EFFICIENT TRANSMISSION OF RENDERING-RELATED DATA USING THE NIPROXY

Maarten Wijnants Tom Jehaes Peter Quax 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
e-mail: {maarten.wijnants,tom.jehaes,peter.quax,wim.lamotte}@uhasselt.be

ABSTRACT

Despite the emergence of broadband Internet connections, client downstream bandwidth remains a scarce resource and should hence be managed judiciously. Based on this observation, we have previously introduced the NIProxy, a network intermediary supporting automatic client bandwidth management. However, the results presented in our previous work were generated using "artificial" test applications whose bandwidth management requirements could be satisfied using relatively straightforward strategies. In this paper, we evaluate the NIProxy's suitableness to manage client downstream bandwidth in a realistic, more complex networked application. The considered application supports real-time audiovisual user communication and in addition employs an advanced rendering scheme which imposes a number of specific requirements regarding the distribution of rendering-related data to clients. Through the presentation of representative experimental results, we demonstrate that the NIProxy is not only capable of fulfilling these requirements but also effectively improves the Quality of Experience (QoE) provided to users of the considered application.

KEY WORDS

Multimedia communication systems, NIProxy, client bandwidth management, QoE optimization, rendering-related data transmission

1 Introduction

Support for custom or user-generated content in networked multi-user applications is rapidly gaining popularity. One example is the currently ongoing evolution of the World Wide Web from a mostly static information source to a more dynamic entity which allows users to generate and distribute content themselves, for instance through blogs and wikis (the so-called Web 2.0). Another example is the emergence of social virtual worlds such as Second Life[1] and There[2], which support creation and even trading of self-made virtual items.

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

The most straightforward way to support client downstream bandwidth management is to integrate it in the software of the networked application itself. However, providing a separate bandwidth management approach for each individual networked application is an unprofitable solution from an economic point of view. A more cost-effective approach is to develop a generic and hence reusable middleware solution that can perform bandwidth management for multiple networked applications. In addition, the user might be running more than one networked application at the same time. If in this case the bandwidth management solution is embedded in the networked applications themselves, correct interoperation between the different approaches might not be guaranteed, resulting in bandwidth distribution outcomes that are suboptimal at best. In contrast, the middleware approach is in such situations capable of maintaining a global picture of all network traffic in which the client is currently interested and hence allows the generation of optimal results.

The subject of this paper is the *Network Intelligence Proxy (NIProxy)*, a network intermediary which provides automatic and dynamic management of client downstream bandwidth. The NIProxy was previously introduced in [1]; a detailed discussion of its bandwidth management functionality is provided in [2]. However, the bandwidth management results presented in our previous work were generated using "artificial" test applications. In contrast, in this paper we report on our findings of employing the NIProxy to manage client bandwidth in a realistic, real-world networked application supporting user-generated content, which in this particular case imposed the need for dynamically distributing rendering-related data to clients. Compared to the previously used test setups, the

[1] http://secondlife.com/
[2] http://www.there.com

networked application considered in this work required a much more sophisticated bandwidth management strategy. The main contribution of this paper is hence an illustration of the NIProxy's ability to effectively manage client bandwidth in real-world networked applications having possibly complex requirements. As a side effect, this also demonstrates the NIProxy's flexibility and general applicability.

The remainder of this paper is organized as follows. In section 2, we briefly describe the NIProxy and list its main objectives and features. The networked application which we used as test case to evaluate the NIProxy's appropriateness to manage client downstream bandwidth in a real-world setting, is described next in section 3. Specific focus in this section will be given to the application's rendering scheme. Section 4 harbors a number of implementational issues and in particular illustrates how the NIProxy was exploited to manage the distribution of rendering-related data to clients in the considered networked application. The impact hereof is investigated next in section 5 through the discussion of representative experimental results. Finally, we review related work in section 6 and present our conclusions in section 7.

2 NIProxy

2.1 Objectives and approach

The high-level objective of the NIProxy is to maximize the Quality of Experience (QoE) provided to users of networked multimedia applications[3]. The NIProxy attempts to achieve this goal by incorporating different types of *awareness* or *context* in the transportation network so that intelligent and efficient delivery of content to clients becomes possible. In particular, the NIProxy exploits its awareness to enhance the user QoE in two complementary manners. First of all, it provides automatic and dynamic management of client downstream bandwidth, meaning it is capable of intelligently allocating the downstream bandwidth available to a client. Secondly, the NIProxy supports multimedia service provision, i.e. it is capable of applying services on network flows containing multimedia content on behalf of its clients.

