Audio and Video Communication in Multiplayer Games through Generic Networking Middleware

Maarten Wijnants
Hasselt University
Expertise Centre for Digital Media
and transnationale Universiteit Limburg
Wetenschapspark 2, BE-3590 Diepenbeek, Belgium

Wim Lamotte
Interdisciplinary institute
for BroadBand Technology (IBBT)
Expertise Centre for Digital Media
BE-3590 Diepenbeek, Belgium

{maarten.wijnants, wim.lamotte}@uhasselt.be

***Abstract*— On-line computer gaming is steadily becoming more popular. This is partly due to the increasing realism, visual and other, which recent games present to players. However, one technological aspect that is often still neglected by the current generation of games is the provision of in-game streaming functionality. This can be attributed for a large part to the considerable effort developers need to put into its implementation. Client bandwidth is a valuable resource which should not be squandered lightly, meaning an efficient and thus time-consuming implementation is required. In this paper, we present our networking middleware which facilitates the integration of real-time streaming facilities in multiplayer games. The networking middleware consists of a number of interconnected proxy servers that are both application and network aware. In addition, the proxies are extensible, meaning game developers can add specific functionality to them when necessary. One example of such an extension we implemented ourselves is a video transcoding plug-in that enables the proxy to transcode video streams to a lower quality on-the-fly. Through experimental results, we demonstrate how our middleware exploits its compound awareness and video transcoding functionality to automatically and intelligently route and possibly also adapt multimedia network streams, thereby relieving game developers of this burden.**

***Index Terms*— Networking middleware, multimedia streaming, multiplayer computer games**

I. Introduction

Continuously improving computer hardware allows developers to create ever more realistic games. For instance, the introduction of the Graphics Processing Unit (GPU) has drastically increased the visual realism of computer games. The release of the first Physics Processing Unit (PPU), planned for the end of this year, is expected to do the same to the physical correctness of game worlds [1]. However, when it comes to the communication facilities provided by on-line multiplayer games, there has been very little progress. Many games at the moment still only support textual chat, which many players consider to be both awkward and slow to input. Some games therefore already allow players to communicate through audio streams, but there is still virtually no support for video communication or, more general, the exchange of multimedia streams other than audio.

Exchanging multimedia streams however does have a number of possible applications in networked computer games. As already mentioned above, one interesting use would be to enable players to communicate in a more natural and immersive way. Audio and certainly video communication inherently transfers information about the current emotional state of players, something textual chat can only fake through the use of emoticons. Even if video communication inside the game world might look a bit overdone at the moment, it would definitely be convenient if it were supported in game lobbies or staging areas. In addition, audiovisual communication attaches a distinct voice and face to nicknames, this way allowing gamers to recognize others more easily in the virtual world. Another possible use could be to enable the dynamic streaming of multimedia content in games. For instance, in the near future we envision game servers dynamically transmitting video fragments, possibly advertisements, to players based on their current interests and location in the game world.

The lack of support for exchanging multimedia network streams in the current generation of on-line multiplayer games can be attributed for a large part to the high bandwidth requirements of these streams. This is especially true for video, which can consume up to half a megabit per second, depending on the video quality and the codec being used. Consequently, game developers need to spend a lot of effort to implement it efficiently, since client bandwidth should never be wasted on multimedia streams the client is not interested in. Furthermore, developers sometimes want their multiplayer games to be playable on both low-bandwidth dial-in modems as well as high-bandwidth DSL or cable modem connections, which again requires extra code to be written.

In this paper, we present a generic networking middleware that makes the process of incorporating multimedia streaming facilities in multiplayer games considerably easier. The middleware consists of a number of interconnected proxy servers that are positioned at the edge of the network, close to end-users. Each proxy server is *application* and *network aware*, meaning it has high-level knowledge of both the application (i.e. game) it is serving as well as the state of the underlying transportation network [2]. In addition, the proxy servers are equipped with video transcoding functionality, which enables them to on-the-fly scale down high-quality video streams [3]. Based on their compound awareness and transcoding capabilities, the proxy servers intelligently route and possibly also adapt multimedia network streams destined for connected clients (i.e. players). Consequently, game developers no longer need to spend time on these issues and can instead focus on other aspects of their game. Finally, our networking middleware is also extensible, meaning game developers can easily

add specific functionality to it when needed.

The remainder of this paper is structured as follows. In section II we briefly review related work on the distributed proxy network architecture and on networking middleware. Next, we discuss the implementation of our own networking middleware in section III. In section IV we describe how we integrated our work into an existing networked application, while section V presents some experimental results that arose from this integration and which clearly demonstrate the benefits of our work. Finally, we conclude our paper and suggest possible future research directions in section VI.

