T E C H N I C A L
R E P O R T

TR-UH-EDM-0502

IMPROVING THE USER QUALITY OF EXPERIENCE BY INCORPORATING INTELLIGENT PROXIES IN THE NETWORK

Maarten Wijnants, Patrick Monsieurs and Wim Lamotte

Expertise Centre for Digital Media
Limburgs Universitair Centrum
Wetenschapspark 2
B-3590 Diepenbeek
`http://www.edm.luc.ac.be/`

Improving the User Quality of Experience by Incorporating Intelligent Proxies in the Network

Maarten Wijnants, Patrick Monsieurs and Wim Lamotte

Abstract

Despite recent advances in computer hardware and the emergence of broadband access networks, client machines are often still not capable of receiving and processing all data generated in networked multimedia applications. Most current systems try to solve this problem by leveraging high-level knowledge of the application to reduce the amount of information clients need to process. However, network related information is often neglected when making such decisions, which can result in a non-optimal Quality of Experience for users. In this work, we propose to insert generic software proxies in the network that are both application and network aware, and we describe how their compound awareness can be exploited to intelligently manage and adapt data streams in networked applications. We also present results from the integration of our work into an existing Networked Virtual Environment. These results clearly indicate that our proxy system can improve the user QoE considerably.

keywords H.4.3 [Information Systems Applications]: Communications Applications

1 Introduction

Networked multi-user applications incorporating audio, video, and/or other sorts of media tend to produce large quantities of data that need to be transmitted over a computer network. Examples of such applications include Massively Multiplayer Online Games (MMOGs), Networked Virtual Environments (NVEs) and videoconferencing programs. Although both the processing power of home computers and the bandwidth of access networks have increased tremendously the past few years, client machines are often not capable of receiving and processing all these data. Therefore, an appeal is usually made to an awareness management model to limit the amount of information clients need to process. Most awareness management models try to accomplish this goal by identifying the data that is most relevant to a given user. Only these data are subsequently transmitted to the user in question.

In nearly all current networked multimedia applications, the awareness management model is part of the client software. This sounds reasonable, since the client software normally is *application aware*: it has high-level knowledge of the application. For example, in case of an NVE, the client software usually knows the position and orientation of the local user and those of other connected users in the virtual world. It should be apparent that this is valuable information for an awareness management model. However, the client software usually is not *network aware*: it has very little or even completely no knowledge of the current state of the network that connects the local user with application servers and/or other users. For example, a typical client has no notion whatsoever of the current throughput, latency or packet loss rate of individual network links. In order to maximize the user's Quality of Experience (QoE) in networked multimedia applications, both application and network awareness should be exploited.

In this work, we propose to insert software proxies in the network that are both application and network aware. When connecting to the application, each client is associated with a proxy in its vicinity (from a network point of view). From this point on, all data streams originating from and destined for a certain client should pass through his proxy. Each proxy periodically probes the network links going to the clients it is currently serving. The clients on the other hand provide their proxy with knowledge of the application. Based on its compound awareness, the proxy then intelligently filters, routes and possibly transcodes data streams that reside in the system. In other words, the proxies can be seen as intelligent network nodes that manage and adapt network traffic. We believe that making the network more intelligent this way can improve the user QoE considerably in many networked multimedia applications.

The general setup of our network architecture has already been discussed in [11]. This paper more specifically addresses the implementation and evaluation of our proxy system, and is organized as follows. We start by reviewing related work in section 2. In section 3 we discuss the implementation of our proxy system and demonstrate how our proxies gain application and network awareness. Sections 4 and 5 describe our findings from integrating our work into an existing NVE. Finally, we present our conclusions and suggest possible future research directions in section 6.

2 Related Work

Due to the limitations of computer hardware, awareness management is one of the main enabling technologies of networked multi-user applications. As a result, it has already received a lot of attention from the research community, especially in the context of NVEs. The awareness management model of the RING system [6] is based on line-of-sight visibility. SPLINE [2] on the other hand partitions the virtual world into spatial regions or *locales* which each have their own communication channel. By only subscribing to the communication channels of the locales they are interested in, clients can reduce the amount of information they need to receive and process. A final awareness management model for NVEs worth mentioning is the *spatial model of interaction* [4]. This model uses concepts such as aura, focus and nimbus to mediate interactions in a virtual environment.

