# Managing client bandwidth in the presence of both real-time and non real-time network traffic



WIJNANTS, Maarten & LAMOTTE, Wim (2008) Managing client bandwidth in the presence of both real-time and non real-time network traffic. In: 3RD INTERNATIONAL CONFERENCE ON COMMUNICATION SYSTEM SOFTWARE AND MIDDLEWARE AND WORKSHOPS, VOLS 1 AND 2. p. 442-450..

DOI: 10.1109/COMSWA.2008.4554454
Handle: http://hdl.handle.net/1942/8513

# Managing Client Bandwidth in the Presence of Both Real-Time and non Real-Time Network Traffic

Maarten Wijnants Wim Lamotte
Hasselt University and Interdisciplinary institute for BroadBand Technology (IBBT)
Expertise Centre for Digital Media and transnationale Universiteit Limburg
Wetenschapspark 2, BE-3590 Diepenbeek, Belgium
{maarten.wijnants,wim.lamotte}@uhasselt.be

***Abstract*—Managing client downstream bandwidth is an issue that is rapidly gaining in importance due to the increasing extent to which multimedia content is being exploited in networked applications. Depending on its characteristics, this multimedia content is exchanged in either a real-time or non real-time manner. In this paper, we present the NIProxy, a network intermediary which introduces different types of intelligence in the transportation network in an attempt to improve the Quality of Experience (QoE) provided to users of networked applications. In particular, we concentrate on the NIProxy's bandwidth distribution functionality and we report on how support for non real-time network traffic was incorporated through the adoption of buffering as well as rate control techniques. Using representative experimental results, we demonstrate the NIProxy's capability to successfully manage client downstream bandwidth in the presence of both real-time and non real-time network traffic. In addition, the presented experimental results are compared to the default scenario in which the NIProxy is not involved, revealing a considerable improvement in the user's QoE in case the NIProxy's bandwidth management functionality is leveraged.**

## I. Introduction

In recent years, the amount of bandwidth consumed by networked applications has risen substantially. This is mainly due to an increasing incorporation of multimedia content into these types of applications. As an example, many recent networked applications have abandoned textual chat and instead now exploit a more immersive form of user communication like, for instance, real-time voice streaming or even video chat. The main reason for doing so is that it will normally provide the user with a better experience. However, considerable bandwidth is required to exchange such multimedia data over a transportation network. In fact, even despite the emergence of broadband client network connections (such as xDSL and broadband cable), clients do not always dispose of sufficient downstream bandwidth to receive all network traffic generated by the networked application(s) they are using.

To mitigate this problem, techniques and mechanisms are needed which can manage the client's available downstream bandwidth by distributing it over the different network flows in which the client is interested. In addition, this bandwidth distribution should occur in an intelligent and effective manner, so that the user's experience is maximized. Furthermore, while it is certainly possible to implement a unique bandwidth management solution for each distinct networked application, it is economically more favorable to develop a solution which can be reused by multiple applications. One might term such a solution *communication middleware*.

One possible way to categorize network traffic is to distinguish between *real-time* and *non real-time* network flows. With real-time network traffic, we refer to network flows transporting 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". Real-time network flows are typically continuous, long-lived streams that are transmitted using a relatively simple transport-layer protocol such as the User Datagram Protocol (UDP) or the Real-time Transport Protocol (RTP). In contrast, non real-time network traffic does not suffer from strict constraints on its delivery time, although in many situations it is still preferable to receive the content it transports 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) for transmitting 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 subject of this paper is the *NIProxy*, a network intermediary which is capable of managing client downstream bandwidth and hence can be considered as an example of communication middleware. The NIProxy and its bandwidth distribution algorithm were previously introduced in [1]. However, the previously proposed version of the NIProxy could only successfully cope with real-time network traffic. In this paper, we report on how the NIProxy's bandwidth management mechanism was extended with support for non real-time network flows. In addition, another contribution of this paper is that we thoroughly investigated the impact of this extension on the usability and effectiveness of the NIProxy.

The remainder of this paper is organized as follows. We begin by presenting an overview of the NIProxy system, discussing its aims and its general mode of operation, in section II. In section III, we discuss the NIProxy's client bandwidth distribution functionality and we describe how we extended it so that it could also successfully cope with non real-time network traffic. The implementational issues entailed by this extension are reported on next in section IV. Section V is devoted to the evaluation of the NIProxy's bandwidth management mechanism and presents some representative experimental results. Finally, we briefly review related work in

section VI and we draw our conclusions and suggest possible future research directions in section VII.

## II. The NIProxy: System Overview

The NIProxy is a network intermediary (a "proxy server") to which clients need to connect if they want to exploit its features. The NIProxy's main goal is to enhance the experience and satisfaction of users of networked applications by improving data and content delivery to clients. To achieve this objective, the NIProxy introduces additional *awareness* or *context* in the transportation network. Hence the term NIProxy, which is an abbreviation for *Network Intelligence Proxy*. Another term which we would like to introduce at this point is *Quality of Experience* (QoE), which we will use in the remainder of this paper to more formally denote the experience provided to the user.