II. Related work

The network architecture of most current on-line multiplayer games is still based on the client/server model. However, game developers are slowly discovering the benefits of incorporating proxy servers inside the network, either to assist the central server or even to completely replace it. The Massively Multiplayer On-line Game (MMOG) Eve Online, for instance, utilizes a centralized server cluster to compute the game state, but exploits proxy servers to relieve the server cluster from certain tasks such as data integrity checking [4]. By adhering to this approach, more players can be simultaneously on-line in the game world. The research community on the other hand has already been investigating this topic for a couple of years now. For instance, Cronin et al. present in [5] the results of porting a popular First Person Shooter (FPS) game to the distributed proxy architecture. In particular, they report that by eliminating the central server and replacing it with a number of interconnected proxies which are located close to the clients, they were able to at all times guarantee a low end-to-end latency, which in turn resulted in an increased player satisfaction. A number of additional advantages of the distributed proxy architecture in the context of multiplayer games is given by Mauve et al. in [6]. Finally, Nguyen et al. discuss in [7] how the distributed proxy architecture can be employed to provide immersive audio communication to users of MMOGs. The presented proxy system is however only capable of managing audio streams, whereas our networking middleware can route and adapt all possible kinds of multimedia streams.

The distributed proxy architecture is also receiving a great deal of attention in the context of multimedia content delivery. Here, proxy servers are normally used to overcome the mismatch between what the content server provides and what the requesting client can handle. For instance, a number of proxy systems can be found in the literature which try to improve the delivery of web pages to clients by transcoding the images contained inside HTML pages to a more suitable format, either based on the client's network connection, the capabilities of his device, or both. Examples include [8], [9] and [10]. Similar proxy systems have been devised to improve the real-time streaming of video fragments to heterogeneous clients, e.g. [11] and [12]. Other interesting proxy systems in this context include *AMPS*, a highly scalable proxy research platform designed specifically to support a wide and extensible set of streaming services [13], and the *QTP* architecture which

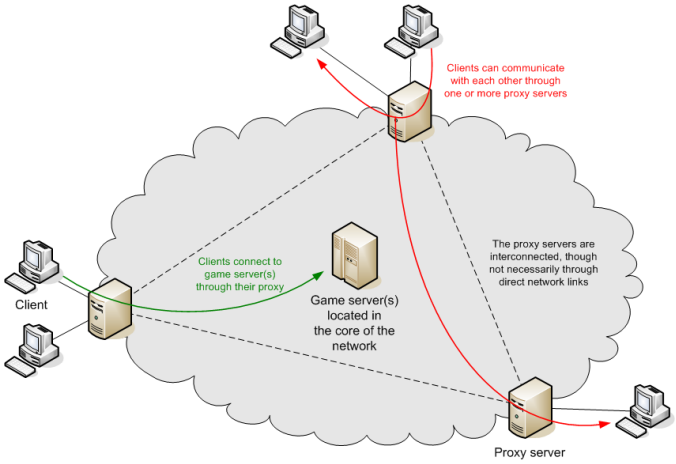


Fig. 1. All network streams originating from or destined for a certain client pass through the proxy server this client is connected to.

tries to satisfy simple Quality of Service requirements by exploiting an intelligent version decision policy and service admission control scheme [14].

There are many resemblances and overlaps between the proxy systems cited in the previous paragraph and our own networking middleware. However, our networking middleware distinguishes itself from these systems by its application awareness. Indeed, all cited systems lack application awareness, which limits their use to simple content delivery scenarios. Our networking middleware on the other hand does take application related information into account when making routing and transcoding decisions. Consequently, our middleware can be employed not only in a content delivery context, but also in more complex networked applications such as multiplayer computer games, which in fact is the topic of this paper.

III. Implementation

As stated in the introduction, our networking middleware consists of a number of interconnected proxy servers that are positioned at the edge of the network. Instead of directly connecting to a central game server or to each other in a peer-to-peer manner, clients need to connect to one of these proxy servers, preferably the one nearest to them in the network topology. By doing so, the client-to-proxy hop count and delay will normally be kept to a minimum. The proxy server subsequently becomes responsible for forwarding relevant network streams to the client and, conversely, for correctly routing packets sent by the client to their destination. This is illustrated in figure 1.

To be able to make intelligent routing decisions, our networking middleware continuously retrieves and stores information regarding the application(s) it is serving as well as the current state of the transportation network. To acquire *application awareness*, our middleware requires the client application to send application related information to the proxy this client is connected to. For instance, the client software could keep the proxy server up-to-date regarding the relative importance, from this client's point of view, of the different multimedia streams that are currently being exchanged inside

the application. The more application related information the proxy receives, the more intelligent decisions it can make. To gain *network awareness* on the other hand, each proxy periodically probes the network links of the clients that are currently connected to it. This probing yields network related information such as the current latency and throughput of individual client network connections. In addition, the proxies record the bandwidth usage of each network stream that passes through them.