Nearly all current networked applications employ either a client/server or a peer-to-peer network architecture [15][7]. In a client/server architecture, clients can only communicate with each other through one or possibly a small number of centralized server machines (see figure 1(a)). The main advantage of this topology is its simplicity: the implementation of billing, security, authentication and other policies is fairly easy in a client/server system. Furthermore, servers can offer their clients valuable services, like intelligent packet forwarding and packet aggregation. However, the drawbacks of the client/server architecture are its lack of robustness, its lack of scalability, and the

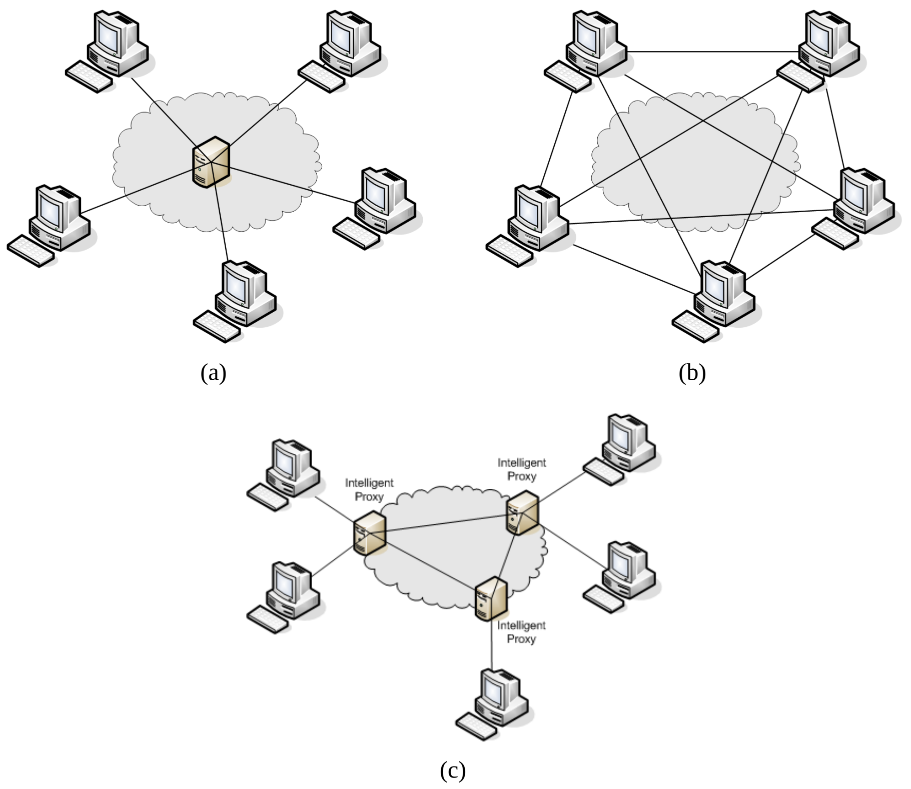


Figure 1: Possible topologies for networked applications: (a) Client/server; (b) Peer-to-peer; (c) Distributed proxy architecture.

fact that routing packets through a server introduces extra delay for clients. In a peer-to-peer topology on the other hand, all connected users are considered to be equal and can communicate directly with each other (see figure 1(b)). Since there is no central server, there is no single point-of-failure in the system and the packet delay is reduced. Unfortunately, managing and controlling a peer-to-peer system is relatively difficult. Furthermore, a peer-to-peer system can require a lot of access bandwidth from clients. Finally, maintaining consistency is much harder than in a client/server architecture.

Since there are issues associated with both the pure client/server and peer-to-peer topology, other possible architectures for networked applications have been investigated. Recently, this resulted in the introduction of the distributed proxy architecture. In this architecture, a number of interconnected proxies are placed at certain locations in the network, and clients only communicate with the proxy nearest to them (see figure 1(c)). The state of the networked application is maintained by either the proxies themselves or by a central server. As discussed in [3] and [10], the distributed proxy architecture enables congestion control and enhances the robustness of the system. Furthermore, proxies can provide services like caching (this way reducing packet delay), packet aggregation and intelligent routing. In [5], the results of porting a popular multiplayer FPS game to the distributed proxy architecture are described. Nguyen et al. investigate in [12] how the distributed proxy architecture can be exploited to provide immersive audio communication to users of online games.

