The NIProxy: a Flexible Proxy Server Supporting Client Bandwidth Management and Multimedia Service Provision

Maarten Wijnants[†‡] Wim Lamotte[†]

[†]Hasselt University, Expertise Centre for Digital Media, Wetenschapspark 2, BE-3590 Diepenbeek, Belgium

[‡]Interdisciplinary institute for BroadBand Technology (IBBT), Expertise Centre for Digital Media, BE-3590 Diepenbeek, Belgium

e-mail: {maarten.wijnants, wim.lamotte}@uhasselt.be

Abstract

We present the NIProxy, a flexible network intermediary which aims to improve the Quality of Experience (QoE) of users of networked applications by pushing more intelligence into the network. More specifically, the NIProxy is network- as well as application-aware, meaning it has knowledge of both the transportation network and the application(s) it is serving. This dual awareness is exploited to improve user QoE in two complementary ways. First of all, the NIProxy is capable of dynamically distributing a client's available downstream bandwidth over the different network streams generated by a networked application. Secondly, the NIProxy supports multimedia service provision, meaning it can apply services on multimedia streams on behalf of its clients. An important feature of the NIProxy is that its two QoE-improving mechanisms are not isolated entities but instead can interact with each other. A comprehensive discussion of the NIProxy's software architecture is provided and the implementation of an example service, which adds video transcoding functionality to the NIProxy, is described. Finally, presented experimental results clearly demonstrate the effectiveness of our approach.

1. Introduction

Multimedia content and real-time multimedia streaming are being exploited by networked applications to an ever increasing extent. For instance, commercial multiplayer games often allow players to communicate verbally with each other through the exchange of voice streams. Other applications, and in particular many academic applications, take user communication to an even higher level by also enabling users to share the video stream captured by their webcam, this way also adding a visual aspect to user communication. Internet content delivery is another example: due to the explosive growth of the number of broadband Internet connections, accessing video information through the Internet has become very popular.

Transmitting or streaming multimedia content over a transportation network however requires a considerable amount of bandwidth. Even despite the emergence of the previously mentioned broadband Internet connections, clients do not always dispose of sufficient downstream bandwidth to receive all multimedia streams that are being exchanged as part of a networked application. When this is the case, mechanisms are needed that can maximize these clients' Quality of Experience (QoE), given their current bandwidth limitations. While these mechanisms can be implemented for every networked application separately by incorporating them in the client or server software, it is economically more favorable to provide them at an intermediate network node so that they can be exploited by multiple networked applications, even concurrently.

In this paper, we present the *NIProxy*, a flexible proxy server which provides two such QoE-increasing mechanisms. On the one hand, the NIProxy supports automatic client bandwidth management, meaning it is capable of intelligently partitioning a client's downstream bandwidth among the different network streams in which the client is interested. On the other hand, the NIProxy can also be considered as a multimedia service provision platform, i.e. it is capable of applying services on network streams containing multimedia content. Example services that could be provided include stream transcoding or the merging of multiple streams into a single stream. Both the client bandwidth management mechanism and the multimedia services can exploit network- as well as application-related information when operating. Hence the term NIProxy, which is an abbreviation for *Network Intelligence* Proxy: its major goal is to introduce more intelligence in the network so that the network can deliver multimedia content to clients in a more intelligent and efficient manner.

The remainder of this paper is organized as follows. We introduce the NIProxy in section 2 and describe its software architecture in section 3. Next, sections 4 and 5 focus on the two QoE-improving mechanisms supported by the NIProxy. In particular, in section 4 the NIProxy's client bandwidth management functionality is discussed, while section 5 describes the implementation of an example service which adds video transcoding functionality to the NIProxy. Experimental results are presented in section 6, followed by a brief review of related work in section 7. Finally, we draw our conclusions and suggest future work in section 8.

2. System overview

The NIProxy is a network intermediary to which clients need to connect if they want to leverage its features. Once connected, all network streams originating from and destined for the client will pass through its NIProxy. It is possible for clients running different networked applications to be connected to the same NIProxy simultaneously. On the other hand, it is also possible to deploy multiple instances of the NIProxy in the network. In this latter case, the different instances form an overlay network and clients can choose to which NIProxy they want to connect. However, the focus of this paper is on the NIProxy as a single entity, so we will not elaborate on the NIProxy overlay case here.

As stated in the introduction, the main goal of the NIProxy is to improve the QoE of users of networked applications by introducing more awareness in the network. Currently, the NIProxy is both network- and application-aware, meaning it has knowledge of the transportation network as well as the application(s) it is serving. In the next two paragraphs, we will discuss how this dual awareness is acquired. A concrete and practical example of how the NIProxy exploits its awareness is given in section 6.

