An Overlay Network Providing Application-Aware Multimedia Services

Maarten Wijnants[†‡] Bart Cornelissen[†‡] Wim Lamotte[†] Bart De Vleeschauwer[*‡]

[†]Hasselt University
Expertise Centre for Digital Media
and transnationale Universiteit Limburg
Wetenschapspark 2, BE-3590 Diepenbeek
firstname.lastname@uhasselt.be

[‡]Interdisciplinary institute
for BroadBand Technology
(IBBT)

[*]Department of Information Technology (INTEC)
Ghent University -IMEC
Gaston Crommenlaan 8 bus 201
B-9050 Gent, Belgium
bart.devleeschauwer@intec.ugent.be

ABSTRACT

Real-time streaming of multimedia content is increasingly becoming a crucial part of networked applications. A logical consequence of this evolution is a growing demand for services that can be applied on these multimedia streams. In this paper, we present our overlay network which provides such multimedia services. Although these services are application-aware and can thus exploit application-specific knowledge, the overlay network itself is completely generic. Consequently, multiple applications can take advantage of the overlay network, even concurrently. Furthermore, the overlay network is highly extensible, meaning additional services can be added to it easily. Besides describing the architecture of the software component of our overlay network, we also discuss the implementation of two example multimedia services. The first service mixes multiple audio streams into a single stream to enable lightweight voice communication in a Networked Virtual Environment. The second service applies face detection on a video stream to generate meaningful avatars in a meeting system called iConnect. The experimental results produced by these two services clearly demonstrate that our overlay network is capable of providing valuable services for a wide range of networked multimedia applications.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Network communications, Network topology*; C.2.3 [**Computer-Communication Networks**]: Network Operations—*Network monitoring*; C.2.4 [**Computer-Communication Networks**]: Distributed Systems

General Terms

Design, Experimentation, Human Factors, Performance

Keywords

Real-time multimedia streaming, overlay network, proxy server, application-aware service provision, Quality of Experience

AAA-IDEA '06 October 10, 2006, Pisa, Italy

1. INTRODUCTION AND RELATED WORK

Many of today's networked applications incorporate functionality to stream multimedia content such as digital audio and/or video in real-time. This functionality is mainly provided to allow distributed users to communicate in an efficient and natural manner. For instance, many on-line computer games as well as instant messaging applications support real-time voice communication between users. However, there also exist networked applications which use digital audio and/or video streaming for purposes other than simple user communication. One such application is Linden Lab's Second Life, a 3D Networked Virtual Environment (NVE) built entirely by its residents [16]. Besides many other features, Second Life allows its users to stream music into the virtual world, hereby enabling musically-minded people to share their creations with other Second Life users. We believe this non-trivial use of audio and video streaming will only increase in future networked applications. For instance, we envision games with a persistent virtual world periodically streaming audio or video bulletins to clients to inform them about remarkable events that have occurred in the virtual world.

However, as streaming of multimedia content becomes more popular, the demand for *services* that can be performed on these streams is likely to increase proportionally. The driving factor behind this evolution is the continuously diversifying market of end-user computation devices. Handheld devices such as PDAs and smartphones are rapidly increasing in popularity. In addition, these devices are being equipped with ever more extensive networking capabilities like, for instance, Wi-Fi and Bluetooth. As a result, users want to start using networked applications on their handheld devices. However, since the hardware capabilities of these devices differ heavily not only from each other but also from those of a standard PC, services are needed which can adapt multimedia streams so that they match the specific capabilities of the receiving end-user device. Besides end-users, the applications themselves sometimes also want to perform operations on multimedia streams. In an on-line computer game, for instance, effects or filters could dynamically be added to an audio stream to increase the realism and feeling of immersion for gamers.

To achieve optimal results and to maximize efficiency, these services should be implemented at some location inside the network, in the delivery path between the sender and the receiver of the multimedia stream. By choosing an in-network location, it suffices for senders to transmit only one version or quality of a multimedia stream, other versions or qualities which receivers may request will be generated by the in-network service on-the-fly. As a result, senders never need to waste bandwidth on sending multiple versions of the same multimedia content. Receivers on the other hand always receive the version or quality which they have requested

or that best matches the capabilities of their device, so no receiver bandwidth is wasted either. Other possible locations to apply multimedia services is at sender-side or at receiver-side, but these approaches are clearly not as flexible nor do they minimize the bandwidth usage as well as is possible with in-network located services.