2.2 Awareness introduction in the network

The context currently gathered by the NIProxy is twofold and comprises both network- and application-related information. The NIProxy acquires its *network awareness* through active probing of the client's last mile network connection, which yields measurements like the current throughput, latency and packet loss rate of the client's access link. The NIProxy's *application awareness* on the other hand is composed of information regarding the networked applications its connected clients are currently running. To amass this type of context, the NIProxy mainly relies on the client software. In particular, the NIProxy expects its clients to relay application-related information to it. To facilitate this process, a support library called the Network Intelligence Layer (NILayer) was developed, which exports an API that allows application developers to provide the NIProxy with application-related knowledge with minimal effort. Due to the generic design of the NI-Layer, it is highly reusable and can hence be integrated and exploited in a wide variety of networked applications.

2.3 Client bandwidth management

The focus of this paper is on the NIProxy's first QoE-improving mechanism, client downstream bandwidth management. A detailed discussion of this subject is beyond the scope of this paper, since it was already treated comprehensively in our previous work [2]. Instead, we will limit ourselves to a recapitulation of its main features and mode of operation so that the reader disposes of all information necessary to comprehend the material presented in this paper, without having to refer to our previous work.

The NIProxy's bandwidth management mechanism supports network traffic having either real-time or non real-time characteristics. With real-time network traffic, we refer to network flows transporting content with stringent delivery constraints like, for instance, interactive audio or video. In contrast, non real-time network traffic (e.g. network flows containing file or P2P data) can typically cope with relatively large and varying amounts of delivery delay, but is more susceptible to packet loss and data corruption. In other words, whereas real-time network traffic typically needs to be delivered to the destination in time, non real-time network traffic mainly needs to be delivered reliably and error-free. Due to their widely divergent characteristics, the NIProxy manages both categories of network traffic differently, as will be explained later on.

From a technical point of view, the NIProxy's bandwidth management mechanism operates by organizing all network flows in which a client is interested in a *stream hierarchy*. Such a 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 specific video stream). Different types of internal nodes are available, each with their distinct characteristics and capabilities. Those internal hierarchy nodes used to manage the bandwidth consumed by the networked application considered in this paper, are briefly enumerated below; a detailed discussion of these nodes and the other internal hierarchy nodes supported by the NIProxy can be found in our previous work [2].

- `Priority`: Partitions bandwidth among its children on a one-by-one basis, in order of decreasing priority value; in particular, bandwidth is first allocated to the child with the highest priority, excess bandwidth (if any) is subsequently allotted to the child with the second highest priority, and so on
- `WeightData`: Considers all of its children collectively and distributes bandwidth among them in two

[3] Throughout this paper, we will use the term QoE to formally denote the user's experience and satisfaction; in contrast to Quality of Service (QoS), it is a rather subjective criterion.

consecutive phases according to their current weight value; in the initial phase, each child c_i receives $BW_i = w_i * BW * \frac{1}{\sum_i w_i}$ bandwidth, where $w_i \in [0, 1]$ represents the child's weight value and BW denotes the total amount of bandwidth available to the `WeightData` node; if any bandwidth remains unused after the execution of phase 1 (which will occur in case at least one child consumes less bandwidth than its assigned amount), this excess bandwidth is exploited in the second phase to allow children, starting with the one with the highest weight value, to switch to a higher bandwidth consumption level

- `WeightStream`: Operates identically to the `WeightData` node, except it also takes the maximal bandwidth consumption of its children into account, resulting in the following bandwidth distribution during the first phase: $BW_i = w_i * MaxBW_i * \frac{BW}{\sum_i (w_i * MaxBW_i)}$
- `Percentage`: Each child c_i has a percentage $p_i \in [0, 1]$ associated with it ($\sum_i p_i$ should equal 1) and is apportioned its corresponding percentage $p_i * BW$ of the total bandwidth BW available to the `Percentage` node; if a child consumes less than its assigned amount, the excess bandwidth is allocated to the other children consecutively, in order of decreasing percentage value

As is the case for the internal nodes, there also exist different categories of hierarchy leaf nodes. In particular, leaf nodes for the management of both real-time and non real-time network flows are available. Due to the strict delivery requirements of real-time network traffic, its processing should introduce as little additional delay as possible. Consequently, real-time leaf nodes are designed to toggle the bandwidth consumption of its associated network flow between a discrete number of values (i.e. they can turn the stream off or can forward it to the client at maximal quality, or at any intermediate bandwidth consumption level supported by the stream[4]). Non real-time network traffic on the other hand is less sensitive to delivery delay, meaning its processing is allowed to be more time-consuming. As a result, non real-time hierarchy leaf nodes are capable of forwarding their associated network flow to the client at any rate in the continuous interval [0, maximal stream bandwidth usage]. This is achieved through the adoption of buffering as well as dynamic rate control techniques.