The NIProxy currently introduces network as well as application awareness in the network. Under network awareness we understand that the NIProxy has knowledge of the current state of the transportation network, and in particular clients' network connections. The NIProxy acquires this kind of awareness by periodically probing the network link that connects clients to their NIProxy instance, which yields network-related measurements such as the current end-to-end throughput, latency and packet loss rate of each client's network connection. With application awareness on the other hand we refer to the fact that the NIProxy has knowledge of the networked application(s) it is serving. To obtain this kind of awareness, the NIProxy relies on the client software. In particular, the NIProxy expects the client software to forward application-related information to the NIProxy instance to which the client is connected. The kind of information actually forwarded will likely depend on the requirements of the networked application and on its type (e.g. a multiplayer computer game versus an instant messenger). One example of knowledge which could constitute the NIProxy's application awareness is information regarding the relative importance of the different network streams that are being exchanged as part of a networked application. After all, from a client's point of view, it is very likely that not all (types of) network streams are equally significant.

Based on its dual awareness, as stated before, the NIProxy attempts to optimize the QoE provided to users of networked applications. More specifically, the NIProxy currently provides two QoE-increasing mechanisms. First of all, the NIProxy supports automatic and dynamic client bandwidth management, meaning it is capable of intelligently partitioning a client's downstream bandwidth among the different network flows in which the client is interested. Secondly, the NIProxy can also be considered as a multimedia service provision platform, since it is capable of applying services on network flows containing multimedia content on behalf of its connected clients. An important feature of the NIProxy is that its two QoE-increasing mechanisms are complementary as well as interoperable. The mechanisms are complementary in the sense that they are both capable of improving the user's QoE, but each in a different manner. Furthermore, by not treating the mechanisms as isolated entities but instead allowing them to interact and collaborate with each other, the NIProxy is able to improve the user QoE to a higher extent than could be achieved by applying the two mechanisms separately.

The focus in this paper is on the NIProxy's first QoE-improving mechanism, automatic client bandwidth management. For detailed information regarding the NIProxy's underlying ideas and principles, its dual awareness, its multimedia service provision functionality and a description of its software architecture, we would like to refer the interested reader to [1].

## III. Client Bandwidth Management

One of the two QoE-improving mechanisms currently supported by the NIProxy is client bandwidth management. In this section, we will describe how the NIProxy approaches the problem of managing client downstream bandwidth and we will discuss how respectively real-time and non real-time network flows are dealt with. As will be explained, the NIProxy's bandwidth distribution mechanism treats these two types of network traffic differently, which should not come as a surprise since they have completely differing characteristics.

### A. Arranging network flows in a stream hierarchy

The NIProxy manages client downstream bandwidth by organizing all network flows in which the client is interested in a *stream hierarchy*. This stream hierarchy can on the one hand express relationships between network flows, but on the other hand can just as well be used to differentiate between them (or between collections of flows, e.g. audio versus video). More specifically, the stream hierarchy has a tree-like structure that is composed of both internal and leaf nodes. The internal hierarchy nodes implement a certain bandwidth distribution strategy, whereas the leaf nodes always correspond to an actual network flow (e.g. a particular video stream). A number of different types of internal nodes are available to construct the stream hierarchy; these are described next.

The most simple type of internal node is the *mutex* node. As its name implies, this node behaves like a mutex, with in this case the shared resource being the bandwidth that has been reserved for the 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 is assigned bandwidth. The decision of which child node should be allocated the bandwidth available to the mutex node is a function of both the available bandwidth itself and the bandwidth requirements of the children. More specifically, the bandwidth is assigned to the child which has the highest bandwidth requirements that are still smaller than or equal to the bandwidth available to the mutex. If no such child node exists, no child will be assigned any bandwidth.

A second type of internal node is the *priority* node, which distributes bandwidth over its children according to their current priority value. The approach taken here is rather static, in that the bandwidth available to the priority node is first assigned to the child with the highest priority value;

any bandwidth that remains is subsequently assigned to the child with the second highest priority, and so on, until either all bandwidth has been allocated or all children have been considered.

The third and most dynamic type of internal node, the *weight* node, operates in two consecutive phases. In the first phase, bandwidth is partitioned among child nodes based on their maximal bandwidth consumption and current weight value. In particular, each child $c_i$ receives $BW_i = w_i * MaxBW_i * f$ bandwidth. In this formula, $w_i \in [0, 1]$ corresponds to the current weight value of the child, $MaxBW_i$ corresponds to amount of bandwidth which the child can maximally consume, and $f$ is a scaling factor which enforces that $\sum_i BW_i <= BW$, with $BW$ denoting the total amount of bandwidth available to the weight node. In particular, the scaling factor $f$ is calculated as follows:

$$f = \frac{BW}{\sum_i (w_i * MaxBW_i)}$$

After executing phase 1, it is possible that some of the bandwidth available to the weight node remains unused. In particular, this situation will arise when one or more child nodes consume less bandwidth than the amount $BW_i$ that has been reserved for them. If this is the case, the weight node will attempt to distribute the excess bandwidth using a second phase, in which it is assigned to the child nodes on a one-by-one basis, in order of decreasing weight value. In other words, the second phase allows child nodes to exploit any unused bandwidth to switch to a higher bandwidth consumption, hereby favoring children with a higher weight value.