To gain its network awareness, the NIProxy periodically probes the network links which connect clients to their proxy server. The outcome is a number of measurements, i.e. the current end-to-end throughput, latency and packet loss rate of each client's network connection. However, these measurements only provide the NIProxy with a snapshot of the state of a client's network connection at a given moment in time, which can change both rapidly and drastically (this is especially true for wireless network links). Therefore, each time the NIProxy successfully finished probing the network connection of a client, it combines the outcome with the measurements obtained during previous probing iterations to estimate the future state of a client's network connection. In addition, the NIProxy's network awareness also encompasses knowledge of the bandwidth requirements of network streams. It acquires this knowledge by monitoring the network streams that pass through it and by recording their bandwidth consumption.

To obtain its application awareness on the other hand, the NIProxy relies on a support library called the Network Intelligence Layer (NILayer). This NILayer needs to be integrated in the client software, where it needs to be interfaced with the application's awareness manager. Once incorporated, the NILayer regularly queries the awareness manager to retrieve application-related information, which it subsequently forwards to the NIProxy to which the client is connected. The information extracted from the awareness manager is application-specific, since it can differ depending on the networked application in which the NILayer has been integrated. The NILayer has been designed to be highly reusable, meaning it can provide the NIProxy with awareness of many different types of networked applications.

Although there are no constraints on the network location where the NIProxy can be deployed, optimal results will be achieved if it is located close to end-users, and more specifically at junction points where network performance degrades significantly. An example of such a junction point is the boundary between the core of the network and the access network (the so-called "last-mile" of the network). In particular, while the core of the network is usually sufficiently capacitated to transport all network streams in which a particular client is interested, the same is not necessarily true for the access network. Analogously, another optimal location to deploy the NIProxy is at the transition from a wired to a wireless network. By placing the NIProxy at such junction points, it can 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.

3. Software architecture

The NIProxy presented in this paper can be considered to be the successor of the proxy server reported on in [13] and [12]. Practical experiments with the original proxy server revealed a number of limitations and shortcomings, which could nearly all be attributed to its software architecture. Consequently, while the majority of the ideas and principles underlying the original proxy server where retained when designing the NIProxy, we opted to completely rework its software architecture to make it more flexible and powerful. In this section, we will first discuss the new software architecture, after which we will highlight its improvements by comparing it with the software architecture of the original proxy server.

A schematic overview of the NIProxy's software architecture is given in figure 1. As is illustrated in this figure, the software architecture on the one hand comprises a number of static components that are not related to a particular client. Examples include the Network Probing module and the Client Administration module. On the other hand, the NIProxy also maintains a number of *packet processing chain* instances, one for each client that is currently connected to it - we will call these *proxy clients* from now on. Before possibly being forwarded, network packets arriving at the NIProxy need to pass through the packet processing chain of the proxy client for which they are destined. During this passage, packets are accompanied by a description of the network stream to which they belong.

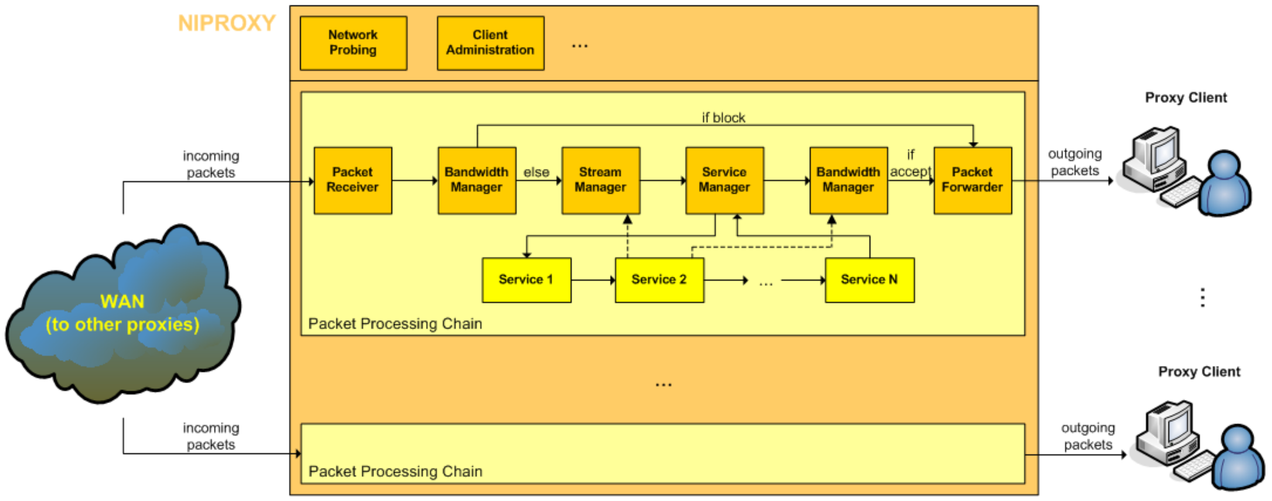


Figure 1. Overview of the NIProxy's software architecture. Each instance of the packet processing chain and of its comprising components, including services, is related to a particular proxy client.

