that this discussion of the sound mixer plug-in's operation again illustrates the global (i.e., multi-user) nature of services in the netfilter-based NIProxy software architecture (see section 6.1).

The effect of the sound mixer service is exemplified in Figure A.12. Network graphs A, B and C visualize the audio traffic that served as input for the sound mixer service, while chart D depicts the resulting audio mix. The first input audio flow was generated by a continuous audio source that terminated its transmission approximately 30 seconds after the experiment had started. The two other input audio streams originated from two verbally communicating ALVIC clients and hence transported alternating bursts of voice data.

The results from Figure A.12 demonstrate that the developed NIProxy plug-in enabled lightweight audio communication in the ALVIC framework. As such, the service entailed promising possibilities to improve the audio experience that is provided to mobile users of ALVIC-based applications. More specifically, while previously being confined to receiving the audio traffic of a single remote client, the sound mixer service enabled mobile users to receive a single audio stream that encompassed all audio data in which they were effectively interested. This implies that the sound mixer service successfully mitigated the disparity that existed between the audio experience perceived by respectively desktop and mobile ALVIC users. Furthermore, this

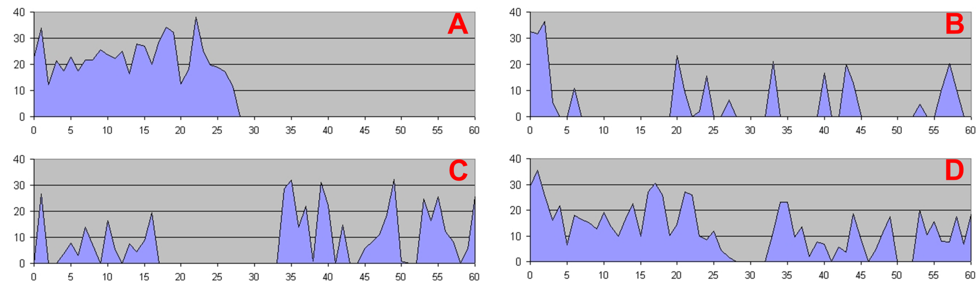


Figure A.12: Impact of the ALVIC sound mixer service on audio traffic dissemination. The horizontal axes of the graphs specify the time (in seconds), the vertical axes the bit count (in Kilobits). Graphs A, B and C depict (the bandwidth consumption of) the input audio traffic, whereas graph D shows the resulting audio mix.

was achieved without introducing additional bandwidth or processing requirements at client-side, since it still sufficed for mobile users to receive and process only a single audio flow. A minor disadvantage of the sound mixer service was that it yielded a small amount of incidental latency since input audio traffic needed to be buffered for a short period of time so that it could be mixed together.

As a concluding remark, notice that although the sound mixer service has been described and explained exclusively from the point of view of mobile users, it could just as well be exploited by ALVIC desktop clients. For this latter user category, the impact of the service was however much less pronounced. In particular, only (modest) decreases in terms of downstream bandwidth consumption and processing requirements were noticed by desktop clients; contrary to the situation with mobile users, the audio experience itself was not improved.

A.2 Incorporation in the iConnect System

To investigate whether the initial implementation of the NIProxy succeeded in concurrently performing QoE optimization for users of various types of distributed applications, it was not only leveraged in the context of the ALVIC framework but also in the iConnect project. This section will first concisely describe the objectives of this project and will subsequently discuss how the NIProxy's multimedia service provision facilities were exploited to enhance the iConnect application with a video-based avatar service.

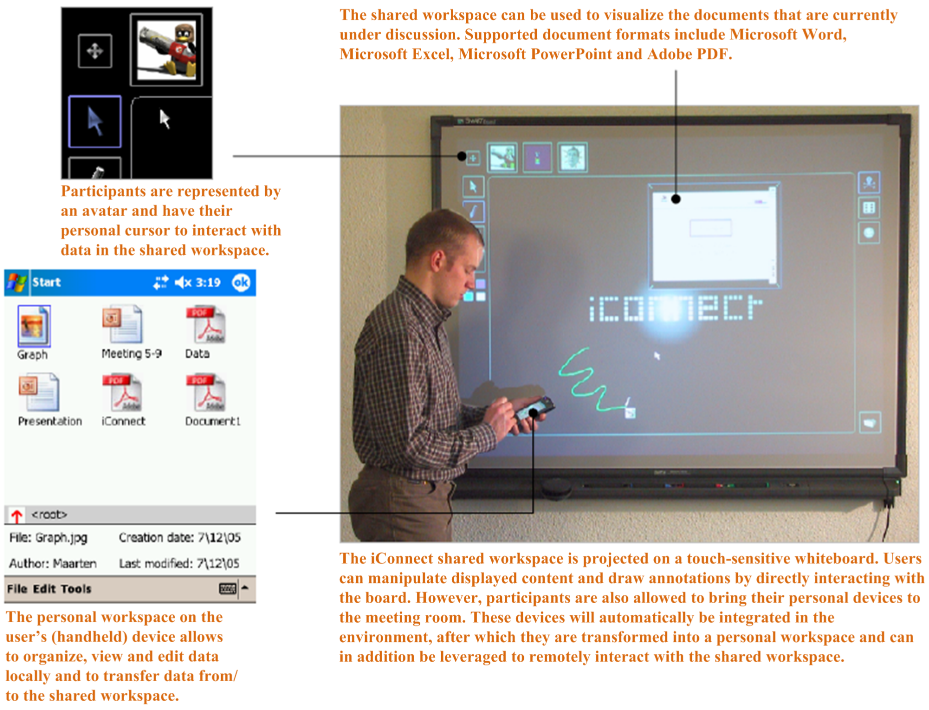
A.2.1 iConnect

Meetings and conferences are nowadays still frequently being held in an inefficient and impractical fashion. For example, the requirement for participants to be physically present at the meeting might lead to large traveling overhead and expenditure. Furthermore, the supporting equipment of a typical meeting room is restricted to a projector which participants can alternately use to project working documents that are stored on their personal laptop computer. Convenient facilities for integrating devices other than laptops (e.g., PDAs, smartphones, . . .) are also largely lacking.

Based on this observation, the Interdisciplinary institute for BroadBand Technology (IBBT) incubated the iConnect project (1/05/2005 - 30/06/2007). The objective of the project consisted of developing an IT architecture and a software framework to support the creation of "connected conference rooms" where both collocated and geographically dispersed participants can efficiently assemble.

The implemented software framework was centered around the semantical discrimination between *shared* and *personal* workspaces [Cardinaels 06]. The rationale behind this separation was flexibility: the shared workspace was intended to allow multiple users to interact simultaneously with meeting-related data, whereas personal workspaces were completely single-user-oriented. The combined use of both workspace types hence permitted participants to interactively and collaboratively manipulate documents in the multi-user shared environment, yet retained the option for users to organize, view and/or manipulate data locally in their personal space without hereby disturbing others.

The iConnect software framework was evaluated in a computer-augmented meeting room that was equipped with a projector, a touch-sensitive whiteboard with a large form factor, a wireless access point, a wired connection to the outside world and supporting server infrastructure which ran the iConnect software. The whiteboard fulfilled the role of shared workspace as it provided a means for physically attending users to simultaneously interact with the data that was displayed on it. Via the wireless network, meeting participants could couple personal devices that they had brought into the meeting room to the iConnect software, after which they could be used as personal workspace. Through an intuitive drag-and-drop interaction metaphor, participants could easily transfer files from their personal device to the shared environment and vice versa. For instance, a participant could copy a document that was currently under discussion and thus displayed on the shared whiteboard to his personal device, alter or annotate it locally, and subsequently transfer the modified document back to the whiteboard to share his changes with the other


The shared workspace can be used to visualize the documents that are currently under discussion. Supported document formats include Microsoft Word, Microsoft Excel, Microsoft PowerPoint and Adobe PDF.

Participants are represented by an avatar and have their personal cursor to interact with data in the shared workspace.

The iConnect shared workspace is projected on a touch-sensitive whiteboard. Users can manipulate displayed content and draw annotations by directly interacting with the board. However, participants are also allowed to bring their personal devices to the meeting room. These devices will automatically be integrated in the environment, after which they are transformed into a personal workspace and can in addition be leveraged to remotely interact with the shared workspace.

The personal workspace on the user's (handheld) device allows to organize, view and edit data locally and to transfer data from/ to the shared workspace.

File: Graph.jpg Creation date: 7\12\05
Author: Maarten Last modified: 7\12\05

Figure A.13: Overview of the iConnect infrastructure and of the functionality of the iConnect software framework.

meeting participants. Besides acting as personal environment, the user's device could also be used as a remote touchpad to operate a cursor in the shared workspace.

The infrastructural setup and functionality of the iConnect system are illustrated in Figure A.13. As can be seen, the shared workspace was projected on the whiteboard, with which users could interact not only directly (i.e., by physically touching it) but also indirectly (i.e., via their personal device). Also note that each meeting participant was represented by a small avatar in the shared workspace.

The infrastructure and software framework also included support for users who were restrained from being physically present in the conference room. Such users could remotely connect to the iConnect system, after which the shared workspace was visualized on the screen of their device (i.e., a laptop or desktop computer). Remote participants had analogous collaboration possibilities at their disposal as the users in the conference room. In addition, audio as well as video conferencing support was provided to allow remote and physically attending participants to communicate in an efficient manner.

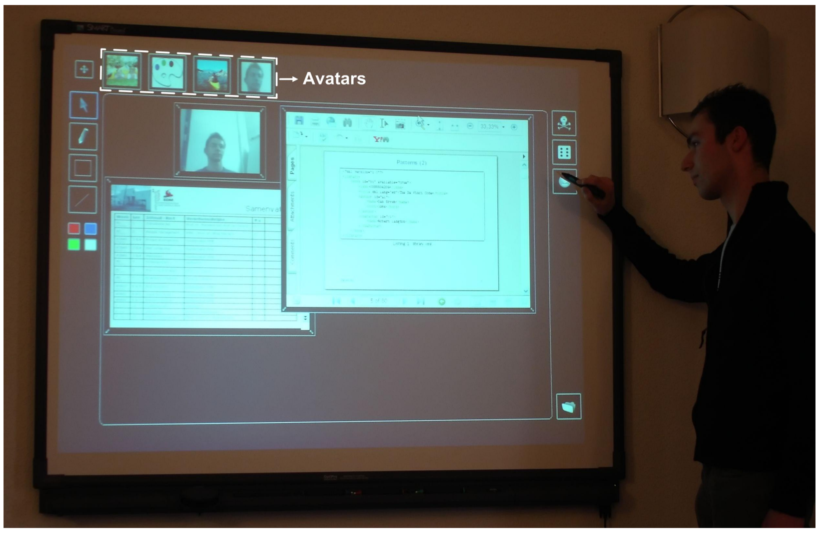
A.2.2 Video-Based Avatar Creation

A NIProxy service was developed to incorporate support for *video-based avatars* in the iConnect system [Wijnants 06]. Recall from section A.2.1 that conference participants were personified by an avatar in the shared workspace. In the iConnect software framework, these avatars took the form of a user-selectable static image. The NIProxy service introduced the possibility to represent remote meeting participants by means of a low-resolution and low-framerate live video feed of their face. In particular, the service intercepted the video conferencing flow emitted by remote participants, decoded it and applied face detection on it. The result was periodically encoded as a small image (i.e., once every second) and subsequently transmitted to the shared whiteboard in the iConnect meeting room.