The *percentage* is the last type of internal node. With this type of internal node, each child has a percentage value $p_i \in [0, 1]$ associated with it and bandwidth is partitioned so that each child receives its corresponding percentage $p_i * BW$ of the total bandwidth $BW$ that is available to the percentage node. In case any bandwidth remains unused (which will occur if one or more children consume less bandwidth than the amount they have been assigned), the percentage node mimics the operation of a weight node in that it executes a second phase in which the excess bandwidth is allocated to the different child nodes successively, in order of decreasing percentage value. Note that, in order to achieve a correct and optimal bandwidth distribution, the percentage values of the different children of a percentage node should sum up to 1.

### B. Real-time network traffic

As stated in the introduction, real-time network traffic has stringent reception delay constraints and hence needs to be received by the destination in time. Consequently, the operations that can be performed on it by communication middleware like the NIProxy are limited. For instance, techniques like mid-stream buffering or reduced-rate transmission are typically not applicable, because they could result in the real-time content becoming obsolete due to late arrival at the destination.

Because of the arguments presented in the previous paragraph, care was taken when designing the NIProxy's bandwidth distribution mechanism to ensure that managing real-time network traffic introduces very little to no additional delay. In particular, to manage real-time network traffic, the NIProxy uses *discrete* stream hierarchy leaf nodes. A discrete leaf node does not perform any operations on its associated network flow; instead, it is capable of setting the flow's bandwidth consumption to a discrete number of values. The simplest type of discrete leaf node supports only two bandwidth consumption levels, i.e. 0 and the associated stream's maximum bandwidth usage, and is hence limited to turning the network stream on or off. More complex discrete leaf nodes supporting a number of additional bandwidth consumption levels can however also be devised. For instance, scalable or multi-layer encoded real-time network flows consist of a base layer, which provides the transported content at a certain basic 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 control over the transmission rate, a source will typically transmit all layers of the scalable network stream, leaving it up to the destination to select and receive only the layers it currently requires or is capable of processing. Consequently, a suitable discrete leaf node for this type of network traffic would support a discrete number of increasing bandwidth levels, each one corresponding to permitting the forwarding of an additional layer to the destination.

### C. Non real-time network traffic

In contrast to real-time network traffic, non real-time network flows normally do not impose hard constraints on their delivery delay. Despite it is often still desirable for this type of network traffic to be received by the destination with as little delay as possible, the NIProxy consequently has a bit more latitude in managing it.

The NIProxy's bandwidth distribution mechanism handles non real-time network flows by associating them with a *continuous* leaf node in the client's stream hierarchy. Contrary to a discrete leaf node, a continuous leaf node is capable of setting a network stream's transmission rate to a continuous range of values (i.e. any value lying in the interval [0, maximal stream bandwidth usage]). It does so by locally buffering the content that is transported by the non real-time network flow and by subsequently forwarding the buffered data to the client at a rate that conforms to the exact amount of bandwidth currently reserved for this network flow.

### D. Constructing the stream hierarchy

Constructing the stream hierarchy and ensuring it remains up-to-date falls under the responsibility of the client itself. In particular, the client is responsible for first of all determining a suitable general structure for its stream hierarchy and secondly for ensuring all necessary network flows are adequately incorporated in it. Selecting a suitable hierarchy layout can be considered as a way of providing the NIProxy with application awareness: it indicates how the different network streams in which the client is interested relate to each other, in terms of

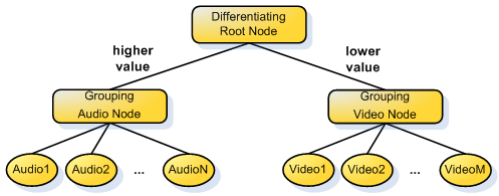



Fig. 1. Example client stream hierarchy preferring audio to video traffic.

significance for the client. For instance, to inform the NIProxy that it attaches greater importance to audio as compared to video, the client could, for instance, construct a stream hierarchy resembling the one depicted in figure 1. In this example, we indicate the client's preference for audio by using some 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 counterpart. The choice of which type of internal node to use in this example as root of the hierarchy and as grouping audio and video node (i.e. priority, weight or percentage) depends on the user's preferences and possibly some other application-related information, and hence the responsibility for making this decision also lies with the client.

The second responsibility of the client, i.e. ensuring concrete network flows are added to its stream hierarchy, is facilitated by the availability of a *stream blocking* mechanism on the NIProxy. In particular, as is discussed in detail in [1], the client can instruct the NIProxy to block certain types of network streams (e.g. audio, video, P2P data, etcetera). Now whenever the NIProxy intercepts a network flow which (i) conceptually belongs to a network traffic type of which the client has indicated it should be blocked and (ii) which the NIProxy's bandwidth distribution mechanism does not know yet, a short description of the network flow is generated and subsequently this description is forwarded to the client instead of the flow itself. Upon arrival of such a description, the client needs to decide how the blocked network flow should be treated by the NIProxy in the future. In particular, the client can either specify that future network packets belonging to this flow should always be forwarded or dropped by the NIProxy, or it can decide to correctly incorporate the blocked network flow in its stream hierarchy. By adhering to the latter alternative, the NIProxy's bandwidth distribution mechanism will start taking the new network flow into account when calculating the distribution of the client's available downstream bandwidth, which makes it the recommended approach since it guarantees the most correct and optimal bandwidth management results.