The packet processing chain consists of the following software components[1]:

Packet Receiver: ensures that all network packets belonging to streams in which the proxy client is interested are handed to that client's packet processing chain

Bandwidth Manager: is responsible for distributing the proxy client's available downstream bandwidth over all network streams in which the client is interested

Stream Manager: registers the bandwidth usage of network streams and keeps track of stream aliveness

Service Manager: manages the loading and unloading of services on behalf of its associated proxy client and applies the currently loaded services on the network packets it receives

Packet Forwarder: is responsible for forwarding completely processed packets to the proxy client

Of these components, the Bandwidth Manager and the Service Manager are the most important ones, and they are described in more detail next. Before continuing however, it is important to note that every packet processing chain stands completely on its own. In particular, from within one packet processing chain, it is impossible to address components belonging to another packet processing chain.

The Bandwidth Manager has access to the NIProxy's bandwidth distribution algorithm, which will be described in section 4. Every time the Bandwidth Manager is handed a network packet, it consults this algorithm to compute a verdict for it. At the moment, the Bandwidth Manager defines three possible packet verdicts: block, accept and drop. A block verdict indicates that the packet belongs to a network stream which the bandwidth distribution algorithm does not know yet. An accept verdict on the other hand indicates that the network stream to which the packet belongs is known to the bandwidth distribution algorithm, and that the algorithm in addition has decided to allocate client bandwidth to this stream. Finally, a drop verdict indicates exactly the opposite, i.e. the packet belongs to a network stream which should currently not be allocated proxy client bandwidth.

As can be seen in figure 1, the Bandwidth Manager actually appears twice in the packet processing chain. The first time it is accessed, it simply ensures that a blocked packet does not continue its traversal of the packet processing chain. Instead, a description of the network stream to which the packet belongs is generated, which is subsequently sent to the proxy client by the Packet Forwarder. As will be discussed in section 4, it is the responsibility of the proxy client to subsequently decide how the NIProxy should treat future packets belonging to this network stream. The second occurrence of the Bandwidth Manager in the packet processing chain on the other hand guarantees that only packets for which it computed an accept verdict (i.e. packets belonging to a network stream for which the bandwidth distribution algorithm has reserved client bandwidth) are transmitted to the proxy client. Packets with a drop verdict are not handed over to the Packet Forwarder but instead are discarded at this point in the packet processing chain to ensure they do not consume any client bandwidth.

Besides managing client downstream bandwidth, the NIProxy can also apply services on multimedia streams on behalf of its connected clients. This functionality is provided by the Service Manager component. Services are im-

[1]Since the NIProxy maintains a separate packet processing chain for each proxy client, they each also have personal instances of its comprising components.

plemented as NIProxy *plug-ins*, which are loaded and unloaded dynamically by the Service Manager at the request of the proxy client. Whenever the Service Manager receives a packet, it passes it to the currently loaded services that have registered interest for the network stream to which the packet belongs[2]. The interested services are traversed in the order in which they were loaded, meaning the firstly loaded service will be the first to receive the packet, and so on. Services are allowed to alter the contents of the packets they receive (for instance, in case of a video frame, by transcoding it to a lower quality); if they do so, subsequent services will receive the altered version of the packet instead of the original version (i.e. the transcoded video frame instead of the original). As a result, when carefully geared to each other, multiple services can collaborate on a single network stream. Also note that since each proxy client has its individual instance of the Service Manager component, the NIProxy is capable of providing each client with a personalized service list that is tailored to the client's specific needs and/or constraints.

There are a number of advantages associated with our design decision to implement the NIProxy's service provision functionality through a plug-in mechanism. First of all, it simplifies and speeds up the process of adding new services to the NIProxy: providing a new service never requires that changes are made to the NIProxy's general software architecture; instead, it suffices to simply implement a new NIProxy plug-in. This also implies that extensive knowledge of the internals of the NIProxy is not required to successfully develop NIProxy services. Consequently, a second advantage of the plug-in mechanism is that it enables third-party service development. Thirdly, it allows for the provision of complex services by composing multiple, smaller services. Finally, although the plug-ins physically reside at the NIProxy, they can be conceptually thought of as being part of the networked application they are supporting. Consequently, NIProxy plug-ins need not be generic but instead can exploit application-specific knowledge and functionality. In effect, the plug-in mechanism neatly separates services from the rest of the NIProxy's software architecture, which is completely application-independent. This design increases the applicability of the NIProxy since it enables the NIProxy to provide valuable and efficient services for a wide variety of networked applications.

Compared to the software architecture of the original proxy server, the new software architecture is much more robust and flexible. Probably the most important improvement is that the NIProxy's bandwidth management component and service provision mechanism no longer are isolated entities but instead can interact with each other. As is illustrated in figure 1, NIProxy services can access both the proxy client's Stream Manager and Bandwidth Manager instances. This implies, for instance, that a service is capable of consulting and even influencing the bandwidth distribution strategy which the NIProxy has computed for its associated proxy client. As will be demonstrated in section 5, supporting interaction between services and the bandwidth distribution algorithm allows for the creation of services that have a level of performance and efficiency which could never be achieved by the original proxy server. Another advantage is that the software architecture of the NIProxy no longer depends on the netfilter/iptables framework. This framework, which provides packet filtering and packet mangling functionality on the GNU/Linux operating system [9], was exploited extensively by the original version of the proxy server. However, by relying on this framework, deployment of the original proxy server was confined to network nodes running GNU/Linux.