In [19], we introduced our overlay network (there more generally termed networking middleware) consisting of a number of interconnected proxy servers and described its capabilities to intelligently manage user bandwidth in networked multimedia applications. In this paper, we shift the focus away from user bandwidth management and towards multimedia services. In particular, we demonstrate that our overlay network can be used to concurrently provide powerful multimedia services for various types of networked applications. This is due to the fact that our overlay network supports services that are application-aware and can thus exploit specific knowledge of the application they are serving. We present the implementation and the evaluation of two such services which are each targeted at a completely different networked application. However, these are only example services and additional services for other applications can be added to our overlay network easily.

A number of other proxy-based systems that provide services which can be performed on real-time multimedia streams have been described in the literature. Many of these systems focus on improving the end-to-end delivery or streaming of content by dynamically transcoding the content inside the network before it reaches the requesting user. Examples include TranSquid [9], Mocha [14], Intel's discontinued Quick Web technology [4], the AMPS framework [21], the transcoding middleware providing pervasive geospatial intelligence access described in [7], the context-aware decision engine described in [8], the video transcoding gateways presented in [6] and [1] and the adaptive QoS framework described in [15]. A more general approach is for instance taken by the Media Services Architecture (MSA) presented in [3]. However, our overlay network distinguishes itself from the cited systems by its application awareness. As will be discussed later on in this paper, the proxy servers of our overlay network combine a generic base layer with application-aware plug-ins (which in this case each implement a particular service). This design ensures that our overlay network can be integrated in a wide variety of networked multimedia applications and can provide valuable and efficient services in all these different situations. Of course, the overlay network could also provide more generic services that can be exploited by groups of applications, but the focus of this paper is on application-aware services targeted at a specific application.

The remainder of this paper is organized as follows. In section 2 we describe the general software architecture of our proxy servers. Section 3 describes two novel plug-ins for our overlay network which implement specific multimedia services for two completely different networked applications. An evaluation of these two services is provided next in section 4. Finally, we present our conclusions and suggest possible future research directions in section 5.

2. SOFTWARE ARCHITECTURE

Our proxy servers run on the GNU/Linux operating system. We chose this operating system for its extensive networking support in general and for its packet filtering and packet mangling functionality in particular. This latter functionality is provided by the Linux netfilter framework [10]. For each network protocol, netfilter defines a set of *hooks* for which kernel modules and userspace processes can register interest. Each hook is in fact a well-defined point in a packet's traversal of a particular network stack. As an example, the different hooks which netfilter defines for the IPv4 network stack are shown in figure 1. This figure illustrates, for instance, that an incoming IPv4 packet destined for a local process will always first pass the prerouting hook and subsequently the input hook before it is handed over to the receiving application.

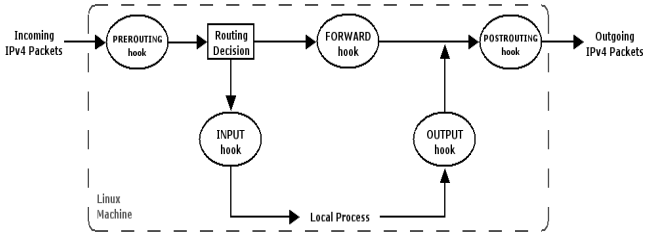


Figure 1: Netfilter IPv4 packet traversal diagram.

Each time a packet arrives at a certain hook, netfilter checks whether a kernel module or a userspace process has registered interest for this hook and network stack. If this is the case, the packet is transferred to this kernel module or userspace process, which can subsequently inspect the packet and process it, if necessary. After processing is complete, the (possibly modified) packet must be reinserted in the network stack together with a *verdict* to inform netfilter what should happen with the packet. The most important verdicts are *accept* and *drop*, which respectively inform netfilter that a packet should continue its traversal normally, and that a packet should be dropped at its current location in the network stack. Suppose for instance that a kernel module, which issues a drop verdict for certain packets based on some classification scheme, registers interest for the IPv4 input hook. As a result, packets matching the rules in the classification scheme will never be delivered to the local process for which they were originally destined.

With this background information about the netfilter framework in mind, we now move on to explain the general software architecture of our proxy servers, which is illustrated in figure 2. As can be seen in this figure, our proxy servers operate entirely in userspace. Although there is a small speed penalty associated with transferring packets from the kernel to a userspace process, we still preferred this approach over a kernel module. After all, the phrase "in kernelspace, a single wild pointer can wipe out your entire file system" unfortunately still applies, making it nearly impossible to implement applications as complex as our proxy server in kernelspace. Figure 2 also illustrates that our userspace process consists of two major subsystems. The first subsystem is a generic base layer which is completely application independent. The second subsystem is a plug-in mechanism through which additional functionality can be added to the proxy. The responsibilities of these two subsystems are described next.