Based on its compound awareness, our networking middleware intelligently manages the downstream bandwidth connected clients have at their disposal. For instance, as players move around in the virtual game world, the relative importance of individual multimedia streams is likely to change. Due to its application awareness, the middleware can detect such shifts in stream importance and change the allocation of the client's downstream bandwidth accordingly. On the other hand, its network awareness enables the middleware to intelligently react to fluctuations in a client's downstream capacity. For instance, if a client's downstream bandwidth no longer suffices to receive all multimedia streams at their original quality, the middleware will automatically determine which streams should be reduced in quality or even completely blocked, this way ensuring that the capacity of the client's network link is never exceeded. When making such decisions, the middleware will always take both the relative importance of the involved streams as well as their bandwidth requirements into account. As a result, whenever possible, less important streams will be transcoded or dropped by the middleware before more significant streams are altered. A concrete example of how our middleware could manage the client bandwidth in practice can be found in section V.

When designing our networking middleware, applicability in a wide range of networked applications was an important objective. To achieve this goal, we decided to keep the middleware as generic as possible. However, we soon realized many applications require specific middleware functionality, either to achieve an acceptable level of performance or to support key features of the application. We therefore equipped our proxy servers with a plug-in mechanism which can be used to extend their functionality. Application developers can implement their own plug-in that suits their particular needs, hereby exploiting knowledge of specific characteristics or features of their application, and install it in the proxy servers. As an example, we implemented a plug-in that adds video transcoding functionality to our networking middleware. We would like to refer the interested reader to [3] for a detailed description of this particular plug-in. Notice that although the plug-ins reside on the proxy servers, they are conceptually part of the application the middleware is serving. As a result, the plug-in approach does not interfere with our middleware's generic design.

The main goal of our networking middleware is to simplify and speed up the process of including efficient real-time streaming functionality in networked applications. Consequently, during the middleware's design phase, we paid extensive attention to the fact that it should cost developers minimal effort to integrate our work into their application. First

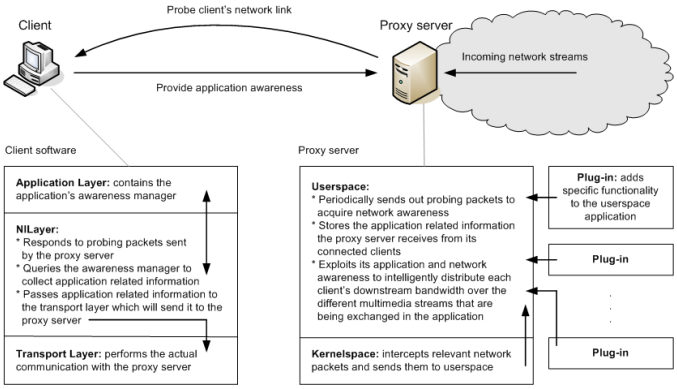


Fig. 2. High-level overview of the operation of our generic networking middleware.

of all, implementing a new plug-in simply involves deriving from a plug-in base class and adding the desired functionality. Secondly, instead of requiring application developers to completely recode their client software, we provide them with a *Network Intelligence Layer* (NILayer) which they can utilize to take care of the communication between the client and our networking middleware. It suffices to insert the NILayer between the transport layer and the application layer of the client software, and to interface it with the application's awareness manager[1]. The NILayer will automatically respond to probing packets sent out by our middleware, and will continuously query the awareness manager to collect application related information, which it will pass on to the proxy server this client is connected to. Besides including the NILayer, no other adaptations to the client software are required.

To implement our networking middleware, we exploited the functionality provided by the netfilter framework [15] which enables packet filtering and mangling on the linux platform. In particular, netfilter inspects all network packets that arrive or pass through a linux machine and allows a userspace application to register interest in particular network streams. All packets belonging to these streams are subsequently transferred from the linux kernel to the application, which can then decide what should be done with them.

The userspace application we developed for our networking middleware continuously combines its application and network awareness to decide whether or not a connected client should receive the packets netfilter transfers to it. If so, the packets are reinjected into the kernel for transmission to the client. If not, they are discarded. By installing plug-ins, the functionality of this basic userspace application can be extended. For instance, the video transcoding plug-in enables the userspace application to on-the-fly scale down high-quality video streams before forwarding them to connected clients [3]. Notice that although our networking middleware inspects and works on individual network packets, it conceptually operates in the application layer. This is illustrated in figure 2, which shows a schematic overview of our middleware's general mode of operation.