4. Client bandwidth management

The first QoE-improving mechanism supported by the NIProxy is dynamic client bandwidth management. The bandwidth distribution algorithm employed by the NIProxy was previously presented in [7]. Since then, we seriously refactored the algorithm to improve its robustness and performance. Consequently, compared to its original version, the algorithm is now capable of producing better and more stable results. However, the algorithm's underlying principles were left untouched. A fully detailed discussion of the internals of the algorithm is therefore not in order in this paper. We nonetheless summarize it in this section, since knowledge of the algorithm's mode of operation is required to comprehend the material described in sections 5 and 6.

The bandwidth distribution algorithm manages proxy client downstream bandwidth by arranging all network streams in which a proxy client is interested in a *stream hierarchy*. Actual network streams are always represented as leaf nodes in this hierarchy, while internal nodes implement a certain bandwidth distribution technique. Currently, three types of internal nodes are available:

Mutex: ensures at all times that at most one of its children is assigned bandwidth; a mutex is the simplest type of internal node and it is useful for, for instance, grouping together multiple qualities of the same stream

Priority: assigns bandwidth to its children according to their current priority value, i.e. bandwidth is first allocated to the child with the highest priority, any remaining bandwidth is subsequently allocated to the child with the second highest priority, and so on; this process is repeated until there is no more bandwidth available to assign, or until all child nodes have been considered (in this latter case, a certain amount of the bandwidth

[2]Services register and unregister stream interest with the Service Manager when they are loaded and unloaded, respectively.

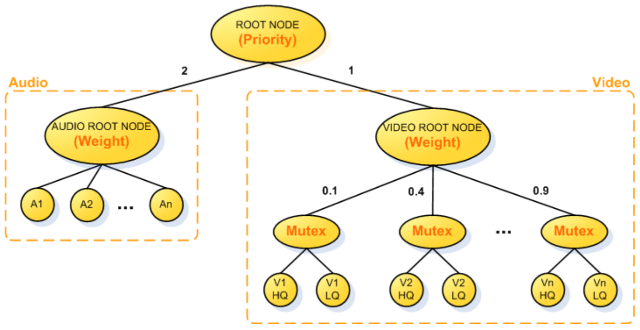


Figure 2. An example stream hierarchy.

available to the priority node remains unused); a priority node is useful for, for instance, distinguishing between different types of network streams

Weight: apportions bandwidth among its children in two consecutive passes; pass 1 ensures that the combined bandwidth usage of all child nodes does not exceed the total amount of bandwidth available to the weight node; it does so by considering all child nodes simultaneously and assigning each a part of the available bandwidth proportionally to its current weight value; any bandwidth that remains after executing pass 1 is subsequently distributed in pass 2, which assigns it to the different children one-by-one, in order of decreasing weight value; in effect, pass 2 allows child nodes to switch to a higher quality if sufficient bandwidth is still available; weight nodes are the most dynamic internal nodes and are useful for, for instance, distinguishing between network streams that are of the same type

An example stream hierarchy is illustrated in figure 2. As can be seen, the root of the hierarchy consists of a priority node and has two children, which are both of type weight. These weight nodes have priority 2 and 1 respectively, and respectively group together all audio and video streams in which the proxy client is interested. Consequently, bandwidth will first be allocated to audio; if afterwards any bandwidth remains (i.e. more bandwidth is available than all audio streams jointly consume), it is allocated to video. In this example, the networked application apparently did not define multiple audio qualities and, as a result, individual audio streams were added to the grouping audio weight node as direct children (i.e. the nodes A1 till An). On the other hand, the application transmitted both a high quality (HQ) and low quality (LQ) version of each video stream. It would be highly bandwidth-inefficient for a proxy client to receive both versions simultaneously, since they differ only in their encoding and otherwise contain exactly the same content. Therefore, before being added to the video weight node, the HQ and LQ variants of each individual video stream were grouped together using a mutex node.

It is the responsibility of the client to create and maintain its stream hierarchy at the NIProxy it is connected to. In particular, as stated in section 3, when the NIProxy receives a packet belonging to a network stream it does not know yet, a description of the stream is sent to the proxy client for which the packet was destined. The proxy client is subsequently responsible for specifying how the NIProxy should treat packets belonging to this new stream. More specifically, the client can specify that these packets should always be accepted, always dropped, or that the new stream should be integrated in the proxy client's stream hierarchy. The first option is mainly useful for application-critical network streams that do not carry multimedia content and hence have negligible bandwidth requirements (e.g. a TCP stream transporting event information). The second option allows the proxy client to specify that it has no interest whatsoever in a particular network stream. Finally, to achieve optimal results, the last option should be adhered to for all multimedia network streams in which the proxy client is interested.