The objective of content-based transcoding is to enable universal access of multimedia content by adapting it to the constraints of both the transportation network and the receiving client device. This relieves content suppliers from having to provide multiple specialized versions of the same content. In [16], an early content-based image transcoder is described which takes the capabilities of the receiving client device and the type and purpose of the image into account. Shen et al. propose in [14] to insert *Transcoding-enabled Caching proxies* (TeC proxies) in the content delivery path to improve the efficiency of video delivery to heterogeneous end-users with various network conditions and client capabilities. In contrast with our intelligent proxies however, the TeC proxies are not application aware. Knutsson et al. describe in [8] the concept of *server-directed transcoding*, apply it to web content, and show how it can be integrated into the HTTP protocol. In the server-directed transcoding approach, the origin server guides the transcoder to make sure the content does not lose its semantics during the transcoding process. This can be seen as a simple case of making the transcoder more aware of the application it is serving, but is not as general as our approach. Finally, [9] discusses the implementation of a video transcoding gateway that aims at enabling video access on mobile devices. Our proxy system very closely resembles the described transcoding gateway, but is more generally applicable since it is capable of managing and adapting *every* kind of data stream, while the gateway is restricted to transcoding video.

3 Implementation

3.1 Network Intelligence Layer

To make our proxies more aware of the application they are serving, we suggest inserting a *network intelligence layer* between the transport layer and the application layer of the client software. This network intelligence layer constantly queries the application's awareness management model in order to obtain information regarding the relative importance of each data stream present in the application. Possible types of data streams in networked multi-user applications include position and orientation updates, geometry, audio, video, etcetera. The application's awareness management model will most likely indicate that certain types of streams have a higher priority than others. For example, in case of an NVE, the awareness manager could indicate that object position and orientation updates are more important than audio or video streams. However, an awareness management model will normally also differentiate between data streams of the same type. Again in case of an NVE, the awareness manager could for instance specify that data streams belonging to objects which are located close to the local user in the virtual world have a higher relative importance than streams belonging to more distant objects. The network intelligence layer is responsible for retrieving such application related information and communicating it to the proxy this client is connected to.

3.2 Network Sensing

Each proxy in the system periodically probes the current latency and throughput of the network links going to the clients currently connected to it. To measure the proxy-to-client latency, the proxy records the time that passes between sending an arbitrarily small probe to the client and receiving an acknowledgment for it. This amount of time is actually the round-trip time; dividing it by two yields an estimate of the proxy-to-client latency. To measure the throughput of a network link, the proxy times how long it takes to send a packet of significant size (e.g. 1000 bytes) to the client and to receive an acknowledgment for it. The throughput can then be calculated by dividing the used probe size by the recorded time minus the measured round-trip time.

Furthermore, for each client connected to it, the proxy also stores the total number of probes sent to this client and the number of probes not acknowledged by the client before a certain time-out interval (meaning either the probe got lost in the network, or the acknowledgment did). These statistics give the proxy an estimate of the packet loss rate of each client's network link. Finally, our proxies record the individual bandwidth usage of every data stream that passes through them. Not only different types of streams (e.g. audio versus video), but also different streams of the same type (e.g. video streams encoded with different quality settings) can require unequal amounts of bandwidth.

3.3 Distributing the Client's Available Bandwidth

Our proxies continuously combine their application and network awareness to devise a strategy which optimally distributes each client's available bandwidth over the different streams present in the system. For example, when a client's available downstream bandwidth no longer suffices to receive all streams at maximum quality, the proxy will determine which streams should be reduced in quality or even completely blocked. This decision will be based on the relative importance of the streams as well as their bandwidth requirements. As a result, whenever possible, less important streams will be transcoded or dropped by the proxy before more significant streams are altered, this way maximizing the user's QoE. The algorithm used to compute the bandwidth allocation strategies is described in full detail in [11].