By adequately constructing the stream hierarchy, it is possible to express relationships between network flows or, conversely, to differentiate between them (or between collections of flows, e.g. audio versus video). As soon as the stream hierarchy has been composed, and assuming it is kept up-to-date, managing client downstream bandwidth simply amounts to allocating the correct amount of bandwidth to the hierarchy root node.

[4]Intermediate bandwidth consumption levels typically only occur in case the stream is scalable or multi-layer encoded.

3 Considered networked application

3.1 Overview

To evaluate the NIProxy's suitableness to manage client downstream bandwidth in a less artificial setting, we integrated it in a real-world 3D Networked Virtual Environment (NVE) application. The considered NVE assigns great importance to user communication and hence supports both audio and video chat between users. Another important feature of the NVE is its rendering scheme, which supports both hybrid 3D rendering techniques and sophisticated Level of Detail (LoD) approaches (see section 3.2). Finally, the NVE application enables users to populate the shared virtual world with custom, possibly self-made 3D content. Distribution of this user-provided data occurs through a dedicated file server, which disseminates the data to clients in a unicast manner as soon as it becomes needed to be able to correctly render the virtual world.

3.2 Rendering scheme

The NVE's rendering scheme uses a hybrid geometric - image-based representation (IBR) approach [3]. In short, relief texture mapped objects (RTMOs) are used, which were first presented in [4], as an efficient representation for distant objects. These representations are made up of a collection of images with depth information, which are warped during run-time in order to get an appropriate view based on the current camera position. On the other hand, objects close to the viewer are rendered using the progressive meshes (PM) approach of [5], hereby assuring that geometric rendering does not cause the framerate to drop below a predefined threshold. The main advantage of the adhered hybrid rendering strategy is that it greatly reduces the time needed for rendering complex virtual worlds, without incurring too large a sacrifice in image quality.

We have also proposed some optimization techniques for transmission of data associated with representations in NVE applications using this hybrid representation approach [6]. By first streaming the low-detail image-based representations of objects in the virtual environment, it becomes possible to quickly present the user with a complete, albeit low-quality view of the world. This view is gradually upgraded as more detailed representations (i.e. geometric information) are downloaded based on the current rendering need. We have analyzed the network traffic associated with these representation-related data streams and it was shown that the reduced-detail representations strike an effective balance between visual quality and transmission time. As a result, the proposed representation transmission scheme significantly reduces the time needed for rendering, at an acceptable quality, an initial view of the virtual environment.

Since the considered NVE application lacked an intelligent bandwidth management solution, such as the proxy solution we are presenting here, the models were previously requested one at a time based on their current scene priority. However, as priorities can possibly change rapidly

during scene traversal, and especially during viewpoint rotation, a better solution needed to be found in order to be able to more effectively manage the downloading of model data required for rendering the world at client-side. Moreover, the combination of non real-time and real-time traffic was not handled in this previous work. The remainder of this paper therefore focusses on the use of the NIProxy for the transmission of 3D model data. Additionally, we will present results in which these transmissions are combined with real-time video transmissions.

3.3 Model data

Name	Text.	Base	L1	L2	L3	Base Compr.
Chimp	304	12	11	21		283
Cow	161	22	21	41	83	168

Table 1. Storage sizes (in KiloBytes) for several progressive mesh models in the NVE database.

Resolution	Num RTs	Size	Compressed
32x32	1	4	2
	3	12	6
	6	24	13
64x64	1	16	7
	3	48	21
	6	96	45

Table 2. Storage sizes (in KiloBytes) for RTMO image-based representations.