Based on the above discussion, it should be apparent that the client software needs to be modified before a networked application can benefit from the NIProxy's bandwidth distribution algorithm. As was previously mentioned in section 2, we developed a reusable support library called the NILayer to facilitate and accelerate this process. This support library implements a number of application-independent, low-level NI operations and makes them available through a clear API. Consequently, application developers can exploit the NILayer's functionality to implement higher-level and application-specific NI operations, as well as to provide the NIProxy with application awareness. For instance, it is the responsibility of the client software to ensure that application-specific information is available to the NIProxy by choosing an appropriate general structure for the client's stream hierarchy (e.g. to reflect that audio has a higher significance than video). However, all functionality required to actually construct the client's stream hierarchy is readily available in the NILayer. Similarly, the client software needs to inform its NIProxy of the relative importance of individual network streams. This information can be retrieved by querying the application's awareness manager, after which it needs to be translated to appropriate weight and/or priority values. While this is an application-specific issue, the task of subsequently communicating the computed values to the client's NIProxy is application-independent and is thus taken care of by the NILayer.

The bandwidth distribution algorithm exploits not only the NIProxy's application awareness, but also its network awareness. Remember that the NIProxy periodically probes the network link of a connected client to estimate its currently available downstream bandwidth. Exactly this amount of bandwidth is subsequently assigned to the root node of the client's stream hierarchy, this way preventing the bandwidth distribution algorithm from allocating more bandwidth than is actually available. As a result, given that the network probing yields sufficiently accurate bandwidth estimates, the capacity of a client's network connection will at all times be respected. On the other hand, the NIProxy's network awareness also comprises knowledge of the bandwidth requirements of network streams and this information of course also influences the operation of the bandwidth distribution algorithm. For instance, specifying that a certain stream has a high significance by assigning its corresponding hierarchy node a high weight value does not guarantee that the stream will actually be "turned on" by the bandwidth distribution algorithm. If siblings exist that have an equally high weight value but require less bandwidth, these siblings will have a higher chance of being forwarded to the proxy client.

5. An example service: Video transcoding

The second QoE-improving mechanism supported by the NIProxy is multimedia service provision. As stated in section 3, this mechanism is implemented using a plug-in approach. More specifically, services correspond to NIProxy plug-ins that can be loaded (and unloaded) dynamically. In this section, we discuss the implementation of an example plug-in which provides a video transcoding service. In addition, we will illustrate how the plug-in exploits its interface to the Bandwidth and Stream Manager to not only consult but also supplement the NIProxy's network and application awareness. Practical results generated by the video transcoding service are presented in section 6.

The video transcoding service enables the NIProxy to reduce the bandwidth requirements of video streams by transcoding them to a lower quality. Since the service is merely a proof of concept, we were concerned more with ease of implementation than with transcoding performance. Consequently, we decided to employ the cascaded pixel-domain approach to transcoding [11]. In this approach, a video stream is transcoded by decoding and subsequently completely re-encoding it with different quality settings. To perform the actual decoding and encoding, we exploited *libavcodec*, the multi-platform audio and video codec library of the open source FFMPEG project [2].

Instead of physically arriving at the NIProxy, transcoded video streams are automatically generated by the video transcoding service. Consequently, the NIProxy has by default no knowledge of the existence of these streams. Therefore, for each distinct video stream it receives, the video transcoding service adds a new node to the stream hierarchy of its associated proxy client[3]. This new node is linked to the transcoded video stream and is added as a sibling of the node that corresponds to the original video stream. Furthermore, since the Stream Manager precedes the Service Manager in the packet processing chain (see figure 1), the NIProxy has no knowledge of the bandwidth usage of transcoded video streams. Consequently, this information is also supplied by the video transcoding service. To adjust the proxy client's stream hierarchy and to complement the NIProxy's network awareness, the video transcoding service exploits its interface to the proxy client's Bandwidth and Stream Manager, respectively.

[3]Remember from section 3 that each service instance is associated with a particular proxy client.

When the video transcoding service is loaded, it informs the Service Manager of its interest in video streams. Consequently, the service will receive all video frames (i.e. network packets belonging to a video stream) that are destined for its associated proxy client. To determine whether the video frames it receives need to be transcoded, the service consults the proxy client's Bandwidth Manager instance. In particular, transcoding only takes place if the Bandwidth Manager computes an accept verdict for the transcoded version of the video stream to which the received packet belongs. If this is the case, the original video frame is replaced by its transcoded counterpart, and the description accompanying the packet is updated so that it now indicates that the frame belongs to the transcoded version of the video stream. Consequently, subsequent services in the proxy client's service list that are also interested in video will receive the transcoded video frame instead of the original. In addition, the second occurrence of the Bandwidth Manager in the proxy client's packet processing chain (see figure 1) will also receive the transcoded video frame, together with its modified accompanying description, and accept it. Notice that in case the description had not been updated by the video transcoding service, the Bandwidth Manager would probably have decided to drop the packet, since it is unlikely that the bandwidth distribution algorithm would allocate bandwidth to both the original and the transcoded version of the same stream.