[1]An awareness manager is a piece of software responsible for identifying which objects and information in a system are most relevant to a certain user.

TABLE I
QUALITY PARAMETERS OF THE THREE VIDEO STREAMS VIDEO-BASED CLIENTS SEND OUT.

	High Quality (HQ)	Medium Quality (MQ)	Low Quality (LQ)
Codec	H263	H263	H263
Resolution	CIF (352x288)	CIF (352x288)	CIF (352x288)
FPS	25	15	15
Bitrate (bps)	200.000	100.000	50.000

IV. INTEGRATION IN AN EXISTING NVE

We tested our networking middleware by integrating it in our in-house developed Networked Virtual Environment (NVE) which was first introduced in [16]. Probably the most remarkable feature of the NVE is its support for *video-based avatars*, a technique which allows users to be represented in the virtual world by the video their webcam records [17]. Consequently, the NVE requires video streams to be exchanged between users in real-time. In addition, the NVE also allows users to communicate verbally through audio streams. Based on these observations, we decided the NVE was an excellent candidate to test our work in. Indeed, although described here in the context of on-line multiplayer gaming, our networking middleware can just as well be incorporated in other kinds of networked applications due to its generic design. Furthermore, our middleware can attain a high level of performance in all these different kinds of applications thanks to its plug-in mechanism which ensures middleware extensibility.

The actual integration proved to be straightforward. All that was required was inserting the NILayer in the client software and linking it with the NVE's awareness manager. The NVE's awareness manager is region-based, meaning it tries to decrease the amount of information clients need to receive and process by spatially dividing the virtual world into a number of regions [16]. Every region has a distinct multicast address associated with it which is used as communication channel for the events that occur in that region. For instance, clients are required to send their state updates to the multicast address of the region in which they are currently residing. The awareness manager at each client constantly determines which regions this client should be aware of, and only the multicast groups associated with these regions are joined. As a result, clients will never receive information originating from regions in which they are not interested.

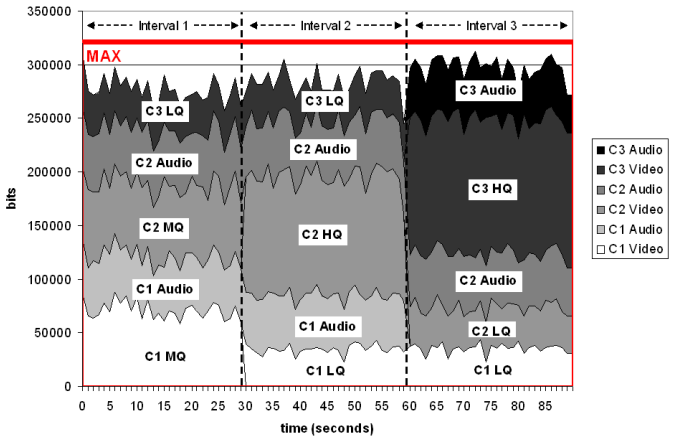
Besides a multicast group for exchanging event information, each region also has three multicast addresses associated with it for distributing video data. Using these multicast groups, the NVE requires video-based clients to send out three distinct qualities of their video stream. This design decision was made with scalability in mind: as can be seen in table I, the High Quality (HQ) video setting requires twice as much bandwidth as the Medium Quality (MQ) setting, which in turn consumes twice the bandwidth of the Low Quality (LQ) setting. Each client's awareness manager is responsible for determining which video quality should be received from every region the local user is currently aware of, after which the corresponding video multicast addresses are joined. A likely strategy would be to subscribe to the HQ video multicast group of the region the user is currently located in, and to the MQ or even LQ multicast group of adjacent regions. Finally, there is also a separate multicast address associated with regions which is used solely to exchange audio streams. Analogous to the distribution of state updates, clients send their audio stream to the audio multicast address of the region in which they are currently residing. Again, the NVE's awareness manager decides which audio multicast groups should be joined. Notice that this means that, in contrast to video, the NVE does not support multiple audio qualities. This decision is based on the fact that audio streams containing voice require considerably less bandwidth than video. As a result, clients either receive an audio stream sent out by another client or they do not receive the stream at all.

The NILayer dynamically extracts information about the currently selected regions (i.e. their associated multicast groups) from the client's awareness manager and transfers this information to the proxy server the client is connected to. In addition, the NILayer continuously monitors the virtual distance between the local user and the other clients that are currently present in any of the regions the client is subscribed to. Based on this positional information, the NILayer informs the proxy server of the relative importance of the different multimedia streams that are currently being exchanged inside the NVE, this way providing our networking middleware with application awareness. After all, a user is most likely to interact with the clients he is closest to in the virtual world. Consequently, if we want these interactions to be both efficient and convincing, the user should receive the multimedia streams sent out by nearby clients at a fidelity that is as high as possible within the user's current downstream bandwidth constraints. This may imply that multimedia streams sent out by more distant clients will be reduced in quality or even completely blocked by our middleware to preserve bandwidth for the more significant streams. Notice that such fine-grained control over the transmission of multimedia streams is by default not supported by the NVE. In particular, the NVE is not capable of at run-time adjusting the quality of individual multimedia streams, but instead is only flexible enough to reduce the stream quality of entire regions.