The generic base layer first of all provides basic user management functionality. In particular, the base layer allows clients to connect to and communicate with our proxy server. Secondly, the base layer is responsible for receiving packets from kernelspace and for reinjecting packets in the kernel after they have been processed by our userspace process. As can be seen in figure 2, an incoming packet is only transferred to userspace if the packet belongs to a stream the proxy server is actually interested in. Other packets simply flow through our proxy servers without ever leaving kernelspace. To control which packets should be transferred to the userspace process, the generic base layer exploits the functionality provided by iptables, a packet selection system built on top of the netfilter framework [10]. Finally, the generic base layer is ca-

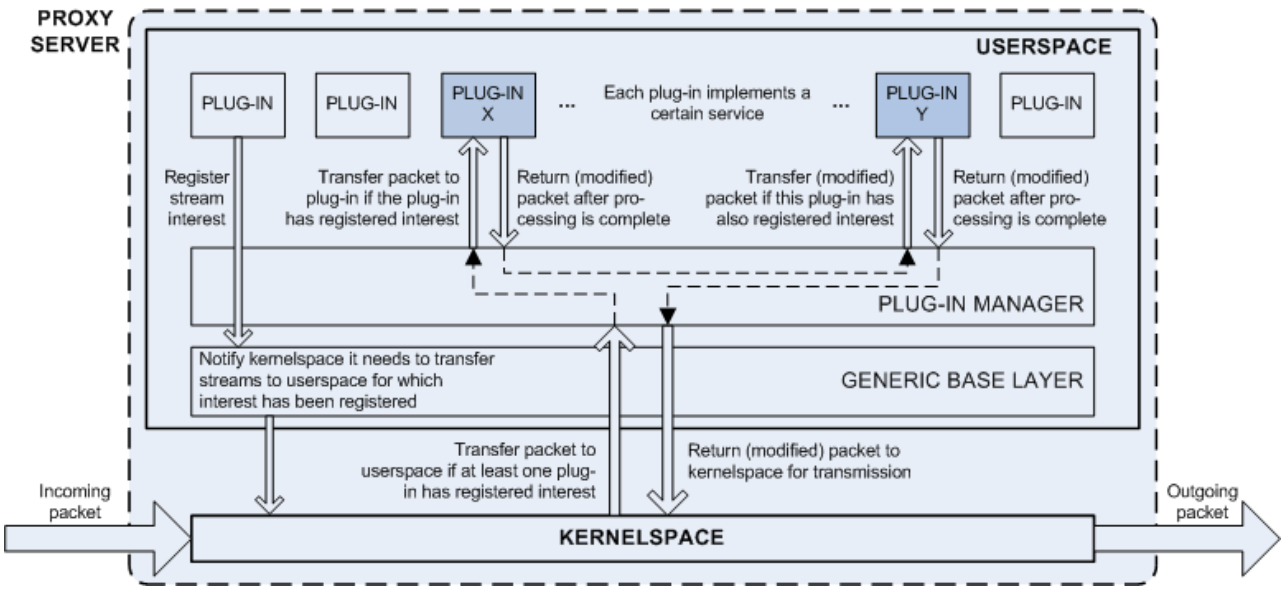


Figure 2: Schematic overview of the software architecture of the proxy servers.

pable of gathering information about the current state of the transportation network and about the relative importance of individual network streams (from a client's point of view). In [19], we described how our overlay network exploits this compound knowledge to intelligently manage user bandwidth in networked applications. However, since this is not the topic of this paper, we will not elaborate on it here but instead would like to refer interested readers to our previous work.

The main component of the plug-in subsystem is the *plug-in manager* which enables and manages the installation of *plug-ins* on our proxy servers. Firstly, plug-ins can register interest for certain network streams with the plug-in manager. The plug-in manager stores this information and relays a description of the stream to the generic base layer, which subsequently translates it to an appropriate iptables command to ensure packets belonging to the specified stream are transferred to userspace. Secondly, when a packet is actually transferred to userspace, it is the responsibility of the plug-in manager to hand it over to the the plug-in which has registered interest for the stream this packet belongs to. If more than one such plug-in is installed, the plug-in manager sequentially passes the packet to all these plug-ins, hereby following the order in which they have registered interest for the stream. After a plug-in has finished processing a packet, it needs to return the (possibly modified) packet to the plug-in manager and specify a verdict for it. The plug-in manager subsequently only continues the traversal of the other interested plug-ins if the previous plug-in returned an accept verdict. Otherwise, the packet is immediately dropped. Finally, when the last plug-in in the sequence of interested plug-ins is reached, the packet and the verdict returned by this plug-in are reinserted into kernelspace.