Based on the discussion so far, it should be apparent that adaptations are required to the client software before a client of a particular networked application can exploit the benefits of the NIProxy. However, the amount of modification effort required is reduced considerably thanks to the availability of an accompanying support library called the *Network Intelligence Layer* (NILayer). In particular, the NILayer exports a number of low-level functions and operations which can be employed by application developers to, for instance, connect the client software to a NIProxy instance and to provide the NIProxy with application awareness. As an example, the NILayer provides functionality through which clients can easily create and maintain their stream hierarchy. The NILayer has been designed to be application-independent and is consequently highly reusable, this way ensuring that the functionality of the NIProxy can be exploited by a wide range of networked applications. More information regarding the NILayer can again be found in our previous work [1].

### E. Managing client bandwidth using the stream hierarchy

Once the client's stream hierarchy has been constructed and assuming it is kept up-to-date, managing the client's available downstream bandwidth simply involves assigning the correct amount of bandwidth to the root node of the stream hierarchy. The stream hierarchy's internal nodes, starting with the root node, will subsequently commence apportioning this bandwidth according to the particular bandwidth distribution technique each implements. Eventually, portions of the client's available downstream bandwidth will reach one or more leaf nodes in the client's stream hierarchy. In case we are dealing with a discrete leaf node, the leaf node will set the bandwidth usage of its associated network flow to the highest discrete bandwidth consumption level that is smaller than or equal to the amount of bandwidth it has been assigned. If we are dealing with a continuous leaf node on the other hand, its associated network flow will be forwarded to the client at a rate which exactly matches the amount of bandwidth that has been reserved for this node. In order to be able to cope with dynamic events like, for instance, network flows turning dead or shifts in stream importance, the NIProxy periodically repeats the entire bandwidth distribution process for each client that is currently connected to it, this way at all times guaranteeing a correct client bandwidth distribution. Practical examples of how the NIProxy manages a client's available downstream bandwidth based on its stream hierarchy, together with the produced results, will be presented in section V.

## IV. Implementation

To add support for non real-time network traffic to the NIProxy's bandwidth management mechanism, we first of all had to implement the continuous stream hierarchy leaf node discussed in section III-C. In contrast to its discrete counterpart, this type of leaf node requires some form of buffering and rate control functionality, since it needs to be capable of setting the bandwidth consumption of its associated network flow to a continuous range of values. Therefore, we have implemented the continuous leaf node so that it (i) buffers the data which the NIProxy intercepts on its associated non real-time network flow, (ii) allows this buffered data to trickle through to the client at a rate equalling the amount of bandwidth that has been assigned to the node by the NIProxy's bandwidth distribution mechanism and (iii) purges processed data (i.e. data that has been forwarded to the client) from its buffer. In effect, the continuous leaf node has been implemented so its operation and behavior resemble those of a *leaky bucket*.

Another issue that had to be addressed was the determination of the maximal bandwidth consumption of non real-time network traffic[1]. For real-time network flows, this issue is resolved by simply registering the rate at which the flow arrives at the NIProxy. A similar approach could however not be employed for non real-time network traffic, since non real-time content is possibly buffered by the NIProxy and forwarded to the client at a rate that differs from the rate at which it was intercepted. We therefore opted 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 transmitted to the client. In other words, the maximal bandwidth usage of continuous leaf nodes at all times equals the current amount of data stored in their associated buffer.

A final implementation decision we had to make concerned the granularity at which we were going to support non real-time network traffic. We opted to manage non real-time network traffic at a rather coarse level, namely at the level of non real-time network flows. This means that all non real-time data that is transferred over the same flow will be treated identically by the NIProxy. Alternatively, we could have decided to enable the incorporation of each individual non real-time content object in the client's stream hierarchy (i.e. even if some of these objects are transported on the same non real-time network stream). However, we believe such fine-grained, low-level support will be useful and meaningful only for a handful of networked applications, while it on the contrary will be considered burdensome by the vast majority. In particular, the fine-grained alternative has the important disadvantage that it will rapidly result in the management of the stream hierarchy becoming complicated and tedious for the client. This is due to the characteristics of non real-time network traffic, which is typically composed of a number of relatively small content objects that are being requested and transmitted in a bursty fashion. As a result, the client would need to update its stream hierarchy very frequently. In addition, calculating the client's downstream bandwidth distribution would start consuming more time due to the increased size of its stream hierarchy. Finally, it is worth noting that the subtle level of control provided by the fine-grained alternative can just as well be achieved by the approach we adhered to, albeit at the cost of some additional overhead: in case a networked application requires control over the bandwidth consumption of individual non real-time content objects, it could transfer each object over a separate non real-time network flow.

## V. Evaluation

This section harbors some representative experimental results which comprehensively demonstrate that support for non real-time network traffic was included successfully in the NIProxy's bandwidth distribution mechanism. In particular, we will describe the bandwidth distribution results produced during two distinct experiments. The first experiment only involved non real-time network traffic, while in the second experiment the NIProxy had to manage client downstream bandwidth in the presence of both real-time and non real-time network traffic. In both experiments, the results produced by the NIProxy's bandwidth management mechanism are compared to the default scenario in which the functionality of the NIProxy is not exploited and it is investigated whether the introduction of the NIProxy led to an improvement of the user's QoE. We begin this section however by briefly describing the application which we employed to generate the presented experimental results.