The NVE's model database stores representations in the form of both progressive meshes and relief texture mapped objects. Focussing first on the progressive mesh data, table 1 presents the progressive mesh storage sizes of two example models in the database. The first column indicates the amount of texture data for each model, stored in the JPEG format. If we compare this value to the data size of the base mesh in the second column, we see that the texture data far outweighs the base mesh data. Using a multi-resolution texturing solution could decrease this gap, but overall, texture data usually takes up a large piece of the total model data. The columns numbered L1 to L3 indicate the data size of each consecutive PM level, whereby at each step the triangle count is doubled. In order to transmit the PM models in an efficient way, they are stored in a compressed format at server-side. After transmission, the models are extracted at client-side, after which they can be used in the render stage. The last column in table 1 indicates the size after compression of the texture data together with the base mesh. Since the texture data is already stored in the JPEG file format, the additional compression does not significantly reduce the total file size.

Looking at table 2, which displays the storage sizes for the RTMO representations, we see a completely different situation. The storage size for an RTMO mainly depends on the resolution used for the relief textures, instead of on the model that is being represented. We use quantized depth values so each depth pixel is stored as an RGBA value, with the alpha value representing the quantized depth value. For rendering an RTMO, at most three relief textures are needed for a specific viewpoint. If we therefore compare the compressed values for three relief textures to the data size needed for the compressed PM base representations, it is obvious that the RTMO transmission size is much smaller. In the NVE database, only relief texture resolutions of 32x32 and 64x64 are used, since these provide sufficient image quality for distant objects and for the initial rendering of nearby objects. We can further see that even if we would send the complete RTMO, we are still better of compared to the PM base representation with regard to transmission time.

4 Implementation

Based on the discussion in the previous section, it is apparent that the NVE application requires the distribution of real-time as well as non real-time network flows to clients. Especially the application's advanced rendering scheme and support for user-generated content causes client bandwidth management to be a far from trivial process and imposes a number of specific requirements and constraints. In this section, we will report on how the NIProxy was exploited to intelligently fulfill this task and present the main implementational issues it encompassed.

4.1 Devising a suitable structure for the client's stream hierarchy

To be able to effectively manage the bandwidth consumption of the considered NVE application, the requirements imposed by its distinct features had to be translated into an appropriate structure for the client stream hierarchy. The structure we arrived at is illustrated in figure 1. As can be seen, the root node of the hierarchy is of type `Percentage` and is used to discriminate between respectively the real-time and non real-time network traffic generated by the NVE application. In more detail, the real-time network traffic receives 30 percent of the client's available downstream bandwidth, while 70 percent is reserved for the reception of non real-time network streams.

As root for the real-time subtree of the hierarchy, we used a node of type `WeightStream`. All real-time (i.e. audio and video) network flows are made direct children of this root node using real-time leaf nodes, with their weight values dynamically depending on virtual distance as well as virtual orientation information[5]. The root of the non real-time branch of the stream hierarchy on the other hand consists of a `Priority` node and has two children, which are both of type `WeightData`. The leftmost `WeightData` node groups together all high-priority (HP) rendering-related files that need to be received by the client, whereas the rightmost is used as root for rendering-related

[5] A scheme similar to the one presented in [1] is used to dynamically determine the weight value of each real-time network flow; please see our previous work for more information.

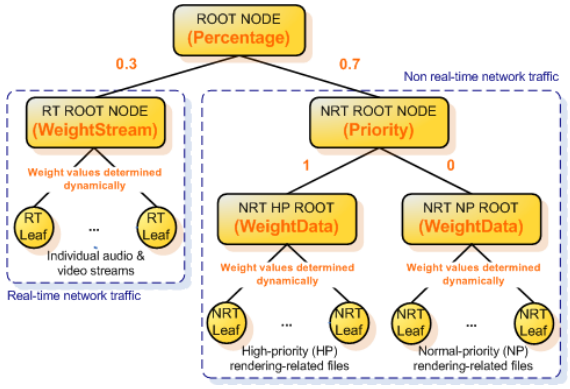

Figure 1. General structure of the stream hierarchy used to manage the downstream bandwidth of clients of the considered NVE application.

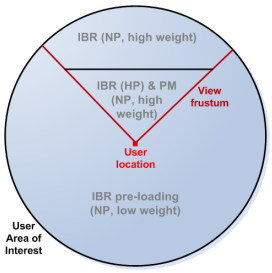

Figure 2. Categorization of rendering-related data.

data having normal-priority (NP). In particular, the HP portion of the non real-time subtree is used exclusively to receive IBR data and, more specifically, only the IBR data of those models that lie in the current viewing frustum and are therefore crucial for rendering a first view of the virtual environment. All other rendering-related files are incorporated under the NP part of the non real-time subtree. The weights of the individual HP as well as NP non real-time hierarchy leaf nodes are based on their scene priority. Scene priority is determined by the NVE's LoD selection mechanism, which takes into consideration distance to the viewer as well as model display size. Furthermore, we perform pre-loading of IBR model data outside the frustum, but inside a circular Area of Interest around the viewpoint. These data streams are given a low weight so they do not hinder the reception of higher priority streams. A schematic overview of how rendering-related data is categorized and incorporated in the stream hierarchy is given in figure 2.