To further clarify the operation of the plug-in manager, consider the example scenario which is depicted in figure 2. As can be seen in this figure, two separate plug-ins (i.e. the highlighted plug-ins X and Y) have registered interest for the stream the incoming packet belongs to. However, plug-in X on the left was the first to do so. Consequently, the plug-in manager starts traversing the sequence of interested plug-ins by handing the packet to plug-in X. After processing is complete, plug-in X returns the packet together with an accept verdict. As a result, the plug-in manager subsequently transfers the packet to plug-in Y on the right. Since this is the last plug-in which has registered interest, the packet and the verdict returned by this plug-in are reinserted into the kernel. This example scenario clearly demonstrates that our plug-in manager allows multiple plug-ins to collaborate on a single network stream. Of course, collaborating plug-ins have to be carefully geared to each other. For instance, if plug-in X had returned drop instead of accept, plug-in Y would never receive the packet, even though it has registered interest for it.

It is important to notice that although the plug-ins physically reside on the proxy servers, they are conceptually part of the applications the overlay network is serving. Consequently, in contrast to the generic base layer, the plug-ins are application-aware and can thus exploit application-specific knowledge and functionality. For instance, in [20] we presented a video transcoding plug-in for our overlay network which belongs to an NVE application. This NVE application supports real-time video communication between users but requires them, for reasons of scalability, to send out not just one but three predefined qualities of the video stream captured by their webcam [12]. The video transcoding plug-in was implemented to relieve NVE clients from this burden. In particular, this plug-in generates the two other video qualities required by the NVE by on-the-fly transcoding the highest quality version of a user's webcam stream. As a result, scarce client upstream bandwidth is saved since it now suffices for NVE clients to transmit only this latter version of their video stream.

To summarize, our proxy servers consist of a generic base layer on top of which application-aware plug-ins can be installed. As will be demonstrated in the next section, this design enables our overlay network to provide powerful and efficient multimedia services for a wide range of networked applications. In other words, the software architecture of our proxy servers eliminates the need for deploying and managing a separate overlay network for each distinct networked application. Once our overlay network has been set up, providing a service for a certain application boils down to implementing and installing a plug-in and ensuring that the overlay network is sufficiently capacitated. Of course, if performance is of

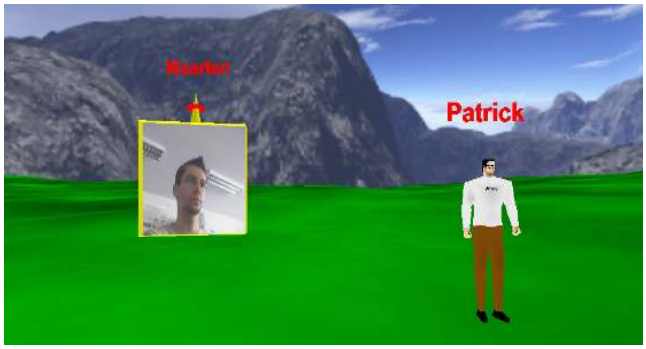


Figure 3: Screenshot of the EDM NVE framework running on a desktop PC.

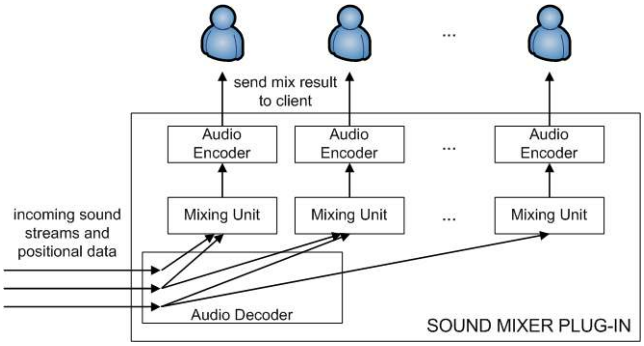


Figure 4: The sound mixer plug-in.

outmost importance, deploying a dedicated overlay network might still be preferable.

3. EXAMPLE SERVICES

3.1 Sound mixer plug-in

The first novel plug-in for our overlay network provides a sound mixing service for the NVE framework developed at the Expertise Centre for Digital Media. This NVE framework divides the virtual world into a number of regions, which each are assigned a set of unique multicast addresses. These multicast addresses are used to exchange different types of information and thus serve as the communication channel for the region they are associated with. For instance, there is a separate multicast address associated with each region to exchange state information of objects located in this region, while another multicast address is used to disseminate the audio (e.g. voice) streams originating from the region. The framework's awareness manager selects the regions a client should be aware of and only the multicast addresses associated with these regions are subsequently joined, hereby limiting the amount of data clients need to receive and process. A detailed description of the NVE framework can be found in [12], and a screenshot is shown in figure 3.