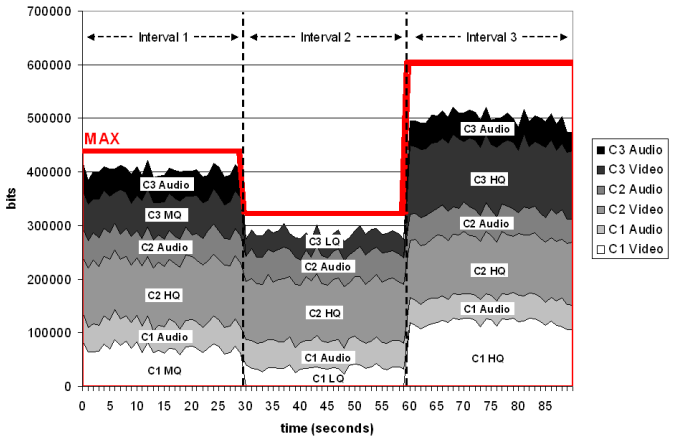
V. EXPERIMENTAL RESULTS

After integrating our networking middleware into the NVE as described in the previous section, we performed several experiments to evaluate our middleware's effectiveness. Two such experiments are described here, one in full detail and the second only briefly. The first experiment involved three video-based clients which did not make use of our middleware (clients C1, C2 and C3), and two clients that did (clients PA and PB). For reasons of simplicity and clarity, these two latter clients did not send out audio nor video streams. Furthermore, all involved clients were located in the same region of the virtual world. The initial client positioning in this region is illustrated in figures 3(c) and 3(d) which respectively show a 2D top-down view and a 3D view of the virtual world.

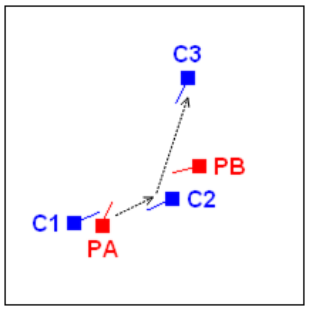
To generate sufficiently elaborate results, we treated clients PA and PB differently during the experiment. First of all, we



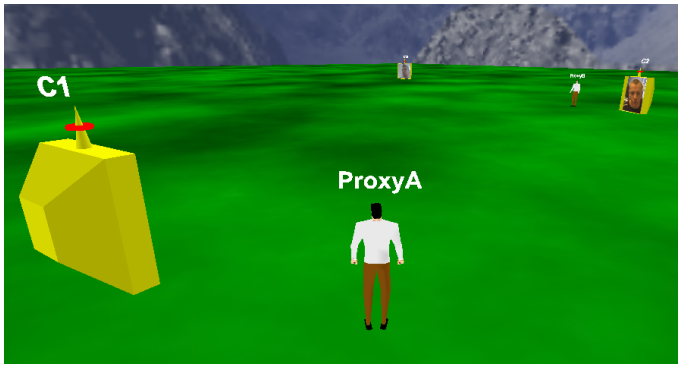
(a) network packets received by PA



(b) network packets received by PB



(c) 2D view of the virtual world



(d) 3D view of the virtual world

Fig. 3. Results of experiment 1: (a) and (b) network traces (stacked graphs) showing all multimedia network packets received by respectively client PA and client PB; (c) 2D view illustrating the client positioning in the virtual world; (d) corresponding 3D view.

kept the downstream bandwidth of client PA fixed at 320 Kbits per second (Kbps) for the entire duration of the experiment, whereas client PB was subject to substantial downstream bandwidth fluctuations. More specifically, at intervals of approximately 30 seconds, we set PB's available downstream bandwidth to respectively 440, 320 and 600 Kbps. Notice that such severe bandwidth fluctuations are not very likely to occur in a time span of 90 seconds in practice, but we nonetheless decided to use these extreme values because they allow us to clearly demonstrate the capabilities of our networking middleware. Secondly, client PB remained stationary in the virtual world during the experiment, whereas client PA moved along the path which is depicted in figure 3(c) as two dotted arrows. Analogous to PB's bandwidth fluctuations, PA's relocations in the virtual world occurred at intervals of approximately 30 seconds. This means PB started off next to client C1, 30 seconds later rapidly moved towards client C2, and again 30 seconds later suddenly moved towards client C3.