3.4 Packet Filtering, Routing and Mangling

Based on the devised client bandwidth allocation strategies, each proxy in the system filters, routes and possibly transcodes data streams originating from and destined for its clients. To implement this functionality, we relied on the netfilter/iptables framework [1] which is part of the linux kernel since version 2.4. Netfilter allows kernel modules to register callback functions for the different *hooks* defined by each network stack. Whenever a packet arrives at one of these hooks, netfilter checks if anyone has registered a callback function for it. If this is the case, the packet is handed over to this function, which can then examine and possibly alter it. IPv4 for example defines five hooks; three of them are depicted in the packet traversal diagram shown in Fig. 2.

Iptables is a generic table structure built on top of netfilter that enables packet selection in userspace. A table in iptables consists of a number of *chains*, which in turn are checklists of rules. A rule specifies what should be done with a packet if its header matches the conditions specified in the rule. Possible verdicts a rule can issue for a packet include ACCEPT (let the packet pass), DROP (discard the packet) and QUEUE (hand the packet over to a userspace application which will decide on its

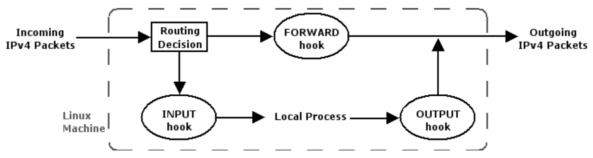


Figure 2: IPv4 packet traversal diagram.

fate). If a packet does not match the conditions of the current rule, the next rule in the chain is consulted. The linux IPv4 filter table contains three chains, one for each of the three hooks depicted in Fig. 2. Each time a packet arrives at one of these hooks, the rules in the corresponding chain are consulted to determine the fate of the packet.

Our proxies use iptables to do some trivial packet filtering directly (e.g. always issue an ACCEPT verdict for network probes), and to QUEUE packets belonging to data streams of the served application to a userspace program. The userspace application subsequently consults the computed bandwidth allocation strategy for the client this stream applies to, and leverages this information to do intelligent routing, packet filtering and data stream transcoding. For example, due to a momentary lack of bandwidth, the bandwidth allocation strategy could specify that this client is no longer interested in a certain audio channel, or that he would like a lower quality version of a video stream. This strategy will subsequently be put into effect by the userspace application.

To achieve maximal usability, we have tried to keep our proxies as generic as possible. As a result, the standard version of the proxy can execute only a limited number of basic operations on data streams. However, the performance and functionality of the proxy can be increased by installing plug-ins. Application designers can write their own plug-in that exploits specific knowledge of the application, and install it in the proxy. This approach ensures that our proxies can be integrated in nearly all networked applications.

4 Integration into an Existing NVE

To test our work, we integrated it into our in-house developed multi-user NVE framework which was first introduced in [13]. This framework relies extensively on multicast communication and moves responsibilities as much as possible from the server(s) to the clients in order to achieve scalability. Furthermore, the framework also tries to increase the immersive experience for connected users by supporting *video-based avatars*, a technique in which the face of the user is continuously captured with a webcam and subsequently textured on his avatar (see figure 3).

The framework divides the offered virtual environment into square regions which each have a distinct multicast address associated with them, much like the locales approach discussed in section 2. Whenever something happens in the virtual world, information about the event is sent only to the multicast address of the region from which the event originated. Each client has an Area of Interest (AoI) manager which constantly determines the regions this client should be aware of. Clients subsequently only subscribe to the multicast groups associated with the selected regions, this way limiting the amount of information they need to receive and process. However, the framework also divides the world into video regions. Each user represented by a