Since the grouping HP root node has a higher priority value associated with it compared to the NP root node, the entire percentage of the client's available downstream bandwidth that has been designated to the reception of non real-time network traffic will first be exploited to receive the HP rendering-related files; only if any bandwidth remains in excess, it will subsequently be granted to the NP branch of the non real-time subtree.

4.2 Support for hierarchy leaf node relocations

Since users can freely roam about the virtual world offered by the considered NVE application, the importance of individual real-time and non real-time network flows is likely to change dynamically over time. For the real-time network traffic, shifts in stream importance are captured by appropriately updating the weight value of its corresponding real-time hierarchy leaf node. For non real-time network traffic however, the situation is somewhat more complicated and possibly requires relocation of non real-time leaf nodes in the client's stream hierarchy. In particular, as a user moves to a new location or changes his viewing direction in the virtual world, non real-time data that previously belonged to the high-priority category might now be classified as being normal-priority, and vice versa. When this occurs, in addition to enforcing any necessary updates to their individual weight values, the involved non real-time leaf nodes are automatically transferred between the HP and NP branches of the non real-time subtree. Finally, besides parent switching, complete removal of leaf nodes from the client's stream hierarchy is also supported. A plausible cause are radical user relocations in the virtual environment, which could reduce the importance of certain network flows to such an extent that they should no longer be allocated any client downstream bandwidth. By (temporarily) pruning the corresponding leaf nodes from the stream hierarchy, this can easily be guaranteed.

5 Evaluation

5.1 Experiment 1

To assess the presented bandwidth distribution strategy, we first performed a minimalist test involving only a limited number of models. This setup allowed us to clearly and comprehensively trace the reception of non real-time rendering-related data at client-side. The results are shown as a stacked graph in figure 3. Besides constraining the model count in this initial experiment, we also opted to limit the client downstream bandwidth to an unrealistically low 20 KiloBytes per second (KBps) to allow for an intelligible analysis of the produced results. In section 5.2, we will present experimental results attained during a more realistic test involving a densely populated scene, a more common client bandwidth limit and contention from real-time network traffic.

Looking at figures 3 and 4, we see that at the beginning of the experiment (view frustum 1), object 0 is further away and only needed the IBR data, while object 1 is close to the viewer and hence needed IBR data as well as all PM levels. We can see that for both objects, the high-priority IBR data was requested first, after which the reception of object 1's PM levels was started. Also, pre-loading of IBR data of objects outside the view frustum began taking up a small portion of the available bandwidth.

After about 30 seconds, the user performed a 90 degree rotation (view frustum 2), which put both object 0 and 1 outside his view frustum. As a result, the weight values associated with both model streams were significantly reduced. On the other hand, the view frustum now contained objects 2 and 3, so their data was requested at a higher