Figures 3(a) and 3(b) show all multimedia network traffic received during the experiment by respectively client PA and client PB. We excluded the TCP data streams used by the NVE (e.g. streams containing state updates) from these network traces, because, compared to multimedia streams, they are negligible in terms of bandwidth usage.

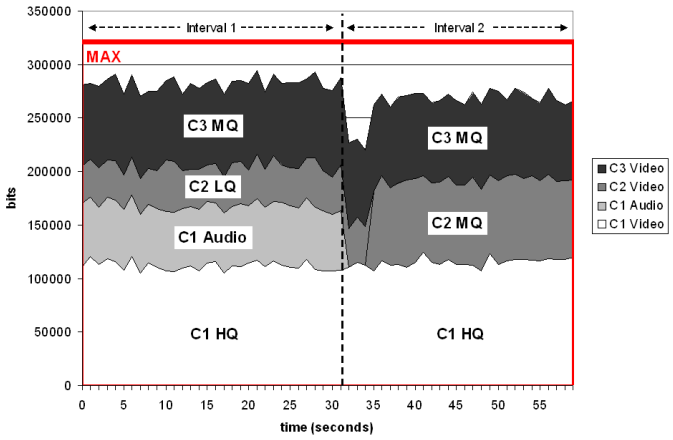
When examining the network traces, a first important observation is that both client PA and client PB stayed within their bandwidth limitations for the entire duration of the experiment. This is due to our middleware's networking awareness, which ensures there is never more data sent to a client than its network link can handle. Also notice that the middleware always leaves a small amount of the client's downstream bandwidth unused to ensure the client can withstand modest irregularities in the bandwidth usage of the multimedia streams he is currently receiving.

The second important observation is that our middleware at all times leverages its application awareness to intelligently distribute a client's downstream bandwidth over the different multimedia streams that are being exchanged in the application. For instance, figure 3(a) indicates that as PA moved away from C1 and towards C2, our middleware automatically reduced the amount of bandwidth allocated to streams sent out by C1 (i.e. PA started receiving the LQ video stream of C1 whereas he previously received the MQ version), and assigned the newly available bandwidth to C2's multimedia streams (i.e. PA started receiving the HQ video stream of C2 instead of the MQ version). As PA subsequently moved towards C3, the middleware adjusted the allocation of PA's downstream bandwidth accordingly. In particular, besides switching to the HQ version of C3's video stream, PA also started receiving C3's audio stream. Client PB on the other hand was located very close to C2 in the virtual world and remained stationary at this position during the experiment. Based on this information,

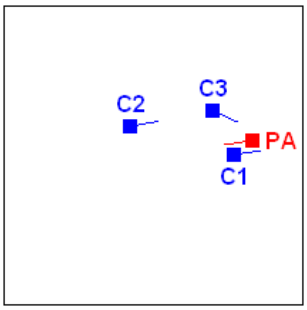
the middleware allocated the majority of PB's bandwidth to the multimedia streams sent out by this client. This is illustrated in the network trace shown in figure 3(b), which indicates PB received the HQ video stream and the audio stream of C2 for the entire duration of the experiment, whereas he received lower fidelity video from C1 and C2 during the first two intervals of the experiment (when the available downstream bandwidth was relatively low). In the second interval, our middleware even completely blocked C3's audio stream to preserve sufficient bandwidth for C2's streams.

To be able to fully comprehend the presented network traces, some additional information about both the experiment and the NVE application is required. First of all, the proxy server used in the experiment had our video transcoding plug-in installed. This plug-in contains specific knowledge of the NVE application (e.g. information about the three distinct video qualities) and in addition adds video transcoding functionality to our middleware. Consequently, the proxy server, whenever necessary, on-the-fly transcoded the HQ video streams sent out by video-based clients to generate the MQ and LQ variants. Again, we would like to refer the interested reader to [3] for more information about the implementation and benefits of this plug-in. Secondly, contrary to what one would expect, the NVE was configured to give video streams preference over audio in the experiment because it allows us to better demonstrate the capabilities of our networking middleware. The NILayer informed our middleware hereof, which in turn influenced the way our middleware distributed the downstream bandwidth of connected clients. In particular, it explains why, whenever a client's downstream bandwidth was inadequate, the middleware decided to block audio streams sooner than video streams, even though an audio stream and a LQ video stream require a comparable amount of bandwidth. Third, instead of requiring participating users to communicate verbally with each other during the experiment, we generated the audio streams artificially by placing microphones in front of a continuous sound source. Although the former approach would have produced more realistic results, it would also have resulted in highly irregular audio streams (i.e. when a user stops talking, the bandwidth usage of his voice stream becomes zero). Consequently, the resulting network traces would have been very complex and nearly impossible to comprehend. However, it is important to note in this context that our middleware can detect changes in stream bandwidth usage and, when it occurs, will adjust the bandwidth distribution of its connected clients accordingly. This is illustrated by the second experiment, which is described in the next paragraph. Fourth, the reason a relatively large amount of PB's downstream bandwidth remains unused during the third interval of the experiment is that PB at that moment is already receiving all multimedia streams at maximal quality. Finally, during the second interval, PA and PB had the same amount of downstream bandwidth at their disposal and in addition were located very close to each other in the virtual world. This explains why they received the same set of multimedia streams during this part of the experiment.