### A. Test setup

To evaluate the NIProxy's bandwidth distribution functionality, we used a simple test application which allows the user to set up both real-time and non real-time network streams with remote hosts. As example of real-time network traffic, we used video since it is the most demanding type of real-time multimedia content, especially in terms of network bandwidth requirements. Consequently, after a real-time network connection has been set up with some remote host, the latter will immediately start streaming video to the local client (using RTP). In contrast, setting up a non real-time network connection with a remote host does by itself not result in the local client receiving data on this new connection. Instead, the user needs to request some content items (i.e. files) on the non real-time network stream before the remote host will actually start transmitting data over it. In other words, we relied on a simple form of P2P file sharing to generate the non real-time network traffic in the presented experiments. Exchanging the non real-time content is done using TCP, since this protocol offers a number of features which make it very suitable for this kind of communication (e.g. reliable and in-order delivery of transmitted data). Finally, the file sources are implemented so that they transmit files sequentially, hereby following the order in which requests are received.

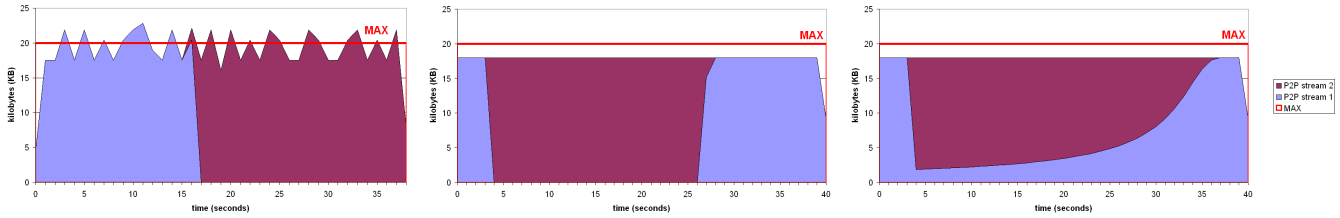
### B. Experiment 1: Managing non real-time network traffic

The goal of the first experiment was to determine whether the NIProxy is capable of successfully managing the downstream bandwidth available to a client in case only non real-time network traffic is being exchanged. To simulate such a scenario, we employed the just described test application to create two separate P2P TCP connections between a client and the same remote host, which we will denote by P2P stream 1 and P2P stream 2 during the remainder of this discussion. After the TCP connections had been set up, the following files were requested:

- `large.png`: 209286 bytes; requested on P2P stream 1
- `small.png`: 102879 bytes; requested on P2P stream 1
- `slide.ppt`: 416768 bytes; requested on P2P stream 2

The files were requested in the order they are enumerated, with 2 second delay intervals applying between each two consecutive requests. Finally, we constrained the client's available downstream bandwidth to 20 kilobytes per second (KBps)

[1]Remember from section III that the maximal bandwidth usage of network flows plays an important role in calculating a client's bandwidth distribution.




(a) Without NIProxy (b) With NIProxy (priority node) (c) With NIProxy (weight node)

Fig. 2. Network traces (stacked graphs) indicating all network traffic received by the client during the three different executions of the first experiment.


(a) (b)

Fig. 3. Exact stream hierarchies used during the second and third execution of the first experiment.

during the experiment. Although this may appear like an unrealistically low value, it allows us to keep the demonstration simple and the generated results compact. In addition, in case there would be contention from real-time network traffic, it might very well be possible that only such a small amount of the client's total downstream bandwidth would be reserved for receiving non real-time content.

The experiment described above was repeated a number of times, each time under different circumstances. First of all, we executed the experiment without involving the NIProxy. Next, the experiment was executed two more times, with the client in both these cases leveraging the NIProxy's bandwidth management functionality. The difference between these two iterations lay in the type of internal node used to differentiate between the two involved P2P streams in the client's stream hierarchy: in the first case a priority node was used, while in the second case we employed a weight node.

During each of the three executions of the experiment, we recorded the network traffic received by the client. The results are shown as stacked graphs in figure 2. The exact client stream hierarchy used by the NIProxy during the second and third execution of the experiment are depicted in figures 3(a) and 3(b) respectively.

Some remarks regarding the network traces shown in figure 2 are in place. First of all, figure 2(a) indicates that, even in case the NIProxy was not involved in the experiment, the client's available downstream bandwidth was more or less respected. This can be attributed to TCP's built-in congestion control mechanism[2]. Secondly, as is illustrated in figures 2(b) and 2(c), the rate control functionality of the continuous stream hierarchy leaf node appears to work very effectively, since the bandwidth consumption of the resulting network traffic is perfectly flat (notice the heavy contrast with the irregularities exhibited in figure 2(a)). Third, figures 2(b) and 2(c) also illustrate that the NIProxy's bandwidth distribution mechanism leaves a small percentage of the client's available downstream bandwidth unused. This unallocated bandwidth is used as safety buffer to guarantee a certain amount of resilience to sudden surges in the bandwidth consumption of the network flows which the client is currently receiving. However, this feature is solely useful in the presence of real-time network traffic, since non real-time network traffic is rate-controlled by the NIProxy before it is forwarded to the client, meaning it will never be subject to such surges. This immediately also explains why it took slightly longer to receive the requested files in the two executions of the experiment involving the NIProxy. It is worth noting however that, although currently not implemented, parametrization of the size of the safety buffer could easily be supported, even on a per client basis. As a result, it would become possible to disable the NIProxy's safety buffer (i.e. set its size to zero) in situations in which only non real-time network traffic is being exchanged, this way effectively eliminating the disadvantage of increased reception times for non real-time content. A final remark relates to the curve separating the bandwidth consumption of the two P2P streams in figure 2(c), whose shape might seem somewhat surprising at first sight. In particular, as can be seen in this figure, the amount of bandwidth assigned to P2P stream 1 gradually increased over time, even though P2P stream 2 had a higher weight value associated with it during the entire experiment. This evolution is explained by the fact that the maximal bandwidth usage of network streams plays an important role in the way weight nodes calculate bandwidth distributions (see section III-A). Since file `slide.ppt` initially (i.e. immediately after it was requested) was allocated much more bandwidth compared to the files requested on P2P stream 1, the maximal bandwidth consumption of P2P stream 2 initially also decreased much faster. As the experiment progressed, the growing difference in maximal bandwidth consumption of both P2P streams increasingly outweighed their associated weight values, which explains why gradually more bandwidth was allocated to P2P stream 1.