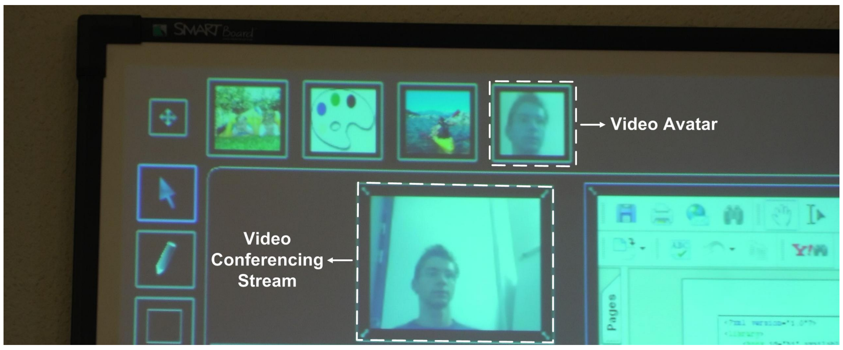
The operation of the video-based avatar service is exemplified in Figure A.14. In this example iConnect meeting, the rightmost avatar represented a remote participant and was video-based. The difference between the remote participant's video conferencing stream and his video avatar, in terms of image quality and resolution as well as image content, is easily deductible from Figure A.14(b). Notice that the (video) avatars were always visualized on the shared whiteboard, while the rendering of a remote participant's video conferencing stream could be turned on and off dynamically by the meeting moderator (i.e., the person standing next to the whiteboard).

The added value of the video-based avatar service was that it in many cases eliminated the need for displaying the video conferencing stream of remote participants on the shared whiteboard. This was mainly true for scenarios in which only limited input was required from this user category. In such situations, visualizing their full-blown video stream added very little benefit, since the feedback provided by the video avatars normally sufficed to guarantee meeting effectiveness. As an example, consider the use case in which the iConnect meeting system is leveraged to stage a presentation or a lecture. The speaker, who is standing next to the shared whiteboard, has direct face-to-face contact with the copresent participants. The video avatars on the other hand also provide the speaker with facial feedback from remote attendees. Consequently, rendering their webcam feed has become superfluous since the information conveyed by their video-based avatar already suffices to allow the speaker to determine whether or not his talk is being understood.

By curtailing the necessity of visualizing the video streams of remote meet-



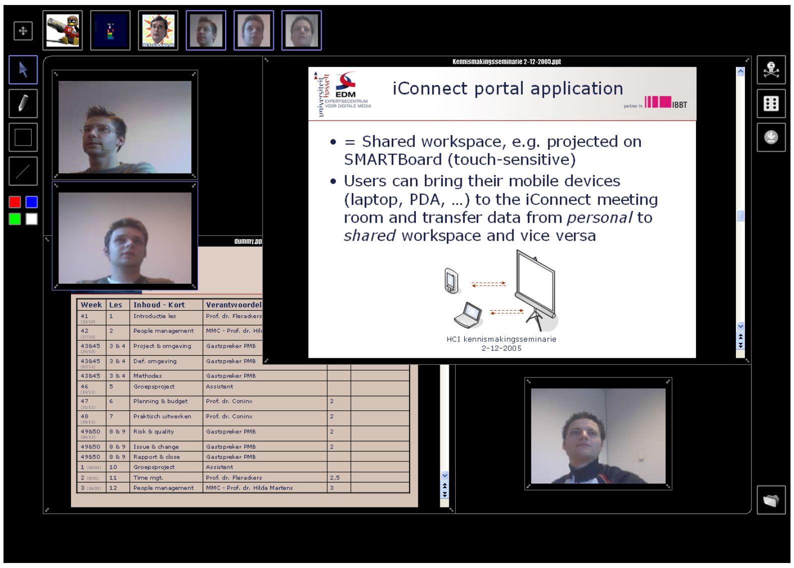
(a) Snapshot of the shared whiteboard during an example iConnect meeting



(b) Comparison of the video conferencing stream of a remote participant and the automatically generated video-based avatar

Figure A.14: The video-based avatar service for the iConnect system.

ing participants, the video avatar service saved screen space on the shared whiteboard. This was an important advantage, since screen space is a scarce commodity. After all, although the whiteboard is blessed with a large form factor, content needed to be displayed on it sufficiently largely so that it could be perceived clearly by all participants in the meeting room. For instance, Figure A.15 illustrates the substantial amount of screen space that was required to visualize the webcam feed of three remote participants. The NIProxy's video-based avatar service was capable of freeing up some of this screen space, which could subsequently be exploited to display other, more relevant information.



iConnect portal application

- = Shared workspace, e.g. projected on SMARTBoard (touch-sensitive)
- Users can bring their mobile devices (laptop, PDA, …) to the iConnect meeting room and transfer data from *personal* to *shared* workspace and vice versa

HCI kennismakingsseminarie
2-12-2005

Week	Les	Inhoud - Kort	Verantwoordel		
41 (10/10)	1	Introductie les	Prof. dr. Fleradkers		
42 (17/10)	2	People management	MMC - Prof. dr. Hil		
43&45 (24/10)	3 & 4	Project & omgeving	Gastspreker PMB		
43&45 (07/11)	3 & 4	Def. omgeving	Gastspreker PMB		
43&45	3 & 4	Methodes	Gastspreker PMB		
46 (14/11)	5	Groepsproject	Assistent		
47 (21/11)	6	Planning & budget	Prof. dr. Coninx	2	
48 (28/11)	7	Praktisch uitwerken	Prof. dr. Coninx	2	
49&50 (05/12)	8 & 9	Risk & quality	Gastspreker PMB	2	
49&50	8 & 9	Issue & change	Gastspreker PMB	2	
49&50	8 & 9	Rapport & close	Gastspreker PMB		
1 (02/01)	10	Groepsproject	Assistent		
2 (9/01)	11	Time mgt.	Prof. dr. Fleradkers	2,5	
3 (16/01)	12	People management	MMC - Prof. dr. Hilda Martens	3	

Figure A.15: Visualizing the webcam feeds of remote participants might consume considerable whiteboard screen space.

It is important to note that the video avatar functionality could also have been implemented in the iConnect client software for remote participants or at server-side. However, one of the design goals for the iConnect project was to keep the client software as lightweight as possible to leave open the possibility of porting it to handheld (i.e., resource-constrained) terminals. This would enable the iConnect framework to address additional user groups like, for instance, people on the move carrying only a smartphone. The iConnect server software on the other hand already bore a large number of responsibilities and it was therefore determined that it should be exempted from the execution of non-critical tasks as much as possible.

Besides being valuable in its own right, the NIProxy's video avatar service was an important achievement because it corroborated the versatility of the NIProxy's multimedia service provision platform. In particular, the NIProxy had thus far been exclusively validated in the context of the ALVIC framework (see section A.1). The iConnect environment in nothing resembles ALVIC-based applications, yet the NIProxy succeeded in providing a valuable service for it. Experiments have even been conducted in which a single NIProxy instance concurrently managed both ALVIC and iConnect clients [Wijnants 06]. The results confirmed that the NIProxy is capable of simultaneously performing QoE optimization for users of multiple distributed applications. This is a significant advantage from an economic perspective, since it implies that the cost of NIProxy deployment can be divided among multiple application providers. Notice that although this feature was demonstrated using the initial (i.e., obsolete) NIProxy implementation, care was taken to ensure that it was retained during the redesign and software refactoring phase.

Appendix B

Example MPEG-21 UED Document

```xml
<?xml version="1.0"?>
<DIA xmlns="urn:mpeg:mpeg21:2003:01-DIA-NS" xmlns:mpeg7="
    urn:mpeg:mpeg7:schema:2001">
  <Description xsi:type="UsageEnvironmentType">

    <!-- Terminal capabilities -->

    <UsageEnvironment xsi:type="TerminalCapabilitiesType">
      <!-- MP3 audio decoding support -->
      <TerminalCapabilities xsi:type="CodecCapabilitiesType">
        <Decoding xsi:type="AudioCapabilitiesType">
          <Format href="urn:mpeg:mpeg7:cs:AudioCodingFormatCS:2001:4.4">
            <mpeg7:Name xml:lang="en">MP3</mpeg7:Name>
          </Format>
        </Decoding>
      </TerminalCapabilities>

      <TerminalCapabilities xsi:type="InputOutputCapabilitiesType">
        <!-- Screen resolution and other display information -->
        <Display bitsPerPixel="24" colorCapable="true">
          <Resolution horizontal="176" vertical="144" />
        </Display>

        <!-- Audio playback capabilities -->
        <AudioOut bitsPerSample="16" numChannels="2" />

        <!-- Input capabilities -->
        <UserInteractionInputSupport>
          <Microphone>true</Microphone>
          <PointingDevice>
            <Mouse buttons="3" scrollwheel="true" />
          </PointingDevice>
        </UserInteractionInputSupport>
      </TerminalCapabilities>

      <!-- Remaining battery lifetime -->
      <TerminalCapabilities xsi:type="DevicePropertyType">
        <Power batteryTimeRemaining="3600" />
      </TerminalCapabilities>
    </UsageEnvironment>

    <!-- Network characteristics -->

    <UsageEnvironment xsi:type="NetworkCharacteristicsType">
      <!-- Static network properties (e.g., minimal throughput) -->
      <NetworkCharacteristics xsi:type="NetworkCapabilityType"
          minGuaranteed="32000" maxCapacity="384000" inSequenceDelivery=
          "false" />

      <!-- Dynamic network-related information (e.g., current delay) -->
      <NetworkCharacteristics xsi:type="NetworkConditionType">
        <Delay packetOneWay="100" delayVariation="40" />
        <Error packetLossRate="0.0003" />
      </NetworkCharacteristics>
    </UsageEnvironment>

    <!-- User characteristics -->

    <UsageEnvironment xsi:type="UserCharacteristicsType">
      <!-- User name -->
      <UserCharacteristics xsi:type="UserInfoType">
        <UserInfo xsi:type="mpeg7:PersonType">
          <mpeg7:Name>
            <mpeg7:GivenName>John</mpeg7:GivenName>
            <mpeg7:FamilyName>Doe</mpeg7:FamilyName>
          </mpeg7:Name>
        </UserInfo>
      </UserCharacteristics>

      <!-- The user prefers sports-related content -->
      <UserCharacteristics xsi:type="ContentPreferencesType">
        <UserPreferences>
          <mpeg7:FilteringAndSearchPreferences>
            <mpeg7:ClassificationPreferences>
              <mpeg7:Genre>
                <mpeg7:Name>Sports</mpeg7:Name>
              </mpeg7:Genre>
            </mpeg7:ClassificationPreferences>
          </mpeg7:FilteringAndSearchPreferences>
        </UserPreferences>
      </UserCharacteristics>

      <!-- Modality conversion preferences: in case a video fragment
          needs to be converted, the user prefers conversion to (lower-
          quality) video, followed by audio and finally text -->
      <UserCharacteristics xsi:type="PresentationPreferencesType">
        <ModalityConversion>
          <GeneralResourceConversions>
            <Conversion order="1" weight="1.0">
              <From href="urn:mpeg:mpeg21:2003:01-DIA-ModalityCS-NS:1">
                <mpeg7:Name>Video</mpeg7:Name>
              </From>

              <To href="urn:mpeg:mpeg21:2003:01-DIA-ModalityCS-NS:1">
                <mpeg7:Name>Video</mpeg7:Name>
              </To>
            </Conversion>

            <Conversion order="2" weight="1.0">
              <From href="urn:mpeg:mpeg21:2003:01-DIA-ModalityCS-NS:1">
                <mpeg7:Name>Video</mpeg7:Name>
              </From>

              <To href="urn:mpeg:mpeg21:2003:01-DIA-ModalityCS-NS:3">
                <mpeg7:Name>Audio</mpeg7:Name>
              </To>
            </Conversion>

            <Conversion order="3" weight="0.1">
              <From href="urn:mpeg:mpeg21:2003:01-DIA-ModalityCS-NS:1">
                <mpeg7:Name>Video</mpeg7:Name>
              </From>

              <To href="urn:mpeg:mpeg21:2003:01-DIA-ModalityCS-NS:4">
                <mpeg7:Name>Text</mpeg7:Name>
              </To>
            </Conversion>
          </GeneralResourceConversions>
        </ModalityConversion>
      </UserCharacteristics>

      <!-- Visual impairment information -->
      <UserCharacteristics xsi:type="AccessibilityCharacteristicsType">
        <Visual>
          <Blindness eyeSide="right" />
        </Visual>
      </UserCharacteristics>
    </UsageEnvironment>

    <!-- Natural environment characteristics -->

    <UsageEnvironment xsi:type="NaturalEnvironmentCharacteristicsType">
      <!-- Illumination characteristics -->
      <NaturalEnvironmentCharacteristic xsi:type="
          IlluminationCharacteristicsType">
        <TypeOfIllumination>
          <ColorTemperature>159</ColorTemperature>
        </TypeOfIllumination>
        <Illuminance>500</Illuminance>
      </NaturalEnvironmentCharacteristic>

      <!-- Background noise level and frequencies -->
      <NaturalEnvironmentCharacteristic xsi:type="AudioEnvironmentType">
        <NoiseLevel>20</NoiseLevel>
        <NoiseFrequencySpectrum>
          40 30 20 10 10 10 10 10 10 10
          10 40 40 40 30 30 30 20 20 20
          10 10 10 10 10 10 10 10 10 10
          10 10 10
        </NoiseFrequencySpectrum>
      </NaturalEnvironmentCharacteristic>
    </UsageEnvironment>

  </Description>
</DIA>
```

Listing B.1: Example MPEG-21 UED profile.