The second experiment, which we mention only briefly here, involved three video-based clients which did not make



(a) network packets received by PA



(b) 2D view of the virtual world

Fig. 4. Results of experiment 2: (a) network trace (stacked graph) showing all multimedia network packets received by client PA; (b) 2D view illustrating the client positioning in the virtual world.

use of our middleware (clients C1, C2 and C3), and only one client that did (client PA). Besides its video stream, client C1 also sent out an audio stream during the first 30 seconds of the experiment. All clients remained stationary in the virtual world as illustrated in figure 4(b). Finally, we set the downstream bandwidth of client PA to 320 Kbps for the entire duration of the experiment. As can be seen in the network trace shown in figure 4(a), our middleware automatically noticed that approximately halfway through the experiment client C1 stopped transmitting audio and, after a short transition period, reacted accordingly. In particular, it exploited the newly available bandwidth to transmit the MQ version of C2's video stream to PA, whereas this client previously only received the LQ variant of this stream. In other words, this experiment indicates that our networking middleware not only intelligently reacts to shifts in stream importance, but also to variations in stream bandwidth usage.

The experimental results presented in this section clearly demonstrate the benefits of our networking middleware. First of all, due to the middleware's network awareness, the current downstream capacity of a client's network connection will always be respected. Consequently, stream playback at client-side will normally improve, since all multimedia streams that are actually sent to a client should arrive there in time and with minimal transmission errors. Secondly, our middleware at all times exploits its application awareness to intelligently distribute a client's available downstream capacity over the different multimedia streams that are being exchanged inside

the application. As a result, clients will always receive the streams that are most important to them at a quality that is as high as possible within their current bandwidth limitations. Since developers no longer need to spend precious time on these issues, we can conclude that our networking middleware considerably facilitates the integration of efficient real-time streaming functionality in networked applications.

VI. Conclusions and future work

In this paper, we have presented our networking middleware consisting of a number of interconnected proxy servers that are both application and network aware. Through experimental results, we have demonstrated how the proxy servers exploit their compound awareness to dynamically and intelligently distribute a client's downstream bandwidth over the different multimedia streams that are being exchanged inside the application. In addition, we have also demonstrated that our middleware at all times respects the current downstream capacity of clients, this way ensuring that client network links are never overwhelmed with data they cannot receive anyway. Furthermore, due to its generic design, the middleware can be incorporated in a wide range of networked applications (including on-line multiplayer games), while the provided plug-in mechanism ensures that our middleware can attain a high level of performance in all these different situations. Finally, to guarantee easy integration in existing applications, we also implemented a Network Intelligence Layer which developers can utilize to take care of the communication between their client application and our networking middleware. Based on these observations, we believe our networking middleware enables developers to incorporate efficient real-time streaming facilities in their application with minimal effort.

It is important to note that the networking middleware presented in this paper is a work in progress. At the moment, we are investigating how we can efficiently add *device awareness* to it. In particular, we envision our networking middleware also taking the capabilities of the receiving client device into account when making routing and transcoding decisions. The first steps in this direction have already been taken, see [18]. Furthermore, we are currently also considering developing some additional plug-ins for our networking middleware, for instance a plug-in which adds 3D audio mixing functionality. Finally, we also plan to integrate our work into a "shared workspace" application that is currently under development at the HCI department of our research institute. The goal of this application is to efficiently support real-time meetings between both collocated and distributed team members. As a result, audio and video communication will play a crucial role in this application.

Acknowledgments

We wish to thank all the members of the NVE research group at the EDM for their help and support.

References

[1] "Physics, Gameplay and the Physics Processing Unit," White paper, AGEIA Technologies, Inc., March 2005. [Online]. Available: http://www.ageia.com/pdf/wp_2005_3_physics_gameplay.pdf

[2] M. Wijnants, P. Monsieurs, and W. Lamotte, "Improving the User Quality of Experience by Incorporating Intelligent Proxies in the Network," Expertise Centre for Digital Media (EDM), Tech. Rep. TR-LUC-EDM-0605, April 2005. [Online]. Available: http://research.edm.luc.ac.be/mwijnants/pdf/wijnantsTRMSAN.pdf

[3] M. Wijnants, P. Monsieurs, P. Quax, and W. Lamotte, "Exploiting Proxy-Based Transcoding to Increase the User Quality of Experience in Networked Applications," in *Proceedings of the 1st International Workshop on Advanced Architectures and Algorithms for Internet DElivery and Applications (AAA-IDEA'05)*, Orlando, Florida, June 2005.