Based on the discussion thus far, it might seem like the inclusion of the NIProxy in the experiment had a rather negative impact instead of a positive one (since it resulted in an increase in the time required to receive the requested files). However, an important advantage of the NIProxy which has not yet been mentioned until now is its ability to easily introduce differentiation between the different network flows in which the client is interested. As an example, suppose the two non real-time network connections requested in this experiment corresponded to respectively a low- and high-

[2]TCP dynamically determines and adjusts its transmission rate to prevent network congestion (in an attempt to guarantee high network performance).

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 requested on P2P stream 1. This kind of behavior can easily be achieved using the NIProxy's bandwidth distribution mechanism. It suffices to create a stream hierarchy in which the two P2P streams are adequately differentiated from each other using some sort of internal node. Moreover, since the NIProxy supports multiple types of bandwidth distribution techniques (i.e. internal stream hierarchy nodes), the differentiation can even be tuned to the specific requirements of the client or the application.

The bandwidth distribution produced by the NIProxy during the second and third iteration of the experiment was based on the example requirement described in the previous paragraph, respectively using a priority and a weight node to differentiate between the two non real-time network streams. By now the added value of incorporating the NIProxy in the experiment should become apparent: comparing figures 2(b) and 2(c) with figure 2(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 equalled 11 and 2 seconds). Consequently, leveraging the NIProxy's bandwidth distribution mechanism resulted in the application behaving as expected by the user. Although we did not perform any formal user tests, we think it is intuitively apparent that this in turn yielded an improvement of the user's QoE.

Before concluding the discussion of the first experiment, it is worth noting that the results produced by the NIProxy's bandwidth distribution mechanism could presumably also be achieved by modifying the client software of the employed test application. In particular, in the current implementation of the client software, the only mechanism available to attach priority to content items is to request them in an appropriate order. By refining this implementation, it could become possible to produce results comparable to the ones delivered by the NIProxy. However, leveraging the functionality provided by the NIProxy allows the implementation of the client software to be kept simple, which will likely result in a significant reduction in application development time. Additionally, modifying the client software so that it supports more advanced bandwidth management is not a reusable solution since it embeds the functionality in one particular application. In contrast, the NIProxy's bandwidth management mechanism can be exploited by a broad range of networked applications, even concurrently, making it a much more favorable solution from an economic point of view.

### *C. Experiment 2: Managing both real-time and non real-time network traffic*

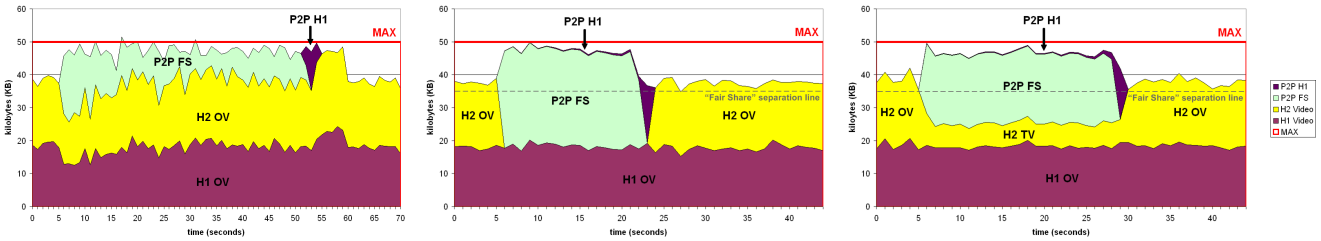
While the first experiment focused solely on non real-time network traffic, the objective of the second experiment was to validate whether the NIProxy is also capable of managing client downstream bandwidth in the presence of both real-time and non real-time network flows. To create a scenario involving both these types of network traffic, we employed the test application to set up (i) a real-time video connection between a client and two remote hosts H1 and H2 and (ii) a non real-time communication channel between the client and remote host H1 as well as another remote host representing a dedicated file server (FS). The non real-time connections were used to request the following files:

- `geom.3ds`: 484652 bytes; requested from file server
- `img.png`: 23419 bytes; requested from remote host H1

The files were requested by the client in immediate succession, in order of enumeration. Furthermore, to allow us to fully demonstrate the capabilities of the NIProxy's bandwidth distribution mechanism, we defined some additional constraints and requirements. In particular, we determined that the video stream emitted by remote host H1 had a higher significance for the requesting client 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. One possible reasoning behind this latter constraint might be, for instance, that the dedicated file server contains files that are crucial for the execution of the application, meaning they should be received as soon as possible, while the other non real-time connection simply allows the local client and host H1 to exchange files in a P2P manner (and hence is far less important). Finally, we also imposed the requirement that non real-time communication should receive a "fair" amount of the total downstream bandwidth available to the client. In particular, we would like at least 30 percent of the client's downstream bandwidth to be designated to the reception of non real-time network traffic.