Appendix C

Dutch Summary

Door de alomtegenwoordigheid van het Internet in onze huidige samenleving zou het idee kunnen ontstaan dat het uitwisselen van data tussen computers die verbonden zijn door middel van een netwerk een triviale opdracht is. Dit is echter een misvatting. Het op een efficiënte en effectieve manier uitwisselen van informatie via een computer netwerk en het implementeren van gedistribueerde applicaties zijn opdrachten die verre van triviaal zijn. De reden hiervoor is dat het merendeel van de hedendaagse gedistribueerde applicaties multimediaal van aard is en aldus de uitwisseling van multimediale inhoud tussen geografisch verspreide gebruikers vereist. Hiermee zijn op zijn beurt vaak een aantal performantie voorwaarden verbonden waaraan het computer netwerk moet voldoen opdat de efficiënte of zelfs correcte werking van de applicatie gegarandeerd is. Een interactieve applicatie zoals bijvoorbeeld een Internet telefonie toepassing (Voice over IP) vereist bijvoorbeeld dat de uitgewisselde pakketten (welke in dit geval spraak bevatten) tijdig arriveren bij de bestemming; indien de vertraging die het netwerk introduceert te hoog oploopt, zal de bestemmeling haperingen opmerken in de spraak van zijn gesprekspartner. Merk het verschil op met toepassingen waarvoor het Internet initieel voornamelijk werd gebruikt zoals e-mail, bestandsoverdracht en het opzoeken van (tekstuele) informatie; dergelijke applicaties zijn in veel mindere mate onderhevig aan de performantie van het netwerk.

Moderne gedistribueerde applicaties vragen dus typisch een zeker niveau van dienstverlening van een computer netwerk en dringen het een aantal vereisten betreffende performantie op. De courante onderzoeksterm voor dit fenomeen is *Quality of Service* (QoS). Indien het netwerk er niet in slaagt de QoS vereisten te vervullen, zal de werking van de applicatie normaliter aanzienlijk verslechteren. Dit zal op zijn beurt een negatieve invloed hebben op de tevredenheid van de eindgebruiker. Gebruikers die ontevreden zijn over het gedrag en de performantie van een gedistribueerde toepassing zullen sneller geneigd zijn om de toepassing links te laten liggen en op zoek te gaan naar een beter alternatief. In het geval we hierbij te maken hebben met commerciële software, zal dit gepaard gaan met mogelijk inkomstenverlies voor de aanbieder van de toepassing, iets wat deze laatste te allen tijde zal willen vermijden.

De huidige generatie telecommunicatie netwerken, inclusief het Internet, beschikken standaard niet over gesofistikeerde mechanismen voor QoS voorziening. De reden hiervoor is redelijk eenvoudig: ten tijde van hun ontwerp had men relatief eenvoudige gedistribueerde applicaties voor ogen, zoals bijvoorbeeld het reeds eerder aangehaalde e-mail, die geen (of hoogstens zeer summier) gebruik maken van multimediale inhoud. Zodra multimediale applicaties hun opwachting begonnen te maken, werd het echter snel duidelijk dat er wel degelijk nood is aan QoS ondersteuning en werd QoS voorziening plotsklaps een zeer actief onderzoeksdomein. In de loop der jaren zijn er bijgevolg een omvangrijk aantal technieken en raamwerken voorgesteld geworden welke QoS voorziening door het computer netwerk mogelijk maken en hun positieve implicaties op de performantie van gedistribueerde (multimediale) applicaties is reeds meermaals vastgesteld geworden.

Aangezien QoS ondersteuning bijdraagt tot het correct en efficiënt functioneren van gedistribueerde applicaties, draagt het ontegensprekelijk bij tot het verbeteren van de tevredenheid van de eindgebruiker. Vrij recentelijk werd het echter duidelijk dat het verzekeren van een zeker niveau van dienstverlening niet noodzakelijk garant staat voor een optimale gebruikerservaring. De ervaring van een gebruiker van een gedistribueerde toepassing is namelijk een multi-dimensionaal concept dat naast technologische aspecten ook verscheidene factoren uit niet-technische domeinen zoals bijvoorbeeld sociologie en psychologie omvat. Deze vaststelling heeft geleid tot het ontstaan van een volledig nieuw onderzoeksgebied dat zich exclusief richt op het verbeteren van de ervaring die aangeboden wordt aan gebruikers van gedistribueerde applicaties. In deze context wordt typisch de term *Quality of Experience* (QoE) gebruikt om de gebruikerservaring formeel aan te duiden.

QoE optimalisatie is een uitermate omvangrijk probleem dat QoS voorziening overvleugelt. In QoE onderzoek wordt bijvoorbeeld regelmatig een beroep gedaan op QoS technieken om de gebruikerservaring te verbeteren. Net zoals huidige telecommunicatie netwerken weinig tot geen voorzieningen voor QoS garantie omvatten, ontbreekt het hen grotendeels aan intrinsieke QoE optimalisatie mogelijkheden. Het matigen van dit gebrek vormt het onderwerp en de doelstelling van deze thesis.

De algemene onderzoeksbijdrage van dit eindwerk is de ontwikkeling van de NIProxy, wat een acroniem is voor Network Intelligence Proxy. De NIProxy is een software entiteit voor het GNU/Linux besturingssysteem welke in IPv4-gebaseerde computer netwerken geïntegreerd kan worden in de vorm van een zogenaamde proxy server. Eens geïntegreerd, voegt het QoE manipulatie en optimalisatie mogelijkheden toe aan het netwerk. Zoals zijn benaming doet vermoeden, vervult *intelligentie* of *kennis* hierbij een cruciale rol. De NIProxy beheert een databank met contextuele gegevens en vult deze door drie verschillende soorten informatie te verzamelen. Doordat de NIProxy een deel uitmaakt van het telecommunicatie netwerk, vertaalt zich dit conceptueel tot het "intelligenter" maken van het netwerk. De eerste informatiebron die geraadpleegd wordt is het computer netwerk zelf. Dit levert kwantitatieve metingen en statistieken op zoals de doorvoersnelheid, de vertraging en de fouteigenschappen van netwerk connecties. Ten tweede interpelleert de NIProxy de gedistribueerde applicatie. De contextuele kennis neemt in dit geval dus de vorm aan van informatie betreffende de toepassing. Dit type context is zeer variabel; de gedistribueerde applicatie kan naar eigen goeddunken de NIProxy op de hoogte brengen van informatie die op één of andere manier te maken heeft met zijn werking. Een typisch voorbeeld van applicatie-gerelateerde informatie is de relatie, in termen van belangrijkheid voor de eindgebruiker, die bestaat tussen de verschillende types netwerk trafiek die door de applicatie geïntroduceerd worden. De derde en laatste soort contextuele kennis omvat informatie betreffende het apparaat en de voorkeuren van de eindgebruiker. Voorbeelden van informatie die onder deze context categorie vallen zijn de resolutie van het scherm en het type inhoud (audio, video, . . .) dat de voorkeur van de gebruiker wegdraagt.

Op basis van zijn vergaarde context biedt de NIProxy twee complementaire (applicatie-laag) QoS gereedschappen aan waarmee de verspreiding van data trafiek gemanipuleerd kan worden. Via deze technieken tracht de NIProxy het vermogen van het netwerk wat betreft de behandeling van multimediale data uit te breiden en aldus het netwerk te voorzien van mogelijkheden voor het beïnvloeden en verbeteren van de QoE van gebruikers van gedistribueerde applicaties.

Het eerste mechanisme dat ondersteund wordt is *network traffic shaping* (NTS) en stelt de NIProxy in staat de manier waarop gedistribueerde applicaties gebruik maken van netwerk bandbreedte te coördineren. De doelstelling van het NTS mechanisme is tweeledig. Ten eerste moet ervoor gezorgd worden dat de verspreiding van data die geassocieerd is met de gedistribueerde applicatie het bandbreedte volume dat hiervoor gereserveerd werd niet overtreedt. Dergelijke schendingen kunnen namelijk resulteren in netwerk congestie en alle negatieve gevolgen die hier typisch mee verbonden zijn, zoals bijvoorbeeld een toename van de introductie van fouten tijdens de transmissie van data. De tweede doelstelling bestaat eruit het beschikbare bandbreedte budget zo effectief mogelijk aan te wenden. Anders gezegd, het NTS algoritme tracht een bepaalde hoeveelheid bandbreedte te verdelen over de netwerk trafiek die geïntroduceerd wordt door de gedistribueerde applicatie op zo een wijze dat de gebruikerservaring gemaximaliseerd wordt.

Een centrale rol in de NTS methode van de NIProxy is weggelegd voor de *stroom hiërarchie*. In deze boomstructuur worden de data stromen die gegenereerd worden door de gedistribueerde applicatie opgenomen en, op basis van de verzamelde contextuele kennis, gerelateerd aan elkaar. De samenstelling en de opmaak van de stroom hiërarchie bepalen hoe de beschikbare bandbreedte zal verdeeld worden over de aanwezige netwerk trafiek. Het structureren van de hiërarchie gebeurt aan de hand van interne knopen en het zijn deze knopen die de operatie van het NTS mechanisme sturen en dus ook grotendeels de NTS uitkomst bepalen. Verschillende soorten interne knopen zijn voorhanden die elk een bepaalde bandbreedte allocatie procedure implementeren. Effectieve netwerk stromen daarentegen worden steeds voorgesteld met behulp van bladknopen in de boomstructuur en ook deze categorie knopen omvat verschillende varianten. In tegenstelling tot hun interne tegenhangers, voorzien bladknopen geen bandbreedte distributie functionaliteit. In plaats daarvan specificeren ze de mogelijkheden betreffende het aanpassen van het bandbreedte verbruik van de netwerk stroom waarmee ze geassocieerd zijn. Zo definieert een `discrete` bladknoop bijvoorbeeld een beperkt aantal afzonderlijke niveaus die elk overeenkomen met een bepaalde consumptie van bandbreedte; afhankelijk van de hoeveelheid bandbreedte die toegewezen wordt aan de bladknoop, zal één van deze niveaus geselecteerd worden en aldus het bandbreedte verbruik van de gerepresenteerde netwerk stroom bepaald worden.

De geldigheid van de NIProxy's NTS aanpak en de mogelijkheden van deze techniek wat betreft QoE optimalisatie blijken uit de uitkomst van meerdere experimentele evaluaties. Deze experimenten waren uiteenlopend van opzet en varieerden aanzienlijk wat betreft de samenstelling van de netwerk trafiek die beheerd diende te worden. Zo wordt het bijvoorbeeld aangetoond dat de

NIProxy erin slaagt op te treden als bandbreedte makelaar in het geval dat het computer netwerk belast is met de gelijktijdige uitwisseling van real-time en non-real-time data. Deze soorten netwerk trafiek vertonen erg uiteenlopende eigenschappen en vereisen aldus een verschillende aanpak wat betreft bandbreedte beheer. De experimentele resultaten bevestigen dat de NTS technologie van de NIProxy voldoende uitgebreid en geavanceerd is om beide soorten netwerk trafiek aan elkaar te relateren en om hun bandbreedte verbruik adequaat te reguleren.