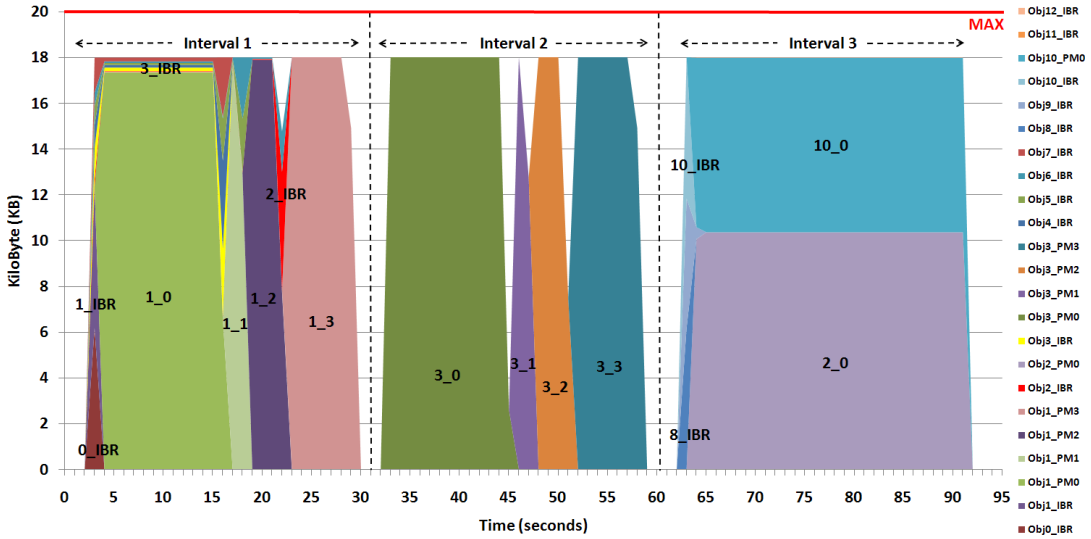

Figure 3. Simple scenario network traffic chart (stacked graph) demonstrating intelligent management of non real-time (i.e. rendering-related) traffic (in KBps).

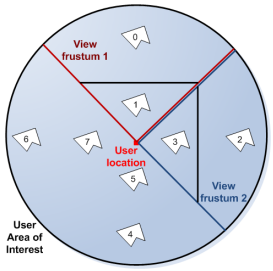

Figure 4. Scene layout for experiment 1.

weight and priority than before. Again, object 3 was close to the viewer and was given more weight, while only IBR data was required for object 2. Note that, after the rotation, only the PM data for object 3 was transmitted since the IBR files for both objects had already been received because of the NVE's IBR pre-loading strategy for objects outside the view frustum. The renderer could therefore immediately present a first view of the objects in the environment.

Finally, at around 60 seconds, a forward translation was performed (not shown in figure 4), resulting in object 3 falling outside the view frustum, objects 2 and 10 being selected for PM rendering and object 8 being added to the view at lowest resolution. The transmission continued as before, except that now objects 2 and 10 needed to split the available bandwidth based on the weights assigned to them by the run-time LoD selection. The RTMO data for object 2 was already received during the first experiment interval, so it first had to wait for the transmission of the IBR models of objects 8 and 10 to complete before it could continue with upgrading its own PM level.

5.2 Experiment 2

Our second experiment involved the transmission of a complex scene and hence demonstrates how the proposed bandwidth management system behaves in lifelike situations. Furthermore, besides rendering-related data, this experiment also included the simultaneous reception of real-time video traffic. During this experiment, the client downstream bandwidth was set to the more realistic value of 100 KBps.

By analyzing the graph in figure 5[6], we notice that at the start of the experiment IBR data was given full priority to be able to quickly provide the user with an initial view of the virtual world using these low-quality model representations. After this, model data was incrementally upgraded based on the resolution levels selected by the NVE's LoD approach. After approximately 35 seconds, most geometric data needed for the higher-fidelity rendering of objects in the frustum was received by the client and, as a result, the bandwidth amount assigned to the IBR pre-loading of models outside the user's view frustum was increased.

At around 55 seconds, two video sources entered the user's area of interest; each video stream consumed a bandwidth of approximately 17 KBps. Recall from section 4.1 that the NIProxy reserves only 30 percent of the total client bandwidth for the transmission of real-time network traffic. This amount did not suffice to simultaneously receive both video streams, but since there was excess bandwidth from the non real-time network traffic category, the real-time network flows were allowed to exceed their allocated bandwidth share. Roughly 15 seconds later however, the user rotated 90 degrees in the virtual world, causing model importance to alter due to the user's changed viewing direction in the virtual world. As a result, new geometric model data needed to be received by the client and hence contention for the available client downstream bandwidth between real-time and non real-time network traffic was introduced. Due to the contention, the real-time network traffic now needed to adhere to its assigned bandwidth per-

[6]Due to the large number of non real-time network flows involved in the second experiment, the graph no longer shows the bandwidth consumption of individual data streams; instead, the aggregate bandwidth consumed by each category of non real-time data is displayed.