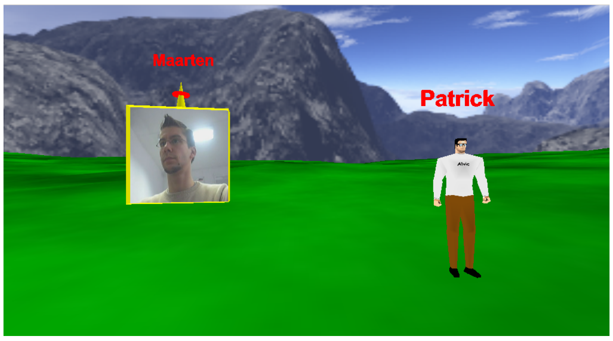


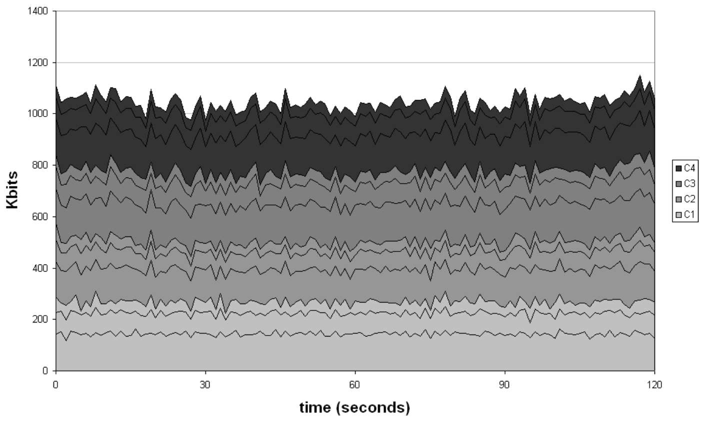
Figure 3: Screenshot of our in-house developed multi-user NVE framework supporting video-based avatars.

video-based avatar in the virtual world continuously sends three separate qualities (low, medium and high) of his video stream to the three multicast addresses associated with the video region he currently is located in. Like the general AoI manager, the Video Area of Interest (VAoI) manager decides which video multicast addresses the client should subscribe to. One strategy could be to subscribe to the high quality multicast group of the video region the user currently is in, and to the medium or even low quality groups of adjacent video regions.

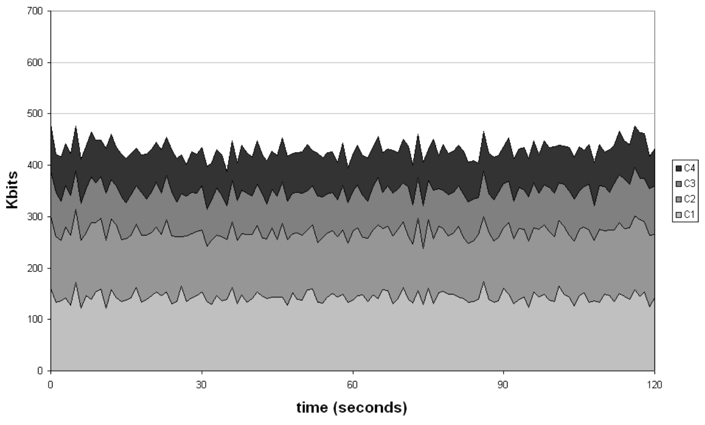
Integrating the network intelligence layer into the framework's client software proved to be quite straightforward. It sufficed to create a link between the network intelligence layer and the AoI and VAoI managers. However, we did have to implement a specific network traffic adaptation plug-in for our proxies. Besides managing data streams based on specific knowledge of the framework, this plug-in also acts as a multicast reflector for clients connected to the proxy. Instead of multicasting a packet themselves, clients can unicast the packet to their proxy which will subsequently exploit its application awareness to multicast the packet to the correct multicast group. We included this feature in the plug-in because sending multicast packets is often still not supported in access networks. On the other hand, the plug-in also dynamically joins and leaves multicast groups on behalf of its clients and converts multicast streams that reside on the inter-proxy network to unicast packets destined for interested clients. By doing so, the plug-in ensures that the information contained in a multicast stream only reaches clients that are actually interested in it, and that no bandwidth of uninterested clients is wasted.