De validatie van de NTS functionaliteit blijft niet beperkt tot artificiële experimenten waarbij gebruik gemaakt wordt van demonstratoren en kunstmatige applicaties die specifiek werden ontworpen voor evaluatie doeleinden. Zo wordt de NIProxy ook aangewend om het bandbreedte consumptie gedrag van een concrete Genetwerkte Virtuele Omgeving (GVO) toepassing te reguleren. Deze applicatie introduceert een aantal vereisten wat betreft de verspreiding van rendering-gerelateerde data en biedt bovendien audiovisuele voorzieningen om zijn (geografisch verspreide) gebruikers in staat te stellen op een aangename manier te converseren met elkaar. De experimentele resultaten bevestigen dat de NIProxy ervoor kan zorgen dat de vereisten van de applicatie wat betreft netwerk communicatie voldaan zijn, wat op zijn beurt een noodzakelijke voorwaarde is voor het garanderen van een aangename gebruikerservaring.

Het tweede mechanisme dat de NIProxy in staat stelt de netwerk transmissie van data en dus, onrechtstreeks, de gebruikerservaring te manipuleren is *dienstverlening*. De NIProxy kan de rol aannemen van een platform voor de voorziening en uitvoering van *diensten* op onderschepte netwerk trafiek en in het bijzonder trafiek dewelke multimediale data transporteert. Dankzij een plug-in-gebaseerd ontwerp waarbij elke dienst overeenkomt met een NIProxy plug-in, vereist de implementatie van diensten geen wijzigingen aan (en dus hercompilatie van) de algemene software architectuur van de NIProxy. Een ander belangrijk voordeel van dit ontwerp is dat het toelaat dat diensten dynamisch geïnstalleerd en gedeïnstalleerd worden terwijl de NIProxy in uitvoering is, wat op zijn beurt dynamische expansie van de aangeboden functionaliteit toelaat. Op deze manier wordt het mogelijk voor de NIProxy om om te gaan met variabele condities in dynamische en heterogene netwerk omgevingen zonder dat hiervoor een herstart vereist is (dit wil zeggen zonder dat dit leidt tot een tijdelijke onderbreking van zijn operatie).

Theoretisch gezien zijn er geen beperkingen wat betreft de diensten die aangeboden kunnen worden door de NIProxy. Zo wordt er bijvoorbeeld een dienst besproken die on-the-fly H.263 video transcoding functionaliteit implementeert en aldus de NIProxy in staat stelt de bitrate van dit type netwerk trafiek te reduceren. Een niet onbelangrijke restrictie van deze dienst is dat hij "statisch" is: na opstart kan de configuratie van het transcoding proces niet meer aangepast worden. Dit impliceert dat alle bitstreams die geproduceerd worden door een bepaalde instantie van de dienst identieke kwaliteitsparameters en zeer vergelijkbare bandbreedte vereisten zullen vertonen. Ondanks deze beperking blijkt uit experimentele resultaten dat de dienst interessante mogelijkheden inhoudt op het gebied van QoE optimalisatie: daar waar de NIProxy standaard verplicht zou zijn om de verspreiding van bepaalde video data te blokkeren tijdens periodes van bandbreedte schaarste (wat zou inhouden dat een gedeelte van de video trafiek zijn bestemming niet zou bereiken), wordt het nu regelmatig mogelijk om de data alsnog af te leveren, zij het in lagere kwaliteit. Een "dynamische" variant van deze dienst, ditmaal echter gericht op H.264/AVC video, wordt ook gepresenteerd. Deze dienst ondersteunt met andere woorden het transformeren van H.264/AVC data naar een willekeurige en dynamisch aanpasbare bitrate. Er wordt getoond, opnieuw via experimentele evaluatie, dat de dienst extra flexibiliteit introduceert in vergelijking met zijn statische voorganger en alzo bijkomende opties met zich meebrengt wat betreft het verbeteren van de gebruikerservaring, bijvoorbeeld door de NIProxy toe te laten de beschikbare netwerk bandbreedte op een volledigere en efficiëntere manier te benutten. Een laatste voorbeeld is de Forward Error Correction (FEC) dienst dewelke redundantie toevoegt aan netwerk trafiek om de ontvanger in staat te stellen data corruptie en zelfs het verlies van data dat opgelopen wordt tijdens netwerk transmissie (enigszins) te verhelpen. Dergelijke problemen treden relatief frequent op in draadloze omgevingen omwille van interferentie of ruis op het communicatie kanaal. Via een video distributie applicatie wordt het experimenteel bewezen dat corrupte of ontbrekende data een aanzienlijk negatieve invloed kan hebben op de gebruikerservaring en dat de FEC dienst dus een waardevolle toevoeging vormt aan de functionaliteit van de NIProxy.

Zowel de NTS als de dienstverlening gereedschappen zijn *context-bewust* en *context-adaptief* aangezien ze beide ongebreidelde toegang hebben tot de context databank van de NIProxy. Door bij het opstellen van de stroom hiërarchie rekening te houden met contextuele kennis, zal deze informatie automatisch een rol spelen tijdens het beheren en alloceren van de netwerk bandbreedte. Diensten kunnen dan weer hun werking afstellen op de huidige context, wat op zijn beurt ervoor zal zorgen dat de operaties die ze uitvoeren effectief zullen leiden tot een verbetering van de ervaring van de eindgebruiker.

Een definiërend kenmerk van de NIProxy is dat het twee gereedschappen voor de manipulatie van netwerk trafiek in één enkel systeem combineert en dat het dit bovendien op zo een manier doet dat interactie en collaboratie tussen beide technieken ondersteund is. In plaats van de NTS en dienstverlening mechanismen als geïsoleerde elementen te implementeren, wordt er gekozen voor een geïntegreerde oplossing waarbij beide technieken in staat gesteld worden samen te werken tijdens QoE manipulatie. Dit resulteert in een symbiose: dankzij de mogelijkheid tot collaboratie wordt het QoE optimalisatie potentieel van de individuele gereedschappen en dus ook dat van de NIProxy in zijn geheel aanzienlijk uitgebreid. De interactieve aanpak evenals de voordelen die eruit voortvloeien wat betreft het beheer van de gebruikerservaring, worden geïllustreerd door elk van de drie diensten die in de vorige paragraaf werden aangehaald. Deze diensten introduceren elk een nieuw soort netwerk trafiek en maken daarom gebruik van hun interface met het NTS mechanisme om het te informeren over het bestaan van deze trafiek en over haar vereisten in termen van netwerk bandbreedte. Zodra dit is gebeurd, zal de aangeleverde informatie in overweging genomen worden tijdens bandbreedte bemiddeling. Omgekeerd consulteren de aangehaalde diensten het NTS raamwerk om zich ervan te vergewissen of ze hun functionaliteit moeten toepassen op onderschepte data en, zo ja, op welke manier. Merk op dat dit inhoudt dat het QoE optimalisatie profijt dat geassocieerd is met elk van de besproken diensten niet enkel de verdienste is van de functionaliteit die ze implementeren maar, net zo belangrijk, ook van hun correcte interactie met het NTS mechanisme. Op deze manier bewijzen ze impliciet dat de NIProxy's geïntegreerde aanpak voor de manipulatie van netwerk trafiek resulteert in een QoE optimalisatie performantie en potentieel die ruimschoots de mogelijkheden overschrijden die bereikt kunnen worden wanneer beide gereedschappen losstaand toegepast worden.

De software architectuur van de NIProxy is voldoende flexibel zodat de aangeboden technieken voor de coördinatie van netwerk trafiek toegepast kunnen worden op zowel data die bestemd is voor een gebruiker die beheerd wordt door de NIProxy als op de data die door een dergelijke gebruiker zelf geïnjecteerd wordt in het netwerk. Het ligt voor de hand dat de eerste optie een substantiële en onmiddellijke impact heeft op de QoE die ervaren wordt door de beheerde eindgebruiker. Experimentele resultaten bevestigen dat ook de tweede mogelijkheid potentieel vertoont wat betreft QoE optimalisatie. Meer zelfs, het manipuleren van de data die een eindgebruiker zelf uitstuurt, houdt QoE optimalisatie mogelijkheden in op twee fronten: niet enkel de bron van de data profiteert van dergelijke operaties, ook andere gebruikers die mogelijk niet rechtstreeks door een NIProxy instantie beheerd worden, kunnen er een positieve invloed van ondervinden.

QoE optimalisatie is een erg uitgebreid onderzoeksdomein. De verzameling van mogelijke technieken en operaties voor het verbeteren van de gebruikerservaring is dermate veelzijdig dat het onrealistisch is te veronderstellen dat een individueel systeem ze allemaal zal kunnen voorzien. Een potentiële oplossing voor dit probleem bestaat eruit meerdere raamwerken, bij voorkeur raamwerken die zich concentreren op complementaire vormen van QoE manipulatie, te bundelen. Het wordt gedemonstreerd dat de NIProxy open staat voor dergelijke initiatieven. Meer bepaald wordt een gelaagd platform voorgesteld waarin de NIProxy gecombineerd wordt met overlay routering technologie. Het platform wendt de NIProxy's faciliteiten voor het beheren van netwerk trafiek aan om de transmissie van multimediale inhoud over het laatste gedeelte van een netwerkverbinding (het zogenoemde access netwerk) te optimaliseren, terwijl de routering functionaliteit gebruikt wordt om veerkracht te bereiken tegen architecturale problemen in de kern van het netwerk zoals hardware defecten of congestie. Representatieve experimentele resultaten bevestigen dat het gecombineerde platform slaagt in het toepassen van QoE optimalisatie operaties langsheen bijna de volledige netwerk route tussen bron en bestemming; enkel het initiële gedeelte van de netwerkverbinding (het access netwerk van de bron) wordt niet expliciet beschermd. Het platform vormt het resultaat van een samenwerkingsverband tussen Universiteit Hasselt en Universiteit Gent.

Samengevat contribueert dit proefschrift dus aan drie specifieke onderzoeksgebieden. De eerste bijdrage situeert zich in het domein van (applicatielaag) QoS voorziening en, meer bepaald, van intra-netwerk manipulatie van data trafiek. De NIProxy incorporeert namelijk twee gereedschappen, network traffic shaping en dienstverlening, waarmee het de netwerk transmissie en distributie van (multimediale) data kan beïnvloeden. Ten tweede draagt de NIProxy bij aan het onderzoek naar het ontwerpen en ontwikkelen van ondersteunende netwerk infrastructuur en in het bijzonder intelligente intra-netwerk entiteiten. De belangrijkste verwezenlijking in dit domein is de geïntegreerde aanpak waardoor de voorziene QoS gereedschappen in staat gesteld worden samen te werken, wat op zijn beurt krachtige extra mogelijkheden betreffende QoE optimalisatie toelaat. Als laatste presenteert dit proefschrift significante resultaten op het gebied van context-bewuste en context-adaptieve data disseminatie via IPv4-gebaseerde telecommunicatie netwerken en toont het aan dat dit kan leiden tot een verbetering van de tevredenheid van de gebruiker van (multimediale) gedistribueerde applicaties. Verschillende experimentele evaluaties en studies bevestigen dat de NIProxy in staat is om de gebruikerservaring te verbeteren in uiteenlopende gedistribueerde scenario's, in dynamische omgevingen, onder variabele netwerk belasting en te midden van heterogene soorten netwerk trafiek. Het wordt ook experimenteel bewezen dat de NIProxy ontvankelijk is voor collaboratie met andere oplossingen voor QoS voorziening of QoE optimalisatie en dat het aldus kan dienen als bouwsteen voor de constructie van omvangrijkere raamwerken.

Tenslotte loont het de moeite de onderwerpen te onderstrepen die door deze thesis *niet* behandeld worden. De NIProxy is *niet* begaan met (methoden voor) QoE meting of met het QoE concept op zichzelf. Daarnaast bespreekt deze dissertatie *evenmin* het verzamelen van kwalitatieve feedback via het organiseren van gebruikerstesten. Als laatste wordt het benadrukt dat de NIProxy *geen* kant-en-klare oplossing is voor QoE optimalisatie maar eerder een QoE manipulatie raamwerk wiens effectieve gedrag afgestemd dient te worden op de omgeving waarin het geïntroduceerd wordt, de gedistribueerde applicatie waarop de QoE manipulatie van toepassing is en mogelijk een hele resem additionele contextuele parameters. Dit houdt in dat het simpelweg installeren van een aantal NIProxy instanties in een computer netwerk *geenszins* een garantie is voor succes. Het houdt eveneens in dat de absolute waardes van de experimentele resultaten die gepresenteerd worden in dit proefschrift van ondergeschikt belang zijn; de experimenten en studies hebben eerder als doel de mogelijkheden van de NIProxy op het gebied van het optimaliseren van de gebruikerservaring te demonstreren. De resultaten worden om deze reden *niet* onderworpen aan een statistische analyse om hun statistische significantie te achterhalen.