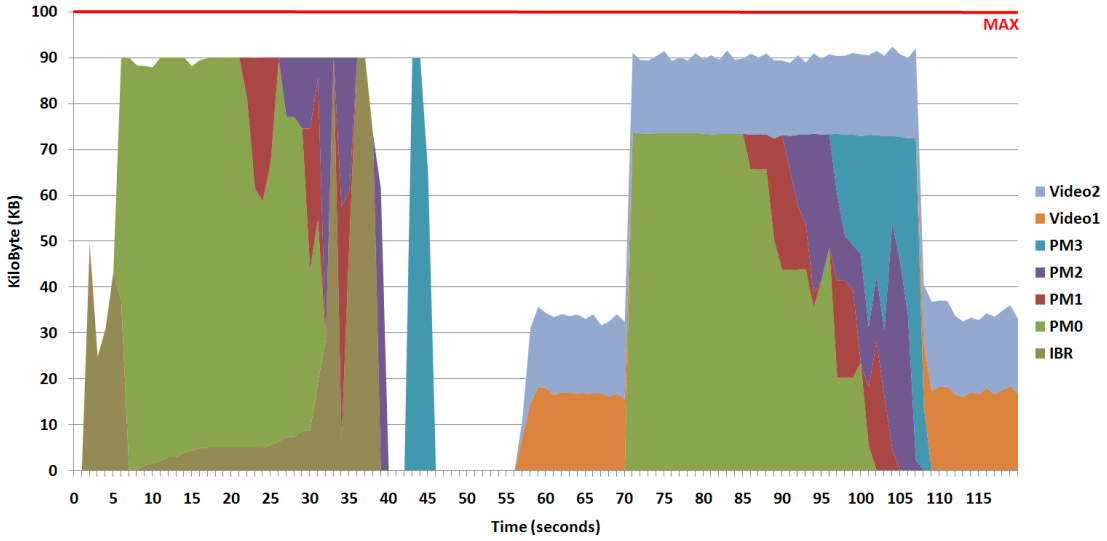

Figure 5. Stacked graph illustrating the client bandwidth distribution produced during a realistic scenario involving the reception of both 3D object data and real-time video traffic (in KBps).

centage, which resulted in the NIProxy temporarily blocking one of the video streams to prevent the non real-time network traffic from being denied its fair bandwidth share. Only after the 3D model transmission was finished, the simultaneous forwarding of both video streams was resumed.

5.3 Discussion

The results produced during the two presented experiments comprehensively demonstrate the benefits and capabilities of the NIProxy's bandwidth management mechanism. A first important observation is that over-encumbrance of the client's network connection was at all times prevented. As a result, packet delay and loss were minimized and an optimal data reception at client-side was achieved. Notice that a small percentage of the client's available downstream bandwidth was even left unused in the experiments. This unallocated amount is used as safety margin to guarantee a certain amount of resilience to sudden surges in the bandwidth consumption of the network flows which the client is currently receiving. Secondly, the requirements of the considered NVE application, and in particular its rendering scheme, were captured successfully by the NIProxy. For instance, as is confirmed by the presented bandwidth distributions, by assigning priority to the transmission of IBR data, the NVE's objective of quickly providing the user with an initial view of the virtual world was fulfilled. Finally, the second experiment also indicates the NIProxy's ability to correctly deal with situations in which real-time and non real-time network traffic contend for the downstream bandwidth available to the client. In particular, by employing a `Percentage` node to differentiate between real-time and non real-time network flows, both traffic categories received their fair bandwidth share during the entire experiment. Based on these findings, we believe it is intuitively apparent that the NIProxy positively impacts the QoE provided to users of the considered NVE application.

6 Related work

Due to the increasing extent to which multimedia content is being exploited in networked applications, management of network resources, and client downstream bandwidth in particular, is a topic of active research. Two of the initial explorers of the issue were Floyd and Jacobson [7]. More recently, Massoulié and Roberts studied the topic from a more mathematical point of view [8]. Work in the network resource management context is however not limited to theoretical research. Interesting examples of concrete systems and frameworks supporting automatic bandwidth management include the GARA architecture [9], the Exact Bandwidth Distribution Scheme (X-BDS) [10], the Bandwidth Allocation Mechanism (BAM) for the Premium Service presented in [11] and the bandwidth sharing algorithm discussed in [12]. The NIProxy differentiates itself from these approaches in that the latter are concerned with QoS provision, whereas the NIProxy's bandwidth distribution mechanism pursues the more high-level goal of optimizing the multimedia experience provided to users of networked applications. Moreover, the NIProxy disposes of and exploits application-related information, a type of context that is lacking in (most) related systems. A final distinctive feature of the NIProxy is that it integrates bandwidth management with multimedia service provision in a single system in such a manner that interoperation between both mechanisms becomes possible (not covered in this paper; see [1][2] for more information).

Efficient network transmission of rendering-related data, the primary focus of this paper, has also already received considerable attention from the research community. Interesting related work in this subdomain of automatic bandwidth management includes the research performed by Ioana Martin [13][14], the 3D models Transport Protocol (3TP) [15] and the generic middleware for the streaming of 3D progressive meshes proposed in [16].