As was the case in section V-B, the just described experiment was repeated three times. We again began by executing the experiment without including the NIProxy, whereas during the second and third execution of the experiment the client did exploit the NIProxy's bandwidth management functionality. These latter two iterations differed from each other in that during the last execution the video transcoding service of the NIProxy was enabled. As its name indicates, this service allows the NIProxy to reduce the bandwidth requirements of video streams by on-the-fly transcoding them to a lower quality. An in depth description of the video transcoding service can be found in our previous work [1], where it was introduced to illustrate the NIProxy's second QoE-improving mechanism, multimedia service provision.

Network traces illustrating the network traffic received by the client during the different executions of the experiment are shown in figure 4, while figure 5 depicts the stream hierarchy based on which the NIProxy distributed the client's downstream bandwidth during the second and third execution. Important to notice in this latter figure is the influence of the constraints identified at the beginning of this section on the layout of the stream hierarchy. In other words, we attempted to construct the stream hierarchy in such a manner that the specified constraints and requirements were satisfied. Further-




(a) Without NIProxy (b) With NIProxy (video transcoding disabled) (c) With NIProxy (video transcoding enabled)

Fig. 4. Network traces (stacked graphs) indicating all network traffic received by the client during the three different executions of the second experiment.

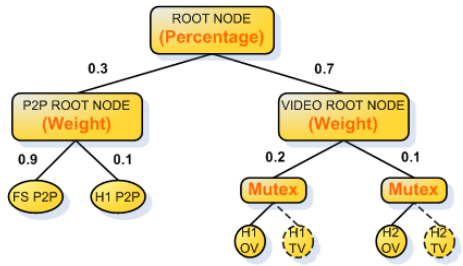



Fig. 5. Stream hierarchy used during the second and third execution of the second experiment.

more, notice that there are actually two leaf nodes associated with each video source in the displayed stream hierarchy. 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 video source), while the rightmost node corresponded with the transcoded, lower-quality version (TV) of this stream (i.e. the version generated by the NIProxy's video transcoding service). The rightmost video leaf nodes are displayed using a dashed outline, since they were only present in the client's stream hierarchy during the third execution of the experiment (when the video transcoding service was enabled).

Figure 4(a) reveals that, during the first execution of the experiment, the simultaneous reception of both real-time and non real-time network traffic 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 kept on transmitting at a constant rate, irrespective of the downstream bandwidth actually available to the client, the non real-time TCP connections automatically adapted their transmission rate to prevent over-encumbrance of the client's network connection. Consequently, the non real-time network traffic needed to content itself with the amount of downstream bandwidth left unused by the real-time traffic, which is in disaccord with our requirement specifying that the non real-time communication should receive a fair share of the total downstream bandwidth available to the client. Secondly, although the real-time video flows claimed the majority of the client's available downstream bandwidth, they still suffered from the contention from the non real-time traffic. In particular, small amounts of packet loss were introduced and the reception of the real-time video traffic became more irregular, which in turn resulted in a deteriorated video playback at client-side.

Looking at figure 4(b), we see that the issues described in the previous paragraph did not occur in case the client exploited the bandwidth distribution functionality of the NIProxy. More specifically, by relying on a percentage node to differentiate between real-time and non real-time network traffic, we were able to guarantee the non real-time network traffic a certain amount of the client's downstream bandwidth capacity during the entire experiment (i.e. at least 30 percent)[3]. The outcome was a much faster reception of the requested files, however at the expense of the least important video stream being turned off 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 forwarding it to the client, 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 4(c), enabling the video transcoding service of the NIProxy had the additional advantage of allowing the real-time video traffic to consume its assigned 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 bandwidth distribution algorithm was now 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 fact that the transcoded version has lower bandwidth requirements than its original counterpart, and hence forwarding it did not result in the non real-time traffic being denied its fair share of the available downstream bandwidth.

To summarize, the advantage of incorporating the NIProxy in the experiment discussed in this section was twofold. First of all, it resulted in the client's downstream bandwidth being distributed correctly over the different real-time and non real-time network flows in which the client was interested. Secondly, it enabled us to satisfy the requirements specified at the beginning of the section with minimal effort (i.e. without requiring substantial modifications to the client software). Based on these observations, we believe it is fair to say that the introduction of the NIProxy in this experiment again had a positive influence on the user's QoE, even though we

[3]Notice that, when no non real-time data needed to be received, the video traffic consumed more than its allocated bandwidth percentage. This is due to the percentage node distributing any bandwidth left unused by its children in a second phase, as explained in section III-A; in this case, the excess bandwidth was assigned to the video root node.

cannot confirm this conclusion with results from formal user tests. Finally, it also worth noting that the experimental results presented in this section once more illustrate the added value of supporting interaction between the NIProxy's bandwidth distribution mechanism and its service provision functionality. As we already stated in [1], enabling collaboration between its two QoE-increasing mechanisms is one of the most important features of the NIProxy, since it allows the NIProxy to improve the user QoE to an extent that could not be achieved by applying the two mechanisms separately.