De technologie achter de NIProxy biedt ruimte voor uitbreiding en een aantal bijkomende toepassingsgebieden zouden onderzocht kunnen worden. Zo wordt kennis van het apparaat en de voorkeuren van de eindgebruiker slechts sporadisch en miniem uitgebuit in de experimenten die besproken worden in dit proefschrift. Het lijkt aangeraden meer aandacht te besteden aan dit type context tijdens toekomstige experimentele evaluaties. Daarnaast wordt het verwacht dat de implementatie van extra soorten interne knopen voor de stroom hiërarchie de NIProxy's opties betreffende bandbreedte bemiddeling zou uitbreiden, terwijl de ontwikkeling van nieuwe diensten de veelzijdigheid en de toepasbaarheid van de NIProxy zou verhogen. Een ander interessant onderwerp voor toekomstig onderzoek is de combinatie van verschillende NIProxy instanties zodat een *hiërarchische* of *gelaagde* oplossing voor QoE optimalisatie ontstaat waarbij de samenstellende instanties gestructureerd zijn volgens kind/ouder relaties en dus "gebruikers" van elkaar zijn. Een volgende mogelijke onderzoekspiste is QoE optimalisatie in draadloze omgevingen. De NIProxy wordt in deze dissertatie voornamelijk onderzocht in de context van draad-gebaseerde computer netwerken. Er bestaan heel wat verschillen tussen beide types netwerken en daarom lijkt een aangepaste, gespecialiseerde aanpak voor draadloze omgevingen aangewezen. Tenslotte houdt ook QoE beheer door middel van mobiele apparatuur zoals laptops en smartphones een groot aantal mogelijkheden in die verder onderzoek verdienen.

Bibliography

[Abowd 99] Gregory D. Abowd & Anind K. Dey. *Towards a Better Understanding of Context and Context-Awareness.* In Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing (HUC 1999), pages 304–307, Karlsruhe, Germany, June 1999.

[Aghera 03] Parixit Aghera, Advait Dixit, Ricardo Oliveira & Vidyut Samanta. *Wireless Middleware: Dynamic Video Transcoding.* Project Report CS211, Computer Science Department, UCLA, 2003. `http://compilers.cs.ucla.edu/vids/mws/mws_final.pdf`.

[AlRegib 05] Ghassan AlRegib & Yucel Altunbasak. *3TP: An Application-Layer Protocol for Streaming 3-D Models.* IEEE Transactions on Multimedia, vol. 7, no. 6, pages 1149–1156, December 2005.

[Amir 95] Elan Amir, Steven McCanne & Hui Zhang. *An Application Level Video Gateway.* In Proceedings of the 3rd ACM International Conference on Multimedia (MULTIMEDIA 1995), pages 255–265, San Francisco, California, USA, November 1995.

[Amir 97] Elan Amir, Steven McCanne & Randy Katz. *Receiver-driven Bandwidth Adaptation for Light-weight Sessions.* In Proceedings of the 5th ACM International Conference on Multimedia (MULTIMEDIA 1997), pages 415–426, Seattle, Washington, USA, November 1997.

[Amir 98] Elan Amir, Steven McCanne & Randy Katz. *An Active Service Framework and its Application to Real-time Multimedia Transcoding.* In Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM 1998), pages 178–189, Vancouver, Canada, September 1998.

[Andersen 00] David Andersen, Deepak Bansal, Dorothy Curtis, Srinivasan Seshan & Hari Balakrishnan. *System Support for Bandwidth Management and Content Adaptation in Internet Applications.* In Proceedings of the 4th Symposium on Operating Systems Design and Implementation (OSDI 2000), pages 213–226, San Diego, California, USA, October 2000.

[Andersen 01] David Andersen, Hari Balakrishnan, Frans Kaashoek & Robert Morris. *Resilient Overlay Networks.* In Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP 2001), pages 131–145, Banff, Canada, October 2001.

[Anjum 99] Farooq M. Anjum & Leandros Tassiulas. *Fair Bandwidth Sharing among Adaptive and Non-Adaptive Flows in the Internet.* In Proceedings of the 18th IEEE Conference on Computer Communications (INFOCOM 1999), pages 1412–1420, New York, New York, USA, March 1999.

[ARIN 08] ARIN. *American Registry for Internet Numbers (ARIN) and the Cooperative Association for Internet Data Analysis (CAIDA): IPv6 Penetration Survey Results.* Presented at the ARIN XXI meeting, available online at `https://www.arin.net/participate/meetings/reports/ARIN_XXI/PDF/monday/IPv6_Survey_KC.pdf`, April 2008.

[Arora 09] Roman Arora, Vangelis Metsis, Rong Zhang & Fillia Makedon. *Providing QoS in Ontology Centered Con-*

text Aware Pervasive Systems. In Proceedings of the 2nd International Conference on PErvsive Technologies Related to Assistive environments (PETRA 2009), pages 1–8, Corfu, Greece, June 2009.

[Balakrishnan 99] Hari Balakrishnan, Hariharan S. Rahul & Srinivasan Seshan. *An Integrated Congestion Management Architecture for Internet Hosts.* ACM SIGCOMM Computer Communication Review, vol. 29, no. 4, pages 175–187, October 1999.

[Bandel 01] David A. Bandel. *Taming the Wild Netfilter.* Online, `http://www.linuxjournal.com/article/4815`, September 2001.

[Banka 07] Tarun Banka, Panho Lee, Anura P. Jayasumana & Jim Kurose. *An Architecture and a Programming Interface for Application-Aware Data Dissemination Using Overlay Networks.* In Proceedings of the 2nd IEEE International Conference on COMmunication System softWAre and MiddlewaRE (COMSWARE 2007), Bangalore, India, January 2007.

[Barrett 01] Lisa F. Barrett & Daniel J. Barrett. *An Introduction to Computerized Experience Sampling in Psychology.* Social Science Computer Review, vol. 19, no. 2, pages 175–185, Summer 2001.

[Barry 03] Douglas K. Barry. Web Services and Service-Oriented Architectures: The Savvy Manager's Guide. Morgan Kaufmann, 2003.

[Beauregard 07] Russell Beauregard & Philip Corriveau. *User Experience Quality: A Conceptual Framework for Goal Setting and Measurement.* In Proceedings of 1ste International Conference on Digital Human Modeling (ICDHM 2007), pages 325–332, Beijing, China, July 2007.

[Blake 98] Steven Blake, David L. Black, Mark A. Carlson, Elwyn Davies, Zheng Wang & Walter Weiss. *An Architecture for Differentiated Services.* RFC 2475, Internet Engineering Task Force, December 1998. http://www.ietf.org/rfc/rfc2475.txt.

[Boier-Martin 03] Ioana M. Boier-Martin. *Adaptive Graphics.* IEEE Computer Graphics and Applications, vol. 23, no. 1, pages 6–10, January 2003.

[Bolla 08] Raffaele Bolla, Matteo Repetto, Saar De Zutter, Rik Van de Walle, Stefano Chessa, Francesco Furfari, Bernhard Reiterer, Hermann Hellwagner, Mark Asbach & Mathias Wien. *A Context-Aware Architecture for QoS and Transcoding Management of Multimedia Streams in Smart Homes.* In Proceedings of 13th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2008), pages 1354–1361, Hamburg, Germany, September 2008.

[Bolot 99] Jean-Chrysostome Bolot, Sacha Fosse-Parisis & Don Towsley. *Adaptive FEC-Based Error Control for Internet Telephony.* In Proceedings of 18th IEEE Conference on Computer Communications (INFOCOM 1999), pages 1453–1460, New York, New York, USA, March 1999.

[Braden 94] Rob Braden, Dave Clark & Scott Shenker. *Integrated Services in the Internet Architecture: an Overview.* RFC 1633, Internet Engineering Task Force, June 1994. http://www.ietf.org/rfc/rfc1633.txt.

[Braden 97] Rob Braden, Lixia Zhang, Steve Berson, Shai Herzog & Sugih Jamin. *Resource ReSerVation Protocol (RSVP).* RFC 2205, Internet Engineering Task Force, September 1997. http://www.ietf.org/rfc/rfc2205.txt.

[Bush 01] Stephen F. Bush & Amit B. Kulkarni. Active Networks and Active Network Management: A Proactive Management Framework. Kluwer Academic Publishers, 2001.

[But 08] Jason But, Grenville Armitage & Lawrence Stewart. *Outsourcing Automated QoS Control of Home Routers* *for a Better Online Game Experience.* IEEE Communications Magazine, vol. 46, no. 12, pages 64–70, December 2008.

[Cai 08] Wen-Yu Cai & Hai-Bo Yang. *A Hierarchical QoS Framework for Wireless Multimedia Network.* In Proceedings of the International Conference on Communications, Circuits And Systems (ICCCAS 2008), pages 732–736, Fujian, China, May 2008.

[Cardinaels 06] Maarten Cardinaels, Geert Vanderhulst, Maarten Wijnants, Chris Raymaekers, Kris Luyten & Karin Coninx. *Seamless Interaction between Multiple Devices and Meeting Rooms.* In Proceedings of the CHI Workshop on Information Visualization and Interaction Techniques for Collaboration across Multiple Displays (IVITCMD 2006), Montreal, Canada, April 2006.

[Chandra 00] Surendar Chandra, Carla Schlatter Ellis & Amin Vahdat. *Differentiated Multimedia Web Services Using Quality Aware Transcoding.* In Proceedings of the 19th IEEE Conference on Computer Communications (INFOCOM 2000), pages 961–969, Tel Aviv, Israel, March 2000.

[Chen 04] Lei Chen & Wendi Heinzelman. *Network Architecture to Support QoS in Mobile Ad Hoc Networks.* In Proceedings of the IEEE International Conference on Multimedia and Expo (ICME 2004), pages 1715– 1718, Taipei, Taiwan, June 2004.

[Clark 92] David D. Clark, Scott Shenker & Lixia Zhang. *Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanism.* In Proceedings of the ACM SIGCOMM Conference on Communications Architectures and Protocols (SIGCOMM 1992), pages 14–26, Baltimore, Maryland, USA, August 1992.

[Creemers 09] Tomas Creemers. Detecteren, Representeren en Uitbuiten van de Mogelijkheden van het Apparaat van de Eind-Gebruiker. Master's thesis, Hasselt University, June 2009.

[D-Link 10] D-Link. *The D-Link DGL-4500 Xtreme N Gaming Router Product Page.* Online, `http://www.dlink.com/products/?pid=643`, 2010.

[DARPA 10] DARPA. *The Defense Advanced Research Projects Agency Homepage.* Online, `http://www.darpa.mil/`, 2010.

[De Marez 09] Lieven De Marez & Katrien De Moor. The Challenge of User- and QoE-Centric Research and Product Development in Today's ICT-Environment, pages 56–78. Chapter in "ICT for Development. Prospects and Problems". The Icfai University Press, 2009.

[De Silva 99] Ranil De Silva, Björn Landfeldt, Sebastien Ardon, Aruna Seneviratne & Christophe Diot. *Managing Application Level Quality of Service through TOMTEN.* Computer Networks - The International Journal of Computer and Telecommunications Networking, vol. 31, no. 7, pages 727–739, April 1999.

[De Vleeschauwer 04] Bart De Vleeschauwer, Filip De Turck, Bart Dhoedt & Piet Demeester. *On the Construction of QoS Enabled Overlay Networks.* In Quality of Future Internet Services (QofIS 2004), volume 3266 of *Lecture Notes in Computer Science*, pages 164–173, Barcelona, Spain, September 2004.

[De Vleeschauwer 06] Bart De Vleeschauwer, Filip De Turck, Bart Dhoedt & Piet Demeester. *Dynamic Algorithms to Provide a Robust and Scalable Overlay Routing Service.* In Proceedings of the 20th IEEE International Conference on Information Networking (ICOIN 2006), pages 945–954, Sendai, Japan, January 2006.