These approaches concentrate solely on optimizing the delivery of 3D data over networks and hence provide advanced techniques targeted specifically at this type of multimedia content. Scenarios requiring the simultaneous reception of 3D data and other types of multimedia, such as real-time video or P2P data, are not covered however. In contrast, the NIProxy employs a more generic bandwidth management scheme and supports client bandwidth management in the presence of different types of real-time as well as non real-time network traffic. While this implies that the mechanisms for the transmission of rendering-related data supported by the NIProxy are less sophisticated compared to the referred systems, it also guarantees a much wider applicability.

7 Conclusions

The rate at which multimedia content is being incorporated in networked applications seems to exceed the rate at which the downstream capacity of client network connections is evolving. As a result, the issue of client downstream bandwidth management is rapidly gaining in importance. In this paper, we have reported on the NIProxy, a network intermediary which introduces different types of awareness in the transportation network in an attempt to improve multimedia content delivery to clients. In particular, we have presented our findings of employing the NIProxy to manage client downstream bandwidth in a real-world NVE application. The considered application requires effective distribution of rendering-related data and in addition supports real-time streaming of audiovisual content. Using representative experimental results, we have demonstrated the NIProxy's ability to translate the requirements imposed by the application into efficient client bandwidth distributions. This in turn yielded an improvement of the QoE provided to the NVE's users, meaning the NIProxy's set forth objective was achieved. The presented results also highlight the amount of flexibility afforded by the NIProxy's generic approach to client bandwidth management, which enables it to be integrated in a multitude of networked applications.

Acknowledgments

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

References

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

[3] Tom Jehaes, Peter Quax, Patrick Monsieurs, and Wim Lamotte. Hybrid Representations to Improve Both Streaming and Rendering of Dynamic Networked Virtual Environments. In *Proceedings of the ACM International Conference on Virtual-Reality Continuum and its Applications in Industry (VRCAI 2004)*, pages 26–32, Singapore, June 2004.

[4] Manuel M. Oliveira, Gary Bishop, and David McAllister. Relief Texture Mapping. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH 2000)*, pages 359–368, New Orleans, USA, July 2000.

[5] Hugues Hoppe. Progressive Meshes. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH 1996)*, pages 99–108, New Orleans, USA, August 1996.

[6] Tom Jehaes, Peter Quax, and Wim Lamotte. Analysis of Scalable Data Streams for Representations in Networked Virtual Environments. In *Proceedings of the ACM Workshop on Network and System Support for Games (NETGAMES 2004)*, Portland, USA, August 2004.

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

[8] Laurent Massoulié and James Roberts. Bandwidth Sharing: Objectives and Algorithms. *IEEE/ACM Transactions on Networking*, 10(3):320–328, June 2002.

[9] Ian Foster, Markus Fidler, Alain Roy, Volker Sander, and Linda Winkler. End-to-End Quality of Service for High-End Applications. *Computer Communications*, 27(14):1375–1388, 2004.

[10] Vasil Hnatyshin and Adarshpal S. Sethi. Architecture for Dynamic and Fair Distribution of Bandwidth. *International Journal of Network Management*, 16(5):317–336, September/October 2006.

[11] Marco Furini and Donald Towsley. Real-Time Traffic Transmission over the Internet. *IEEE Transactions on Multimedia*, 3(1):33–40, March 2001.

[12] Farooq M. Anjum and Leandros Tassiulas. Fair Bandwidth Sharing among Adaptive and Non-Adaptive Flows in the Internet. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM 1999)*, pages 1412–1420, New York, USA, March 1999.

[13] Bengt-Olaf Schneider and Ioana M. Martin. An Adaptive Framework for 3D Graphics over Networks. *Computers & Graphics*, 23(6):867–874, 1999.

[14] Ioana M. Boier-Martin. Adaptive Graphics. *IEEE Computer Graphics and Applications*, 23(1):6–10, January 2003.

[15] Ghassan AlRegib and Yucel Altunbasak. 3TP: An Application-Layer Protocol for Streaming 3-D Models. *IEEE Transactions on Multimedia*, 7(6):1149–1156, December 2005.

[16] Hui Li, Ming Li, and Balakrishnan Prabhakaran. Middleware for Streaming 3D Progressive Meshes over Lossy Networks. *ACM Transactions on Multimedia Computing, Communications and Applications*, 2(4):282–317, November 2006.