[4] The EVE Online Technical FAQ. [Online]. Available: http://www.eve-online.com/faq/faq_07.asp

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

[6] M. Mauve, S. Fischer, and J. Widmer, "A Generic Proxy System for Networked Computer Games," in *Proceedings of the 1st Workshop on Network and System Support for Games (NetGames'02)*, Bruanschweig, Germany, 2002, pp. 25–28.

[7] C. D. Nguyen, F. Safaei, and P. Boustead, "A Distributed Proxy System for Provisioning Immersive Audio Communication to Massively Multi-Player Games," in *Proceedings of the 3rd Workshop on Network and System Support for Games (NetGames'04)*, Portland, Oregon, September 2004, p. 166.

[8] J. Smith, R. Mohan, and C.-S. Li, "Content-Based Transcoding of Images in the Internet," in *Proceedings of the IEEE International Conference on Image Processing (ICIP'98)*, vol. 3, Chicago, Illinois, 1998, pp. 7–11.

[9] B. Knutsson, H. Lu, J. Mogul, and B. Hopkins, "Architecture and Performance of Server-Directed Transcoding," *ACM Transactions on Internet Technology*, vol. 3, no. 4, pp. 392–424, 2003.

[10] A. Maheshwari, A. Sharma, K. Ramamritham, and P. Shenoy, "TranSquid: Transcoding and Caching Proxy for Heterogenous E-Commerce Environments," in *Proceedings of the 12th IEEE International Workshop on Research Issues in Data Engineering (RIDE)*, San Jose, California, February 2002, pp. 50–59.

[11] E. Amir, S. McCanne, and H. Zhang, "An Application Level Video Gateway," in *Proceedings of the 3rd ACM International Conference on Multimedia*, San Francisco, California, November 1995, pp. 255–265.

[12] B. Shen, S.-J. Lee, and S. Basu, "Caching Strategies in Transcoding-enabled Proxy Systems for Streaming Media Distribution Networks," *IEEE Transactions on Multimedia, Special Issue on Streaming Media*, vol. 6, no. 2, pp. 375–386, April 2004.

[13] X. Zhang, M. Bradshaw, Y. Guo, B. Wang, J. Kurose, P. Shenoy, and D. Towsley, "AMPS: A Flexible, Scalable Proxy Testbed for Implementing Streaming Services," in *Proceedings of the 14th International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV'04)*, Kinsale, Ireland, June 2004, pp. 116–121.

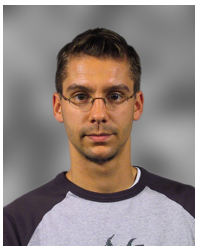
[14] J.-L. Huang, M.-S. Chen, and H.-P. Hung, "A QoS-Aware Transcoding Proxy Using On-demand Data Broadcasting," in *Proceedings of IEEE INFOCOM*, Hong Kong, China, March 2004.

[15] The Netfilter/IPTables Project Homepage. [Online]. Available: http://www.netfilter.org/

[16] P. Quax, T. Jehaes, P. Jorissen, and W. Lamotte, "A Multi-User Framework Supporting Video-Based Avatars," in *Proceedings of the 2nd workshop on Network and System Support for Games (NetGames'03)*, Redwood City, California, May 2003, pp. 137–147.

[17] P. Quax, C. Flerackers, T. Jehaes, and W. Lamotte, "Scalable Transmission of Avatar Video Streams in Virtual Environments," in *Proceedings of the IEEE International Conference on Multimedia & Expo (ICME'04)*, Taipei, Taiwan, June 2004.

[18] P. Quax, T. Jehaes, M. Wijnants, and W. Lamotte, "Mobile Adaptations for a Multi-User Framework Supporting Video-Based Avatars," in *Proceedings of the 9th International Conference on Internet & Multimedia Systems & Applications (IMSA'05)*, Hawaii, August 2005.



Maarten Wijnants graduated in computer science in 2003 at the Limburgs Universitair Centrum (LUC), Belgium. After a short stay at Androme NV, he is currently a PhD candidate at the EDM, a research institute of the LUC. His main research interest is computer networking, with specific interest in network architectures, multiplayer games, scalability, and making the network more intelligent. He is also interested in real-time computer animation and performing services for real-time data such as audio and video.