[De Vleeschauwer 07a] Bart De Vleeschauwer, Filip De Turck, Bart Dhoedt & Piet Demeester. *Online Management of QoS Enabled Overlay Multicast Services.* In Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM 2007), San Fransisco, California, USA, November 2007.

[De Vleeschauwer 07b] Bart De Vleeschauwer, Pieter Simoens, Wim Van de Meerssche, Filip De Turck, Bart Dhoedt, Piet Demeester, Kris Struyve, Tom Van Caenegem, Edith Gilon, Hans Dequeker & Erwin Six. *Enabling Autonomic Access Network QoE Management Through TCP Connection Monitoring.* In Proceedings of the 1st IEEE Workshop on Autonomic Communications and Network Management (ACNM 2007), Munich, Germany, May 2007.

[De Vleeschauwer 08a] Bart De Vleeschauwer. *Quality Optimization of Multimedia Services through Overlay Networks.* PhD thesis, Ghent University, Department of Information Technology (INTEC), 2008.

[De Vleeschauwer 08b] Bart De Vleeschauwer, Filip De Turck, Bart Dhoedt, Piet Demeester, Maarten Wijnants & Wim Lamotte. *End-to-end QoE Optimization Through Overlay Network Deployment.* In Proceedings of the 22nd IEEE International Conference on Information Networking (ICOIN 2008), Busan, Korea, January 2008.

[De Vleeschauwer 10] Bart De Vleeschauwer, Filip De Turck, Bart Dhoedt & Piet Demeester. *Dynamic Overlay Networks for Robust and Scalable Routing.* In Handbook of Research on P2P and Grid Systems for Service-Oriented Computing: Models, Methodologies and Applications, pages 543–565. Information Science Publishing, January 2010.

[Deryckere 08] Tom Deryckere, Wout Joseph, Luc Martens, Lieven De Marez, Katrien De Moor & Katrien Berte. *A Software Tool to Relate Technical Performance to User Experience in a Mobile Context.* In Proceedings of the 9th IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM 2008), pages 673–678, Newport Beach, California, USA, June 2008.

[Dey 08] Sujit Dey. *Mobile Video Streaming Quality and System Design for QoE.* EE Times India, available online at `http://www.eetindia.co.in/`

ART_8800546515_1800005_TA_9482845d.HTM, October 2008.

[Dovrolis 99] Constantinos Dovrolis & Parameswaran Ramanathan. *A Case for Relative Differentiated Services and the Proportional Differentiation Model.* IEEE Network, vol. 13, no. 5, pages 26–34, September/October 1999.

[Dovrolis 08] Constantine Dovrolis. *What Would Darwin Think about Clean-Slate Architectures?* ACM SIGCOMM Computer Communication Review, vol. 38, no. 1, pages 29–34, January 2008.

[Eberle 05] Wolfgang Eberle, Bruno Bougard, Sofie Pollin & Francky Catthoor. *From Myth to Methodology: Cross-Layer Design for Energy-Efficient Wireless Communication.* In Proceedings of the 42nd Annual Design Automation Conference (DAC 2005), pages 303–308, Anaheim, California, USA, June 2005.

[Feamster 03] Nick Feamster, David G. Andersen, Hari Balakrishnan & M. Frans Kaashoek. *Measuring the Effects of Internet Path Faults on Reactive Routing.* In Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, pages 126–137, San Diego, California, USA, June 2003.

[FFMPEG 10] FFMPEG. *The FFMPEG Homepage.* Online, http://ffmpeg.org/, 2010.

[Filho 06] Fernando Silveira Filho, Edson H. Watanabe & Edmundo de Souza e Silva. *Adaptive Forward Error Correction for Interactive Streaming Over the Internet.* In Proceedings of the 49th IEEE Global Telecommunications Conference (GLOBECOM 2006), pages 1–6, San Francisco, California, USA, November 2006.

[Floyd 95] Sally Floyd & Van Jacobson. *Link-sharing and Resource Management Models for Packet Networks.* IEEE/ACM Transactions on Networking, vol. 3, no. 4, pages 365–386, August 1995.

[Foster 04] Ian Foster, Markus Fidler, Alain Roy, Volker Sander & Linda Winkler. *End-to-End Quality of Service for High-End Applications.* Computer Communications, vol. 27, no. 14, pages 1375–1388, September 2004.

[Fox 96] Armando Fox, Steven D. Gribble, Eric A. Brewer & Elan Amir. *Adapting to Network and Client Variability via On-Demand Dynamic Distillation.* ACM SIGOPS Operating Systems Review, vol. 30, no. 5, pages 160–170, December 1996.

[Fox 98] Armando Fox, Steven D. Gribble, Yatin Chawathe & Eric A. Brewer. *Adapting to Network and Client Variation Using Active Proxies: Lessons and Perspectives.* IEEE Personal Communications, vol. 5, no. 4, pages 10–19, August 1998.

[Froehlich 07] Jon Froehlich, Mike Y. Chen, Sunny Consolvo, Beverly Harrison & James A. Landay. *MyExperience: A System for In Situ Tracing and Capturing of User Feedback on Mobile Phones.* In Proceedings of the 5th International Conference on Mobile Systems, Applications and Services (MobiSys 2007), pages 57–70, San Juan, Puerto Rico, June 2007.

[Frossard 01] Pascal Frossard & Olivier Verscheure. *Joint Source/FEC Rate Selection for Quality-Optimal MPEG-2 Video Delivery.* IEEE Transactions on Image Processing, vol. 10, no. 12, pages 1815–1825, December 2001.

[Furini 01] Marco Furini & Donald Towsley. *Real-Time Traffic Transmission over the Internet.* IEEE Transactions on Multimedia, vol. 3, no. 1, pages 33–40, March 2001.

[Gavrilovska 05] Ada Gavrilovska, Sanjay Kumar, Srikanth Sundaragopalan & Karsten Schwan. *Platform Overlays: Enabling In-Network Stream Processing in Large-scale Distributed Applications.* In Proceedings of the 15th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV 2005), pages 171–176, Stevenson, Washington, USA, 2005.

[Gelas 00] Jean-Patrick Gelas & Laurent Lefèvre. *TAMANOIR: A High Performance Active Network Framework.* In Proceedings of the 2nd Annual Workshop on Active Middleware Services (AMS 2000), pages 105–114, Pittsburgh, Pennsylvania, USA, August 2000.

[Gilon - de Lumley 07] Edith Gilon - de Lumley, Jeroen Hoet, Hans Dequeker, Raf Huysegems, Erwin Six, Wim Van de Meerssche, Pieter Simoens, Bart De Vleeschauwer, Cristina Pena, Bjoern Nagel & Peter Vetter. *Service Rich Access Networks: The Service Plane Solution.* In Proceedings of BroadBand Europe, Antwerp, Belgium, December 2007.

[Gimson 03] Roger Gimson, Shlomit Ritz Finkelstein, Stéphane Maes & Lalitha Suryanarayana. *Device Independence Principles.* W3C Working Group Note, W3C, September 2003. `http://www.w3.org/TR/2003/NOTE-di-princ-20030901/`.

[Gruenen 06] Jana Van Gruenen, Yury Markovsky, Chris R. Baker, Jan Rabaey, John Wawrzynek & Adam Wolisz. *ZUMA: A Platform for Smart-Home Environments, The Case for Infrastructure.* In Proceedings of the 2nd International Conference on Intelligent Environments (IE 2006), pages 257–266, Athens, Greece, July 2006.

[GT NPG 10] GT NPG. *The Georgia Tech Network Processors Group Homepage.* Online, `http://www.cercs.gatech.edu/projects/npg/index.html`, 2010.

[Gupta 02] Rajarshi Gupta, Mike Chen, Steven McCanne & Jean Walrand. *A Receiver-Driven Transport Protocol for the Web.* Telecommunication Systems, vol. 21, no. 2, pages 213–230, December 2002.

[Hemminger 05] Stephen Hemminger. *Network Emulation with NetEm.* In Proceedings of linux.conf.au (LCA 2005), Canberra, Australia, April 2005.

[Hnatyshin 06] Vasil Hnatyshin & Adarshpal S. Sethi. *Architecture for Dynamic and Fair Distribution of Bandwidth.* International Journal of Network Management, vol. 16, no. 5, pages 317–336, September/October 2006.

[Hoppe 96] Hugues Hoppe. *Progressive Meshes.* In Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH 1996), pages 99–108, New Orleans, Louisiana, USA, August 1996.

[Huston 00] Geoff Huston. *Next Steps for the IP QoS Architecture.* RFC 2990, Internet Engineering Task Force, November 2000. `http://www.ietf.org/rfc/rfc2990.txt`.

[IETF 10] IETF. *The Internet Engineering Task Force Homepage.* Online, `http://www.ietf.org/`, 2010.

[ISO/IEC 02] ISO/IEC. *ISO/IEC 21000-1: Information Technology — Multimedia Framework (MPEG-21) — Part 1: Vision, Technologies and Strategy.* Technical report, ISO, 2002.

[ISO/IEC 04] ISO/IEC. *ISO/IEC 21000-7: Information Technology — Multimedia Framework (MPEG-21) — Part 7: Digital Item Adaptation.* Technical report, ISO, 2004.

[Jacobs 98] Stephen Jacobs, Alexandros Eleftheriadis & Ros Eleftheriadis. *Streaming Video using Dynamic Rate Shaping and TCP Congestion Control.* Journal of Visual Communication and Image Representation, vol. 9, no. 3, pages 211–222, September 1998.

[Jain 05] Manish Jain & Constantinos Dovrolis. *End-to-end Estimation of the Available Bandwidth Variation Range.* In Proceedings of the ACM International Conference on Measurements and Modeling of Computer Systems (SIGMETRICS 2005), pages 265–276, Banff, Canada, June 2005.

[Jehaes 04a] Tom Jehaes, Peter Quax & Wim Lamotte. *Analysis of Scalable Data Streams for Representations in Networked Virtual Environments.* In Proceedings of the ACM SIGCOMM Workshop on Network and System Support for Games (NETGAMES 2004), Portland, Oregon, USA, August 2004.

[Jehaes 04b] Tom Jehaes, Peter Quax, Patrick Monsieurs & Wim Lamotte. *Hybrid Representations to Improve Both Streaming and Rendering of Dynamic Networked Virtual Environments.* In Proceedings of the ACM SIGGRAPH International Conference on Virtual-Reality Continuum and its Applications in Industry (VRCAI 2004), pages 26–32, Singapore, June 2004.

[Jehaes 08] Tom Jehaes. *Efficient Representation, Transmission and Rendering of Networked 3D Virtual Environments on Desktop and Mobile Systems.* PhD thesis, Hasselt University, Expertise centre for Digital Media (EDM), 2008.

[Kassler 01] Andreas Kassler, Christian Kücherer & Andreas Schrader. *Adaptive Wavelet Video Filtering.* In Proceedings of the 2nd International Workshop on Quality of Future Internet Services (QofIS 2001), pages 32–44, Coimbra, Portugal, September 2001.

[Kassler 03] Andreas Kassler & Andreas Schorr. *Generic QoS Aware Media Stream Transcoding and Adaptation.* In Proceedings of the 13th IEEE Packet Video Workshop (PV 2003), Nantes, France, April 2003.

[Knutsson 03] Björn Knutsson, Honghui Lu, Jeffrey Mogul & Bryan Hopkins. *Architecture and Performance of Server-Directed Transcoding.* ACM Transactions on Internet Technology, vol. 3, no. 4, pages 392–424, November 2003.

[Kohler 00] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti & M. Frans Kaashoek. *The Click Modular Router.* ACM Transactions on Computer Systems, vol. 18, no. 3, pages 263–297, August 2000.

[Kumar 03] Rajeev Kumar. *A Protocol with Transcoding to Support QoS over Internet for Multimedia Traffic.* In Proceedings of the IEEE International Conference on Multimedia and Expo (ICME 2003), pages 465–468, Baltimore, Maryland, USA, July 2003.