## VI. Related Work

The topic of automatic bandwidth distribution, and network resource management in general, has already received considerable attention from the research community. For instance, the Differentiated Services (DiffServ) networking architecture supports end-to-end allocation of networking resources, possibly even across separate domains, through a so-called Bandwidth Broker (BB) entity. Interesting work in this BB context includes, for instance, the GARA architecture [2], the ARM approach [3], the transcoding protocol described in [4] and the QoS management framework proposed in [5]. Examples of network resource management in network architectures other than DiffServ are for instance presented in [6], [7], [8] and [9]. The NIProxy distinguishes itself from these approaches in that the latter are concerned with Quality of Service (QoS) provision (i.e. guaranteeing that the requirements of data flows are satisfied), whereas the NIProxy's bandwidth distribution mechanism pursues the more high-level goal of maximizing the multimedia experience provided to users of networked applications. To accomplish this objective, the NIProxy extensively exploits its application awareness, a type of context that is often left unconsidered in related systems. Moreover, another unique feature of the NIProxy is that it integrates bandwidth management and multimedia service provision in a single system in such a manner that interoperation between both mechanisms becomes possible. Finally, while the NIProxy focuses solely on managing downstream bandwidth, we would like to point out that it is just as well possible to manage the uplink capacity of network links; an example hereof is given in [10].

## VII. Conclusions and Future Work

Networked applications are increasingly exploiting multimedia content and, as a result, the issue of client downstream bandwidth management is gaining importance at an equally steady rate. Depending on its characteristics, multimedia content is transmitted over the transportation network using either real-time or non real-time network flows. In this paper, we have reported on our continued development of the NIProxy, a network intermediary which aims to improve the QoE provided to users of networked applications by incorporating additional awareness in the network. More specifically, we have focused on automatic client bandwidth management, one of the two QoE-improving mechanisms currently supported by the NIProxy, and we have described how it was extended with support for non real-time network traffic (the previously proposed version of the NIProxy could only successfully cope with real-time network streams). Through the presentation of representative experimental results, we have demonstrated that this addition resulted in an increased applicability and effectiveness of the NIProxy, since it is now capable of successfully managing client downstream bandwidth in the presence of real-time as well as non real-time network traffic. In addition, the presented experimental results also indicate that incorporating the NIProxy in networked applications yields a considerably positive influence on the QoE of their users, meaning its set forth objective is achieved.

As part of future work, we intend to investigate the impact of employing the NIProxy to manage client downstream bandwidth in a more realistic networked application generating both real-time and non real-time network traffic. In particular, we plan to incorporate the NIProxy in an in-house developed Networked Virtual Environment (NVE) application which supports both real-time voice and video chat and which relies on non real-time network communication to exchange different types of geometry information (IBR data, 3D models, ...) that are required to render the virtual world at client-side.

## Acknowledgments

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

## References

[1] M. Wijnants and W. Lamotte, "The NIProxy: a Flexible Proxy Server Supporting Client Bandwidth Management and Multimedia Service Provision," in *Proceedings of the IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM 2007)*, Helsinki, Finland, June 2007.

[2] I. Foster, M. Fidler, A. Roy, V. Sander, and L. Winkler, "End-to-End Quality of Service for High-End Applications," *Computer Communications*, vol. 27, no. 14, pp. 1375–1388, 2004.

[3] A. Ramanathan and M. Parashar, "Active Resource Management for The Differentiated Services Environment," in *Proceedings of the 3rd Annual International Workshop on Active Middleware Services (AMS 2001)*, San Francisco, USA, August 2001, pp. 78–86.

[4] R. Kumar, J. Rao, A. Turuk, S. Chattopadhyay, and G. K. Rao, "A Protocol to Support QoS for Multimedia Traffic over Internet with Transcoding," in *Proceedings of the HiPC Trusted Internet Workshop (TIW 2002)*, Bangalore, India, December 2002.

[5] E. Kusmierek, B.-Y. Choi, Z. Duan, and Z.-L. Zhang, "An Integrated Network Resource and QoS Management Framework," in *Proceedings of the IEEE Workshop on IP Operations and Management (IPOM 2002)*, Dallas, USA, October 2002, pp. 68–72.

[6] S. Floyd and V. Jacobson, "Link-sharing and Resource Management Models for Packet Networks," *IEEE/ACM Transactions on Networking*, vol. 3, no. 4, pp. 365–386, August 1995.

[7] V. Hnatyshin and A. S. Sethi, "Architecture for Dynamic and Fair Distribution of Bandwidth," *International Journal of Network Management*, vol. 16, no. 5, pp. 317–336, September/October 2006.

[8] F. M. Anjum and L. Tassiulas, "Fair Bandwidth Sharing among Adaptive and Non-Adaptive Flows in the Internet," in *Proceedings of the IEEE Conference on Computer Communications (INFOCOM 1999)*, New York, USA, March 1999, pp. 1412–1420.

[9] M. Furini and D. Towsley, "Real-Time Traffic Transmission over the Internet," *IEEE Transactions on Multimedia*, vol. 3, no. 1, pp. 33–40, March 2001.

[10] S. Chandra, C. S. Ellis, and A. Vahdat, "Differentiated Multimedia Web Services using Quality Aware Transcoding," in *Proceedings of the IEEE Conference on Computer Communications (INFOCOM 2000)*, Tel Aviv, Israel, March 2000.