5 Experimental Results

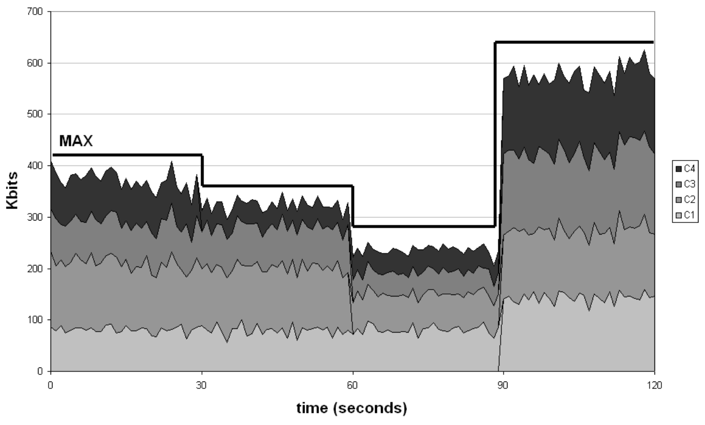
We performed several experiments to assess the validity of our approach in the context of the NVE framework. The majority of these tests focused on filtering video streams since these are the predominant streams in the system in terms of bandwidth usage. Two such experiments are described below. However, we would like to emphasize here that our work was not written with this specific framework in mind. We have tried to keep our network intelligence layer and our proxies as generic as possible so that they can be incorporated in any networked application.



(a) Promiscuous Mode

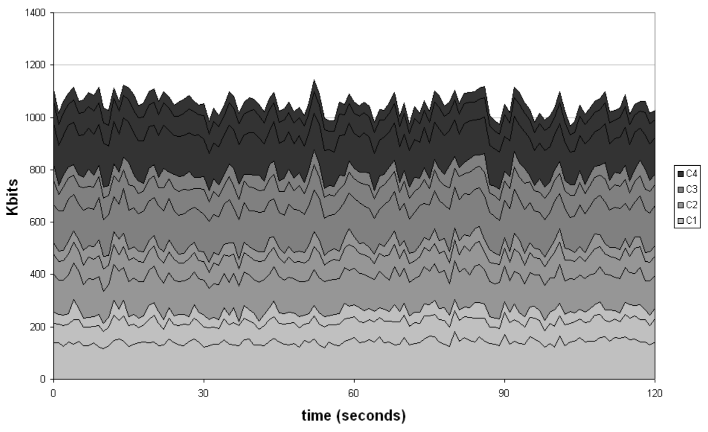


(b) Unmodified version of the framework

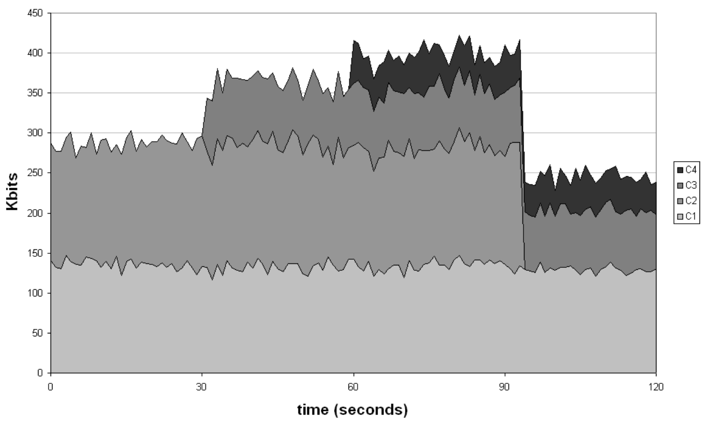


(c) Proxy-enhanced framework

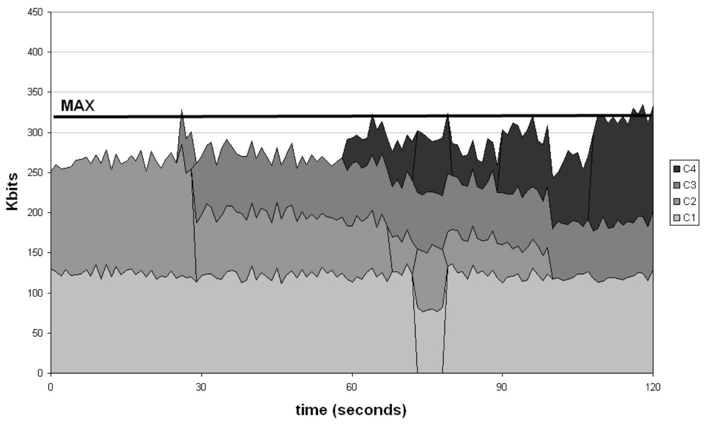
Figure 4: Network traces (stacked graphs) of an experiment in which we varied a user's downstream bandwidth over time.