[Kumar 05] Kshitij Kumar. *A Marriage Made in QoE Heaven: Flow-Based IP and PacketCable Multimedia.* CED Magazine, July 2005.

[Kusmierek 02] Ewa Kusmierek, Baek-Young Choi, Zhenhai Duan & Zhi-Li Zhang. *An Integrated Network Resource and QoS Management Framework.* In Proceedings of the IEEE Workshop on IP Operations and Management (IPOM 2002), pages 68–72, Dallas, Texas, USA, October 2002.

[Lefèvre 01] Laurent Lefèvre, Cong duc Pham, Pascale Primet, Bernard Tourancheau, Benjamin Gaidioz, Tourancheaubenjamin Gaidioz, Jean-Patrick Gelas & Moufida Maimour. *Active Networking Support for The Grid.* In Proceedings of the IFIP-TC6 3rd International Working Conference on Active Networks (IWAN 2001), pages 16–33, Philadelphia, Pennsylvania, USA, October 2001.

[Legedza 98] Ulana Legedza, David J. Wetherall & John Guttag. *Improving the Performance of Distributed Applications Using Active Networks.* In Proceedings of the 17th IEEE Conference on Computer Communications (INFOCOM 1998), pages 590–599, San Francisco, California, USA, April 1998.

[Lei 03] Zhijun Lei & Nicolas D. Georganas. *Video Transcoding Gateway For Wireless Video Access.* In Proceedings of the IEEE Canadian Conference on Electrical and Computer Engineering (CCECE 2003), pages 1775–1778, Montreal, Canada, May 2003.

[Li 04] Zhi Li & Prasant Mohapatra. *QRON: QoS-aware Routing in Overlay Networks.* IEEE Journal on Selected Areas in Communications, vol. 22, no. 1, pages 29–40, January 2004.

[Li 06] Hui Li, Ming Li & Balakrishnan Prabhakaran. *Middleware for Streaming 3D Progressive Meshes over Lossy Networks.* ACM Transactions on Multimedia Computing, Communications and Applications, vol. 2, no. 4, pages 282–317, November 2006.

[Li 07] Adam Li. *RTP Payload Format for Generic Forward Error Correction.* RFC 5109, Internet Engineering Task Force, December 2007. `http://www.ietf.org/rfc/rfc5109.txt`.

[Liang 05] Jin Liang & Klara Nahrstedt. *Service Composition for Advanced Multimedia Applications.* In Proceedings of the 12th Annual Multimedia Computing and Networking Conference (MMCN'05), pages 228–240, San Jose, California, USA, January 2005.

[Lin 04a] Ching-Yung Lin, Apostol Natsev, Belle L. Tseng, Matthew Hill, John R. Smith & Chung-Sheng Li. *Content Transcoding Middleware for Pervasive Geospatial Intelligence Access.* In Proceedings of the IEEE International Conference on Multimedia and Expo (ICME 2004), Taipei, Taiwan, June 2004.

[Lin 04b] Shu Lin & Daniel J. Costello. Error Control Coding. Prentice-Hall, 2nd edition, 2004.

[Lum 02] Wai Yip Lum & Francis Lau. *A Context-Aware Decision Engine for Content Adaptation.* IEEE Pervasive Computing, vol. 1, no. 3, pages 41–49, July-September 2002.

[Lyijynen 03] Marko Lyijynen, Titta Koskinen, Sami Lehtonen & Juuso Pesola. *Content Adaptation on LANE Active Network Platform.* In Proceedings of the 7th International Conference on Telecommunications (ConTEL 2003), pages 11–14, Zagreb, Croatia, June 2003.

[Maheshwari 02] Anuj Maheshwari, Aashish Sharma, Krithi Ramamritham & Prashant Shenoy. *TranSquid: Transcoding and Caching Proxy for Heterogenous E-Commerce Environments.* In Proceedings of the 12th IEEE International Workshop on Research Issues in Data Engineering (RIDE 2002), pages 50–59, San Jose, California, USA, February 2002.

[Mankin 97] Allison Mankin, Fred Baker, Bob Braden, Scott Bradner, Michael O'Dell, Allyn Romanow, Abel Weinrib & Lixia Zhang. *RSVP Applicability and Deployment.*

RFC 2208, Internet Engineering Task Force, September 1997. `http://www.ietf.org/rfc/rfc2208.txt`.

[Marcus 98] William S. Marcus, Ilija Hadzic, Anthony J. McAuley & Jonathan M. Smith. *Protocol Boosters: Applying Programmability to Network Infrastructures.* IEEE Communications Magazine, vol. 36, no. 10, pages 79–83, October 1998.

[Martin 00] Ioana M. Martin. *ARTE - An Adpative Rendering and Transmission Environment for 3D Graphics.* In Proceedings of the 8th ACM International Conference on Multimedia (MULTIMEDIA 2000), pages 413–415, Los Angeles, California, USA, November 2000.

[Martin 02] Ioana M. Martin. *Hybrid Transcoding for Adaptive Transmission of 3D Content.* In Proceedings of the IEEE International Conference on Multimedia and Expo (ICME 2002), pages 373– 376, Lausanne, Switzerland, August 2002.

[Mohan 99] Rakesh Mohan, John R. Smith & Chung-Sheng Li. *Adapting Multimedia Internet Content for Universal Access.* IEEE Transactions on Multimedia, vol. 1, no. 1, pages 104–114, March 1999.

[Monsieurs 05] Patrick Monsieurs, Maarten Wijnants & Wim Lamotte. *Client-controlled QoS Management in Networked Virtual Environments.* In Proceedings of the 4th International Conference on Networking (ICN 2005), pages 268–276, Reunion Island, April 2005.

[Moon 05] Todd K. Moon. Error Correction Coding: Mathematical Methods and Algorithms. Wiley-Interscience, 2005.

[Mulder 05] Ingrid Mulder, Henri ter Hofte & Joke Kort. *SocioXensor: Measuring User Behaviour and User Experience in Context with Mobile Devices.* In Proceedings of the 5th International Conference on Methods and Techniques in Behavioral Research (Measuring Behavior 2005), pages 355–358, Wageningen, the Netherlands, September 2005.

[Mulroy 06] Patrick Mulroy. *Application Layer QoS for Videotelephony.* BT Technology Journal, vol. 24, no. 2, pages 167–173, April 2006.

[MyExperience 10] MyExperience. *The MyExperience Homepage.* Online, `http://myexperience.sourceforge.net/`, 2010.

[Nahrstedt 01] Klara Nahrstedt, Dongyan Xu, Duangdao Wichadakul & Baochun Li. *QoS-Aware Middleware for Ubiquitous and Heterogeneous Environments.* IEEE Communications Magazine, vol. 39, no. 11, pages 140–148, November 2001.

[Nahrstedt 05] Klara Nahrstedt, Bin Yu, Jin Liang & Yi Cui. *Hourglass Multimedia Content and Service Composition Framework for Smart Room Environments.* Elsevier Journal on Pervasive and Mobile Computing, vol. 1, no. 1, pages 43–75, March 2005.

[netem 10] netem. *The netem Homepage.* Online, `http://www.linuxfoundation.org/en/Net:Netem`, 2010.

[Netfilter 10] Netfilter. *The Netfilter/iptables Project Homepage.* Online, `http://www.netfilter.org/`, 2010.

[Niedermeier 03] Christoph Niedermeier, Reiner Schmid, Changpeng Fan, David Carlson, Andreas Schrader, Andreas Kassler & Andreas Schorr. *MASA - A Scalable QoS Framework.* In Proceedings of the 7th IASTED International Conference on Internet and Multimedia Systems and Applications (IMSA 2003), Honolulu, Hawaii, USA, August 2003.

[Nielsen 94] Jakob Nielsen. Usability Engineering. Morgan Kaufmann, 1994.

[Noble 97] Brian D. Noble, M. Satyanarayanan, Dushyanth Narayanan, James Eric Tilton, Jason Flinn & Kevin R. Walker. *Agile Application-Aware Adaptation for Mobility.* In Proceedings of the 16th ACM Symposium on Operating Systems Principles (SOSP 1997), pages 276–287, Saint-Malo, France, October 1997.

[Nokia 04] Nokia. *Quality of Experience (QoE) of Mobile Services: Can it Be Measured and Improved?* White paper, November 2004.

[Oliveira 00] Manuel M. Oliveira, Gary Bishop & David McAllister. *Relief Texture Mapping.* In Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH 2000), pages 359–368, New Orleans, Louisiana, USA, July 2000.

[Perkis 06] Andrew Perkis, Solveig Munkeby & Odd I. Hillestad. *A Model for Measuring Quality of Experience.* In Proceedings of the 7th Nordic Signal Processing Symposium (NORSIG 2006), pages 198–201, Reykjavik, Iceland, June 2006.

[Postel 80] Jon Postel. *User Datagram Protocol.* RFC 768, Internet Engineering Task Force, August 1980. `http://www.ietf.org/rfc/rfc768.txt`.

[Postel 81a] Jon Postel. *Internet Protocol.* RFC 791, Internet Engineering Task Force, September 1981. `http://www.ietf.org/rfc/rfc791.txt`.

[Postel 81b] Jon Postel. *Transmission Control Protocol.* RFC 793, Internet Engineering Task Force, September 1981. `http://www.ietf.org/rfc/rfc793.txt`.

[Quax 03] Peter Quax, Tom Jehaes, Pieter Jorissen & Wim Lamotte. *A Multi-User Framework Supporting Video-Based Avatars.* In Proceedings of the 2nd Workshop on Network and System Support for Games (NetGames 2003), pages 137–147, Redwood City, California, USA, May 2003.

[Quax 04] Peter Quax, Chris Flerackers, Tom Jehaes & Wim Lamotte. *Scalable Transmission of Avatar Video Streams in Virtual Environments.* In Proceedings of the IEEE International Conference on Multimedia and Expo (ICME 2004), pages 631–634, Taipei, Taiwan, June 2004.

[Quax 05a] Peter Quax, Tom Jehaes, Maarten Wijnants & Wim Lamotte. *Mobile Adaptations for a Multi-User Framework Supporting Video-Based Avatars.* In Proceedings of the 9th IASTED International Conference on Internet and Multimedia Systems and Applications (IMSA 2005), pages 412–417, Honolulu, Hawaii, USA, August 2005.

[Quax 05b] Peter Quax, Maarten Wijnants, Tom Jehaes & Wim Lamotte. *Bridging the Gap between Fixed and Mobile Access to a Large-Scale NVE Incorporating Both Audio and Video.* In Proceedings of the IASTED International Conference on Web Technologies, Applications, and Services (WTAS 2005), Calgary, Canada, July 2005.

[Quax 07] Peter Quax. *An Architecture for Large-scale Virtual Interactive Communities.* PhD thesis, Hasselt University, Expertise centre for Digital Media (EDM), 2007.

[Rakocevic 01] Veselin Rakocevic, John Griffiths & Graham Cope. *Performance Analysis of Bandwidth Allocation Schemes in Multiservice IP Networks using Utility Functions.* In Proceedings of the 17th International Teletraffic Congress (ITC 2001), Salvador da Bahia, Brazil, December 2001.

[Ramanathan 01] Ananthanarayanan Ramanathan & Manish Parashar. *Active Resource Management for The Differentiated Services Environment.* In Proceedings of the 3rd Annual International Workshop on Active Middleware Services (AMS 2001), pages 78–86, San Francisco, California, USA, August 2001.

[Rejaie 01] Reza Rejaie & Jussi Kangasharju. *Mocha: A Quality Adaptive Multimedia Proxy Cache for Internet Streaming.* In Proceedings of the 11th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV 2001), pages 3–10, Port Jefferson, New York, USA, June 2001.

[Rho 05] Seungmin Rho, Eenjun Hwang & Minkoo Kim. *An Implementation of QoS Adaptive Multimedia Content Delivery.* In Proceedings of the 9th IASTED International Conference on Internet and Multimedia Systems and Applications (IMSA 2005), pages 316–321, Honolulu, Hawaii, USA, August 2005.

[Rosen 01] Eric C. Rosen, Arun Viswanathan & Ross Callon. *Multiprotocol Label Switching Architecture.* RFC 3031, Internet Engineering Task Force, January 2001. `http://www.ietf.org/rfc/rfc3031.txt`.