6. Evaluation

6.1. Test application

To evaluate and test the NIProxy, we used a simple in-house developed Networked Virtual Environment (NVE) application. This application, which is leveraged regularly by our research department to test new ideas and principles, provides users with a top-down 2D view of the shared virtual world and allows them to communicate with each other through both voice and video streaming.

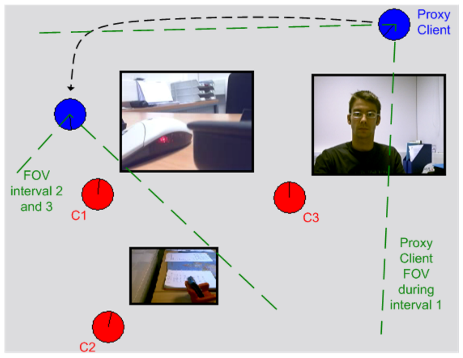
Before we could start the test process, we had to modify the test application so that it was capable of connecting to the NIProxy and, in a later stage, of providing the NIProxy with application awareness. However, this task was simplified tremendously thanks to the availability of the NILayer.

The largest obstacle turned out to be the test application's awareness manager. In particular, due to its straightforward implementation, the awareness manager proved to be incapable of generating sufficiently detailed information regarding the relative importance of network streams. This in turn often resulted in a non-optimal operation of the NIProxy. We mitigated this problem by implementing an additional scheme on top of the awareness manager which determines stream importance in a more fine-grained manner. In particular, the scheme assigns importance to voice streams inversely proportionally to the distance in the virtual world between the local user and the stream source. For video streams on the other hand, the scheme employs not only virtual distance but also virtual orientation information. More specifically, as a video source moves farther away from the Field of View (FOV) of the local user, an increasing penalty value is subtracted from the importance of its video stream.

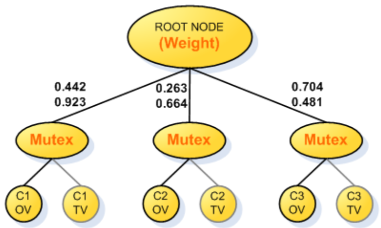
6.2. Experimental results

Using the described test application, we performed several experiments to examine whether the NIProxy is capable of improving the QoE of users of networked applications. However, due to space limitations, we discuss only one representative experiment here. This particular experiment involved four clients, of which only one was connected to a NIProxy instance. The three other clients - which we will call clients C1, C2 and C3 from now on - ran the unmodified version of the test application and hence did not leverage any of the NIProxy's functionality. Consequently, the goal of the experiment was to register the multimedia experience provided to the sole proxy client. Furthermore, to prevent the results from being too complex and hence difficult to grasp, we decided to consider only one type of multimedia traffic in the experiment. In particular, we opted for video since it places the largest load on the transportation network. Opting for video in addition allows us to demonstrate the video transcoding service described in section 5.

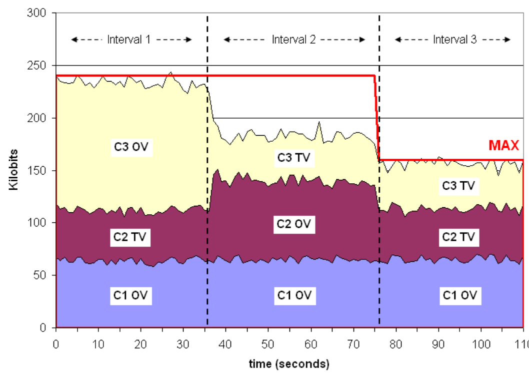
The experiment itself can be thought of as consisting of three distinct intervals. Within each interval, all conditions remained constant, while every transition from one interval to the next was triggered by a change in one or more conditions. The client positioning in the virtual world during the first interval is illustrated in figure 3(a). As can be seen in this figure, the proxy client was oriented towards clients C1, C2 and C3, which consequently all lay in its FOV. After approximately 35 seconds, the proxy client moved along the dashed path depicted in figure 3(a) to arrive at a new virtual location, at which point the second interval of the experiment commenced. Again approximately 40 seconds later, we triggered a transition to the third interval by artificially decreasing the downstream bandwidth available to the proxy client. In particular, during the third interval the proxy client had a downstream bandwidth of only 160 Kilo-



(a) Client positioning in the virtual world.



(b) Stream hierarchy maintained during the experiment.



(c) Stacked graph showing all multimedia network traffic received by the proxy client during the experiment.

Figure 3. Experimental results.

bits per second (Kbps) at its disposal, against 240 Kbps during the first two intervals.

The stream hierarchy which the NIProxy maintained for the proxy client during the experiment is depicted in figure 3(b). This hierarchy was constructed entirely by the proxy client itself, except for the edges in grey and the leaf nodes with a grey outline, which were added by the video transcoding service. In particular, each leaf node created by the proxy client corresponded to the original version (OV) of a specific video stream, while their siblings corresponded to the transcoded version (TV) of this stream. These transcoded versions were not transmitted by the test application but instead were generated automatically by the NIProxy's video transcoding service.