(a) Promiscuous Mode



(b) Unmodified version of the framework



(c) Proxy-enhanced framework

Figure 5: Network traces (stacked graphs) of an experiment in which we simulated a dynamic conversation between users.

In the first experiment, we artificially varied the downstream bandwidth a connected user had at his disposal over time. As can be seen in the network trace shown in Fig. 4(c), this user constantly had four other clients (C1 to C4) in his VAoI. All involved clients remained stationary in the virtual world for the entire duration of the experiment. At the beginning of the experiment, we set the user's downstream bandwidth to 420 Kbit per second. The proxy of the traced user noticed that this bandwidth did not suffice to receive the high quality video streams of all four clients. In such situations, the proxy leverages its application awareness to decide which video quality the user should receive from each of the clients in his VAoI. In this case, the proxy decided to give the most importance to client C2 (e.g. because this client was located closer to the traced user in the virtual world than C1, C3 and C4). As a result, the traced user received the high quality video stream of client C2, but only the medium quality video streams of clients C1, C3 and C4. After 30 seconds, we limited the user's bandwidth to 360 Kbps, and after 60 seconds we further decreased it to 280 Kbps. As can be seen in the network trace, this resulted in the user receiving more medium and even low quality video streams. Finally, after 90 seconds, we increased the bandwidth back to 640 Kbps which allowed the user to receive the high quality video streams of all four clients.

This experiment clearly indicates how our proxy system tries to offer users a maximal QoE in the case of sudden bandwidth changes, an event not unlikely to occur in mobile networks. If the unmodified version of the framework had been used, the user would have received the same set of video streams during the entire experiment, which would have resulted in the client's available downstream bandwidth being exceeded numerous times (see Fig. 4(b)). As a result, the playback of received video streams would not have been smooth, which has a seriously negative effect on the user's immersive experience. However, it is worth noting that the VAoI manager of the unmodified version of the framework already drastically limits the amount of video data a client needs to receive. This becomes clear when we compare the network trace in Fig. 4(b) with the one in Fig. 4(a) which shows all video network traffic generated during the experiment. The VAoI manager however lacks network awareness and as a result does not take the current status of the network into account when deciding which video regions a client should subscribe to. This explains why our proxy system, which is both application *and* network aware, achieves better results.

In the second experiment, we simulated a dynamic conversation between a variable number of users, which is a common scenario in many NVEs. We kept the downstream bandwidth of the traced user fixed at 320 Kbps during the entire experiment. The conversation started off with three participants, namely the traced user and clients C1 and C2. After 30 and 60 seconds, client C3 and client C4 respectively joined the conversation. Finally, after approximately 90 seconds, client C2 decided to leave the conversation. The network traces of this experiment, which are shown in Fig. 5, demonstrate that our proxy system at all times exploits all available downstream bandwidth and dynamically adjusts the bandwidth allocated to different streams in order to provide users with a maximal QoE. Again, if the unmodified version of the framework had been used, the available bandwidth would have been exceeded numerous times, and no attempts would have been made to adjust the bandwidth usage of individual data streams based on their changing relative importance.

6 Conclusions and Future Work

In this work, we have proposed to integrate intelligent software proxies in the network that are both application and network aware. To gain network awareness, each proxy periodically probes the network links of all clients connected to it. The clients on the other hand provide their proxy with high-level knowledge of the application. Each proxy subsequently exploits its application and network awareness to increase the user QoE in networked applications by intelligently filtering, routing and transcoding data streams. We also described results from integrating our work into an existing NVE. These results confirm the effectiveness of our work and clearly demonstrate how our proxy system dynamically and intelligently adjusts the bandwidth usage of individual data streams.

In the future we would like to improve our work in several ways. First of all, our proxy could gather information regarding the capabilities of the receiving client device and leverage this knowledge to devise even more efficient bandwidth allocation strategies. For example, it is pointless to send a video stream to a client if this client at the moment has insufficient processing power available to decode it. Secondly, our proxies could probe the network more extensively and for example try to detect congestion in the inter-proxy network. Finally, we would like to perform more complex experiments, for instance by including mobile devices.