[Saltzer 84] Jerome H. Saltzer, David P. Reed & David D. Clark. *End-to-End Arguments in System Design.* ACM Transactions on Computer Systems, vol. 2, no. 4, pages 277–288, November 1984.

[Schill 99] Alexander Schill, Sascha Kümmel, Thomas Springer & Thomas Ziegert. *Two Approaches for an Adaptive Multimedia Transfer Service for Mobile Environments.* Computers & Graphics, vol. 23, no. 6, pages 849–856, December 1999.

[Schneider 99] Bengt-Olaf Schneider & Ioana M. Martin. *An Adaptive Framework for 3D Graphics over Networks.* Computers & Graphics, vol. 23, no. 6, pages 867–874, December 1999.

[Schulzrinne 03] Henning Schulzrinne, Stephen L. Casner, Ron Frederick & Van Jacobson. *RTP: A Transport Protocol for Real-Time Applications.* RFC 3550, Internet Engineering Task Force, July 2003. `http://www.ietf.org/rfc/rfc3550.txt`.

[Second Life 10] Second Life. *The Second Life Official Website.* Online, `http://secondlife.com/`, 2010.

[Sels 09] Olivier Sels. Beheren van een Netwerk van Proxy Servers. Bachelor's thesis, Hasselt University, June 2009.

[Siller 03a] Mario Siller & John Charles Woods. *Improving Quality of Experience for Multimedia Services by QoS Arbitration on a QoE Framework.* In Proceedings of the 13th IEEE Packet Video Workshop (PV 2003), Nantes, France, April 2003.

[Siller 03b] Mario Siller & John Charles Woods. *QoS Arbitration for Improving the QoE in Multimedia Transmission.* In Proceedings of the International Conference on Visual Information Engineering (VIE 2003), pages 238–241, Guildford, United Kingdom, July 2003.

[Simoens 07] Pieter Simoens, Bart De Vleeschauwer, Wim Van de Meerssche, Filip De Turck, Bart Dhoedt, Piet Demeester, Tom Van Caenegem, Kris Struyve, Hans Dequeker & Edith Gilon. *RTP Connection Monitoring for Enabling Autonomous Access Network QoS Management.* In Proceedings of the 11th European Conference on Networks and Optical Communications (NOC 2007), Stockholm, Sweden, July 2007.

[Singhal 99] Sandeep Singhal & Michael Zyda. Networked Virtual Environments: Design and Implementation. Addison-Wesley Professional, 1999.

[Soldani 06] David Soldani. *Means and Methods for Collecting and Analyzing QoE Measurements in Wireless Networks.* In Proceedings of the 7th IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM 2006), pages 531–535, Buffalo, New York, USA, June 2006.

[Sommerville 06] Ian Sommerville. Software Engineering. Addison-Wesley, 8th edition, 2006.

[Spirent 10] Spirent. *The Spirent Homepage.* Online, `http://www.spirent.com/`, 2010.

[Steenkiste 02] Peter Steenkiste, Prashant Chandra, Jun Gao & Umair Shah. *An Active Networking Approach to Service Customization.* In Proceedings of the DARPA Active Networks Conference and Exposition (DANCE 2002), pages 305–318, San Francisco, California, USA, May 2002.

[Subramanian 04] Lakshminarayanan Subramanian, Ion Stoica, Hari Balakrishnan & Randy Katz. *OverQoS: An Overlay based Architecture for Enhancing Internet QoS.* In Proceedings of the 1st Symposium on Networked Systems Design and Implementation (NSDI 2004), pages 71–84, San Francisco, California, USA, March 2004.

[Tanenbaum 02] Andrew S. Tanenbaum. Computer Networks. Prentice Hall PTR, 4th edition, August 2002.

[Tennenhouse 96] David L. Tennenhouse & David J. Wetherall. *Towards an Active Network Architecture.* ACM SIGCOMM Computer Communication Review, vol. 26, no. 2, pages 5–18, April 1996.

[Tennenhouse 97] David L. Tennenhouse, Jonathan M. Smith, W. David Sincoskie, David J. Wetherall & Gary J. Minden. *A Survey of Active Network Research.* IEEE Communications Magazine, vol. 35, no. 1, pages 80–86, January 1997.

[Underwood 01] Todd Underwood. *Netfilter and iptables: Stateful Firewalling for Linux.* Tech News on ZDNet, available online at `http://news.zdnet.com/2100-10532_22-296775.html`, October 2001.

[Vanhaverbeke 09] Frederik Vanhaverbeke, Marc Moeneclaey, Koen Laevens, Natalie Degrande & Danny De Vleeschauwer. *Video Quality Protection Strategies for HDTV in the Presence of Buffer Overflow.* In Proceedings of the 8th International Conference on Networks (ICN 2009), pages 342–346, Cancun, Mexico, March 2009.

[Vetro 03] Anthony Vetro, Charilaos Christopoulos & Huifang Sun. *Video Transcoding Architectures and Techniques: An Overview.* IEEE Signal Processing Magazine, vol. 20, no. 2, pages 18–29, March 2003.

[Vetro 05] Anthony Vetro & Christian Timmerer. *Digital Item Adaptation: Overview of Standardization and Research Activities.* IEEE Transactions on Multimedia, vol. 7, no. 3, pages 418–426, June 2005.

[Wetherall 99] David J. Wetherall, John Guttag & David L. Tennenhouse. *ANTS: Network Services without the Red Tape.* IEEE Computer, vol. 32, no. 4, pages 42–48, April 1999.

[White 97] Paul P. White. *RSVP and Integrated Services in the Internet: A Tutorial.* IEEE Communications Magazine, vol. 35, no. 5, pages 100–106, May 1997.

[Wijnants 05a] Maarten Wijnants & Wim Lamotte. *Audio and Video Communication in Multiplayer Games through Generic Networking Middleware.* In Proceedings of the 7th International Conference on Computer Games (CGAMES 2005), pages 52–58, Angoulême, France, November 2005.

[Wijnants 05b] Maarten Wijnants, Patrick Monsieurs & Wim Lamotte. *Improving the User Quality of Experience by Incorporating Intelligent Proxies in the Network.* Technical Report TR-UH-EDM-0502, Expertise centre for Digital Media (EDM), April 2005. `http://research.edm.uhasselt.be/~mwijnants/pdf/wijnantsTRMSAN.pdf`.

[Wijnants 05c] Maarten Wijnants, Patrick Monsieurs, Peter Quax & Wim Lamotte. *Exploiting Proxy-Based Transcoding to Increase the User Quality of Experience in Networked Applications.* In Proceedings of the 1st International Workshop on Advanced Architectures and Algorithms for Internet DElivery and Applications (AAA-IDEA 2005), pages 73–80, Orlando, Florida, USA, June 2005.

[Wijnants 06] Maarten Wijnants, Bart Cornelissen, Wim Lamotte & Bart De Vleeschauwer. *An Overlay Network Providing Application-Aware Multimedia Services.* In Proceedings of the 2nd International Workshop on Advanced Architectures and Algorithms for Internet DElivery and Applications (AAA-IDEA 2006), Pisa, Italy, October 2006.

[Wijnants 07] Maarten Wijnants & Wim Lamotte. *The NIProxy: a Flexible Proxy Server Supporting Client Bandwidth*

Management and Multimedia Service Provision. In Proceedings of the 8th IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM 2007), Helsinki, Finland, June 2007.

[Wijnants 08a] Maarten Wijnants, Tom Jehaes, Peter Quax & Wim Lamotte. *Efficient Transmission of Rendering-Related Data Using the NIProxy.* In Proceedings of the IASTED International Conference on Internet and Multimedia Systems and Applications (EuroIMSA 2008), Innsbruck, Austria, March 2008.

[Wijnants 08b] Maarten Wijnants & Wim Lamotte. *Managing Client Bandwidth in the Presence of Both Real-Time and non Real-Time Network Traffic.* In Proceedings of the 3rd IEEE International Conference on COMmunication System softWAre and MiddlewaRE (COMSWARE 2008), Bangalore, India, January 2008.

[Wijnants 08c] Maarten Wijnants, Wim Lamotte, Bart De Vleeschauwer, Filip De Turck, Bart Dhoedt, Piet Demeester, Peter Lambert, Dieter Van de Walle, Jan De Cock, Stijn Notebaert & Rik Van de Walle. *Optimizing User QoE through Overlay Routing, Bandwidth Management and Dynamic Transcoding.* In Proceedings of the 2nd International Workshop on Adaptive and DependAble Mobile Ubiquitous Systems (ADAMUS 2008), Newport Beach, California, USA, June 2008.

[Wijnants 09a] Maarten Wijnants & Wim Lamotte. *Effective and Resource-Efficient Multimedia Communication Using the NIProxy.* In Proceedings of the 8th International Conference on Networks (ICN 2009), pages 266–274, Cancun, Mexico, March 2009.

[Wijnants 09b] Maarten Wijnants & Wim Lamotte. *FEC-Integrated Network Traffic Shaping Using the NIProxy.* In Proceedings of the 1st International Conference on Emerging Network Intelligence (EMERGING 2009), pages 51–60, Sliema, Malta, October 2009.

[Wijnants 10] Maarten Wijnants, Wim Lamotte, Bart De Vleeschauwer, Filip De Turck, Bart Dhoedt, Piet Demeester, Peter Lambert, Dieter Van de Walle, Jan De Cock, Stijn Notebaert & Rik Van de Walle. *Optimizing User Quality of Experience through Overlay Routing, Bandwidth Management and Dynamic Transcoding.* Special Issue of the International Journal of Adaptive, Resilient and Autonomic Systems (IJARAS) on the Adaptive and Dependable Mobile Ubiquitous Systems (ADAMUS) Workshop, 2010. In Press.

[Wolski 97a] Rich Wolski. *Forecasting Network Performance to Support Dynamic Scheduling Using the Network Weather Service.* In Proceedings of the 6th IEEE International Symposium on High Performance Distributed Computing (HPDC 1997), pages 316–325, Portland, Oregon, USA, August 1997.

[Wolski 97b] Rich Wolski, Neil Spring & Chris Peterson. *Implementing a Performance Forecasting System for Metacomputing: The Network Weather Service.* In Proceedings of the ACM/IEEE Conference on Supercomputing (SC 1997), San Jose, California, USA, November 1997.

[Woodrow 04] Chris Woodrow, Johan Hjelm, Hidetaka Ohto, Luu Tran, Franklin Reynolds, Graham Klyne & Mark H. Butler. *Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies 1.0.* W3C Recommendation, W3C, January 2004. `http://www.w3.org/TR/2004/REC-CCPP-struct-vocab-20040115/`.

[Wu 09] Wanmin Wu, Ahsan Arefin, Raoul Rivas, Klara Nahrstedt, Renata Sheppard & Zhenyu Yang. *Quality of Experience in Distributed Interactive Multimedia Environments: Toward a Theoretical Framework.* In Proceedings of the 17th ACM International Conference on Multimedia (MULTIMEDIA 2009), pages 481–490, Beijing, China, October 2009.

[Yang 06] Zhenyu Yang, Bin Yu, Klara Nahrstedt & Ruzena Bajcsy. *A Multi-stream Adaptation Framework for Bandwidth Management in 3D Tele-immersion.* In Proceedings of the 16th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV 2006), Newport, Rhode Island, USA, May 2006.

[Zenel 99] Bruce Zenel. *A General Purpose Proxy Filtering Mechanism Applied to the Mobile Environment.* Wireless Networks, vol. 5, no. 5, pages 391–409, October 1999.

[Zhang 04] Xiaolan Zhang, Michael Bradshaw, Yang Guo, Bing Wang, Jim Kurose, Prashant Shenoy & Don Towsley. *AMPS: A Flexible, Scalable Proxy Testbed for Implementing Streaming Services.* In Proceedings of the 14th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV 2004), pages 116–121, Kinsale, Ireland, June 2004.

[Zheng 06] Yongjie Zheng, Alvin T. S. Chan & Grace Ngai. *Applying Coordination for Service Adaptation in Mobile Computing.* IEEE Internet Computing, vol. 10, no. 5, pages 61–67, September-October 2006.