Notice that all weight values are depicted in pairs in figure 3(b). The uppermost value of each pair applied during the first interval of the experiment, while the value below it applied during the second and third interval[4]. To determine the weight values, the proxy client exploited both virtual distance and virtual orientation information, as explained in section 6.1. Since in the first interval all clients lay in the proxy client's FOV, stream importance depended solely on the virtual distance between the proxy client and the stream sources. Consequently, the video stream sent out by C3 was assigned the highest weight, while C2's video stream was assigned the lowest. However, at the end of the first interval, the proxy client moved to a new virtual location, which resulted in shifts in stream importance. In particular, during the second and third interval, C1's video stream was assigned the highest weight since C1 was now located closest to the proxy client and in addition lay in the proxy client's FOV. Furthermore, although the virtual distance between the proxy client and clients C2 and C3 was almost identical, C2's video stream was assigned a higher weight value since C3 was located outside of the proxy client's FOV.

Based on the just described stream hierarchy, the NIProxy distributed the proxy client's downstream bandwidth as illustrated in figure 3(c). In particular, this network trace indicates which video streams the proxy client received during the experiment, the bandwidth consumed by each and the total amount of downstream bandwidth available to the proxy client. For instance, during the first interval of the experiment, the proxy client had a downstream bandwidth of 240 Kbps at its disposal, and this bandwidth was used to receive the original version of the video streams sent out by C1 and C3 and the transcoded version of C2's video stream. Notice the correspondence with the proxy client's stream hierarchy: during the first interval, the nodes associated with C1's and C3's video stream were assigned a higher weight value than the node associated with C2's video stream.

The network trace shown in figure 3(c) comprehensively demonstrates the capabilities and benefits of the NIProxy. First, it illustrates that the NIProxy at all times respects the downstream capacity of a client's network connection (i.e. no more data is forwarded to a proxy client than its network link can handle). Consequently, packet loss and packet latency are kept to a minimum, resulting in improved playback of received multimedia streams at client-side. Secondly, the bandwidth actually available to a proxy client is partitioned in such a way that multimedia streams which the client deems important are assigned more bandwidth than less important streams. Consequently, the proxy client will always receive the multimedia streams that are most important to it in the highest quality possible, conceivably at the expense of less important streams. Finally, besides its bandwidth management functionality, the NIProxy's service provision mechanism is also capable of improving the user QoE. The video transcoding service, for instance, enables the NIProxy to forward lower-quality and consequently less bandwidth consuming versions of video streams to a proxy client. If this service were not available, the NIProxy would be forced more frequently to drop video streams all together in the event of shortage of client downstream bandwidth. Due to the video transcoding service, the NIProxy is in such situations capable of still forwarding some of these video streams to the proxy client, albeit in a lower quality.

For reasons of completeness, we conclude this section with some remarks on the experiment and the presented results. First, notice from figure 3(c) that not all original video streams consumed equal amounts of network bandwidth. For instance, the original version of C3's video stream consumed considerably more bandwidth than the video streams sent out by C1 and C2. This can be attributed to the fact that the webcams of clients C1 and C2 captured a static scene which did not change at all during the experiment, while C3's webcam captured the face of a user. Also note that the bitrate of the original video stream did not influence the bitrate of the transcoded version of this stream (i.e. all streams produced by the video transcoding service have a comparable bitrate). Secondly, the reason why a considerable amount of the proxy client's downstream bandwidth remained unused during the second interval of the experiment is that it did not suffice to boost C3's video stream to a higher quality and that it could not be exploited in any other way. Finally, the scheme adhered to to determine stream importance would be much more meaningful in case the test application provided users with a first-person instead of a top-down view of the virtual environment. We nonetheless decided to employ this scheme because it allowed us to clearly demonstrate the capabilities of the NIProxy.

7. Related work

The two QoE-improving mechanisms supported by the NIProxy are both topics of active research. Automatic bandwidth management and network link sharing have been investigated mainly in the context of Quality of Service (QoS) provision; see for instance [3] and [5]. However, while QoS techniques are typically concerned with guaranteeing that the requirements of data flows are satisfied, the NIProxy's bandwidth distribution algorithm has the more high-level goal of maximizing the multimedia experience provided to users of networked applications. Related work on multimedia service provision on the other hand includes, for instance, the work done by Klara Nahrstedt (e.g. [8]), the MobiGATE system [14], the Active Service framework described in [1] and the on-demand dynamic distillation approach presented in [4]. However, in contrast to the

[4]Since no client relocations occurred at the transition from the second to the third interval, the proxy client computed identical weight values during these two intervals.