Acknowledgments

Part of this research was funded by the IWT project number 020659, Alcatel Bell, the IBBT (Interdisciplinary institute for BroadBand Technology) and the Flemish Government. Part of Maarten Wijnants' work was carried out at ANDROME NV before he moved to the Limburgs Universitair Centrum.

References

[1] The Netfilter/IPTables Project Homepage. 3.4

[2] John Barrus, Richard Waters, and David Anderson. Locales and Beacons: Efficient and Precise Support for Large Multi-User Virtual Environments. In *Proc. IEEE Virtual Reality Annual International Symposium (VRAIS'96)*, pages 204–213, Santa Clara, CA, 1996. 2

[3] Daniel Bauer, Sean Rooney, and Paolo Scotton. Network Infrastructure for Massively Distributed Games. In *Proc. 1st Workshop on Network and System Support for Games (NetGames'02)*, pages 36–43, Bruanschweig, Germany, 2002. 2

[4] Steve Benford and Lennart Fahlen. A Spatial Model of Interaction in Large Virtual Environments. In *Proc. 3rd European Conference on Computer Supported Cooperative Work (ECSCW'93)*, pages 109–124, Milano, Italy, September 1993. 2

[5] Eric Cronin, Burton Filstrup, and Anthony Kurc. A Distributed Multiplayer Game Server System. UM Course Project Report EECS589, Electrical Engineering and Computer Science Department, University of Michigan, May 2001. 2

[6] Thomas Funkhouser. RING: A Client-Server System for Multi-User Virtual Environments. In *Proc. Symposium on Interactive 3D graphics*, pages 85–92, Monterey, CA, April 1995. 2

[7] Thomas Funkhouser. Network Topologies for Scalable Multi-User Virtual Environments. In *Proc. IEEE Virtual Reality Annual International Symposium (VRAIS'96)*, pages 222–229, Santa Clara, CA, 1996. 2

[8] Bjorn Knutsson, Honghui Lu, Jeffrey Mogul, and Bryan Hopkins. Architecture and Performance of Server-Directed Transcoding. *ACM Transactions on Internet Technology*, 3(4):392–424, 2003. 2

[9] Zhijun Lei and Nicolas Georganas. Video Transcoding Gateway For Wireless Video Access. In *Proc. IEEE Canadian Conference on Electrical and Computing Engineering (CCECE'03)*, Montreal, May 2003. 2

[10] Martin Mauve, Stefan Fischer, and Jorg Widmer. A Generic Proxy System for Networked Computer Games. In *Proc. 1st Workshop on Network and System Support for Games (NetGames'02)*, pages 25–28, Bruanschweig, Germany, 2002. 2

[11] Patrick Monsieurs, Maarten Wijnants, and Wim Lamotte. Client-controlled QoS Management in Networked Virtual Environments. In *Proc. 4th International Conference on Networking (ICN'05)*, pages 268–276, to appear (available on request), Reunion Island, April 2005. 1, 3.3

[12] Cong Duc Nguyen, Farzad Safaei, and Paul Boustead. A Distributed Proxy System for Provisioning Immersive Audio Communication to Massively Multi-Player Games. In *Proc. 3rd Workshop on Network and System Support for Games (NetGames'04)*, page 166, Portland, Oregon, September 2004. 2

[13] Peter Quax, Tom Jehaes, Pieter Jorissen, and Wim Lamotte. A Multi-User Framework Supporting Video-Based Avatars. In *Proc. 2nd workshop on Network and System Support for Games (NetGames'03)*, pages 137–147, Redwood City, CA, May 2003. 4

[14] Bo Shen, Sung-Ju Lee, and Sujoy Basu. Performance Evaluation of Transcoding-enabled Streaming Media Caching System. In *Proc. 4th International Conference on Mobile Data Management*, pages 363–368, Melbourne, Australia, January 2003. 2

[15] Sandeep Singhal and Michael Zyda. *Networked Virtual Environments: Design and Implementation*. Addison-Wesley Pub Co, 1999. 2

[16] John Smith, Rakesh Mohan, and Chung-Sheng Li. Content-Based Transcoding of Images in the Internet. In *Proc. IEEE International Conference on Image Processing (ICIP'98)*, volume 3, pages 7–11, Chicago, Illinois, 1998. 2