NIProxy, none of these systems take application-related information into account. A completely different approach to multimedia service provision is Active Networking [10][6]. An important economic disadvantage of this approach however is that it requires that substantial modifications are made to the architecture of current networks. In contrast, the NIProxy is readily deployable since is compatible with current network architectures and, in particular, the Internet. To summarize, the NIProxy distinguishes itself from related work in both the areas of bandwidth management and multimedia service provision. In addition, an important contribution of the NIProxy is that it combines techniques from these two research topics into a single system.

8. Conclusions and future work

To provide users with a better experience, networked applications are increasingly exploiting multimedia content and real-time multimedia streaming. However, if done improperly, transmitting multimedia content over a transportation network can result in a deterioration of the user's QoE instead of an improvement. In this paper, we have presented the NIProxy, a flexible proxy which introduces more intelligence in the network in an attempt to enable the network to deliver multimedia content in a more intelligent manner. In particular, we have described how the NIProxy acquires both network- and application-related information and we have demonstrated how the NIProxy leverages its dual awareness to intelligently manage the downstream bandwidth available to clients as well as to apply services on multimedia streams on behalf of clients. In addition, we have demonstrated that these two mechanisms are not isolated entities but instead can collaborate with each other. The outcome is an improved multimedia experience for users of networked applications, as is confirmed by the presented experimental results. In particular, the NIProxy prevents overencumbrance of a client's network connection by intelligently deciding which multimedia streams should be forwarded to the client and at which quality.

We intend to improve the NIProxy in several ways in the future. First, the NIProxy lacks knowledge of the client terminal and the end-user's preferences; also incorporating this kind of awareness in the network would enable a myriad of new possibilities. Secondly, the process of loading and unloading services is at the moment directed entirely by the proxy client. We are going to investigate whether some of this control could be migrated to the NIProxy. In particular, we envision the NIProxy automatically (un)loading services based on the client's current requirements. Thirdly, the present version of the bandwidth distribution algorithm is optimized for continuous, long-lived network streams. We would like to rework its implementation so that it can also successfully cope with more bursty network traffic (e.g. the network traffic generated by file transfer). Finally, we would like to make the video transcoding service more dynamic by enabling it to transcode video streams to a range of output qualities and associated bitrates.

Acknowledgments

This research is part of the IBBT E2E QoE project. Part of this research is also funded by the EFRD.

References

[1] E. Amir, S. McCanne, and R. Katz. An Active Service Framework and its Application to Real-time Multimedia Transcoding. In *Proceedings of ACM SIGCOMM'98*, pages 178–189, Vancouver, Canada, September 1998.

[2] The FFMPEG Homepage. `http://ffmpeg.mplayerhq.hu/`.

[3] S. Floyd and V. Jacobson. Link-sharing and Resource Management Models for Packet Networks. *IEEE/ACM Transactions on Networking*, 3(4):365–386, August 1995.

[4] A. Fox, S. D. Gribble, E. A. Brewer, and E. Amir. Adapting to Network and Client Variability via On-Demand Dynamic Distillation. In *Proceedings of ASPLOS-VII*, pages 160–170, Cambridge, Massachusetts, October 1996.

[5] V. Hnatyshin and A. S. Sethi. Architecture for Dynamic and Fair Distribution of Bandwidth. *International Journal of Network Management*, 16(5):317–336, Sept/Oct 2006.

[6] M. Lyijynen, T. Koskinen, S. Lehtonen, and J. Pesola. Content Adaptation on LANE Active Network Platform. In *Proceedings of ConTEL'03*, pages 11–14, Zagreb, Croatia, June 2003.

[7] P. Monsieurs, M. Wijnants, and W. Lamotte. Client-controlled QoS Management in Networked Virtual Environments. In *Proceedings of ICN'05*, pages 268–276, Reunion Island, April 2005.

[8] K. Nahrstedt, B. Yu, J. Liang, and Y. Cui. Hourglass Multimedia Content and Service Composition Framework for Smart Room Environments. *Elsevier Journal on Pervasive and Mobile Computing*, 1(1):43–75, March 2005.

[9] The Netfilter/IPTables Project Homepage. `http://www.netfilter.org/`.

[10] D. L. Tennenhouse and D. J. Wetherall. Towards an Active Network Architecture. In *Proceedings of DANCE'02*, pages 2–15, San Francisco, California, May 2002.

[11] A. Vetro, C. Christopoulos, and H. Sun. Video Transcoding Architectures and Techniques: An Overview. *IEEE Signal Processing Magazine*, 20(2):18–29, March 2003.

[12] M. Wijnants, B. Cornelissen, W. Lamotte, and B. D. Vleeschauwer. An Overlay Network Providing Application-Aware Multimedia Services. In *Proceedings of AAA-IDEA'06*, Pisa, Italy, October 2006.

[13] M. Wijnants and W. Lamotte. Audio and Video Communication in Multiplayer Games through Generic Networking Middleware. In *Proceedings of CGAMES'05*, pages 52–58, Angoulême, France, November 2005.

[14] Y. Zheng, A. T. S. Chan, and G. Ngai. Applying Coordination for Service Adaptation in Mobile Computing. *IEEE Internet Computing*, 10(5):61–67, September-October 2006.