Although the NVE framework was originally developed for the PC platform, it was recently adapted for use on mobile devices (i.e. PDAs) [13]. However, due to the limited bandwidth and processing power of these devices, some sacrifices had to be made during the porting process. Probably one of the most drastic changes was the decision to limit the number of audio streams a mobile client can receive to one. This means that while a desktop user receives the audio streams sent out by all clients currently residing in the regions he is interested in, a mobile user only receives the audio stream of the object (e.g. avatar) he has selected on his screen. This decision was made based on the high bandwidth and processing requirements associated with receiving and decoding multiple audio streams simultaneously. Furthermore, received audio streams need to be mixed together locally to obtain the final output signal, which again requires processing power.

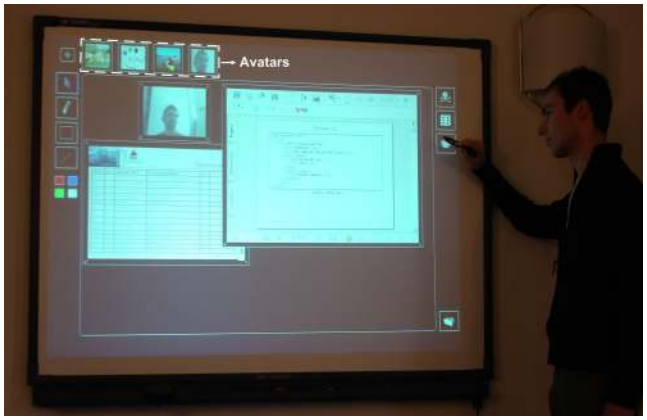
The above discussion clearly indicates that a substantial difference exists between the audio experience available to respectively desktop and mobile users of the NVE framework. The sound mixer plug-in was designed to bridge this gap without increasing the bandwidth or processing requirements for the mobile device. A high-level overview of the operation of this plug-in is illustrated in figure 4. As can be seen in this figure, the sound mixer plug-in maintains a separate mixing unit for every client it serves and subscribes to the correct multicast addresses to ensure it receives all necessary input audio streams. Incoming audio streams are decoded and subsequently only transferred to the mixing units associated with clients that are actually interested in this stream. As a result, a unique audio mix is created for each client served by the plug-in. Furthermore, since the mixing units also take positional information into account, they produce an output stream in which the contributing audio sources are correctly localized in 3D space. The sound mixer plug-in relies on JVOIPLIB [5] for the mixing of the audio streams and on the Speex library [17] for compressing the mix result.

3.2 iConnect plug-in

The second plug-in belongs to a meeting system called iConnect. The overall goal of this project is to enable both collocated and geographically dispersed participants to meet in an efficient manner [2]. To achieve this goal, a computer-augmented meeting room has been set up at our research institute containing a touch-sensitive whiteboard. This whiteboard acts as a *shared workspace* since it allows users to simultaneously interact with the data presented on it. Furthermore, the iConnect project also supports the notion of *personal workspaces* by allowing participants who physically attend the meeting to bring along their personal devices and connect them to the iConnect system through a wireless network. Once connected, participants can easily transfer files from their personal device to the shared workspace and vice versa through an intuitive drag-and-drop interaction technique. For instance, a participant could copy a document which is currently under discussion and thus displayed on the shared whiteboard to his personal device, alter or annotate it locally, and subsequently transfer the modified document back to the whiteboard to share his changes with the other meeting participants.

People who are restrained from physically attending the meeting, for instance due to excessive traveling costs, can still participate by remotely connecting to the iConnect system through a PC. In this case, the shared workspace is displayed on the screen of their workstation. Remote participants have disposal of the same collaboration possibilities as the participants in the iConnect meeting room. In addition, the iConnect system provides both audio and video conferencing support to allow remote and physically attending participants to communicate in an efficient manner.

An example iConnect meeting is shown in figure 5(a). In this meeting, an Adobe PDF document and a Microsoft Powerpoint slideshow are simultaneously being discussed. Furthermore, this snapshot also illustrates that each iConnect meeting participant is represented by an avatar at the top of the shared whiteboard. While physically attending participants are always represented by a static image, remote participants can also be represented by a *video*



(a) Snapshot of the shared whiteboard during an example iConnect meeting

(b) Comparison of the video conferencing stream of a remote participant and his automatically generated video avatar

Figure 5: The iConnect meeting system.

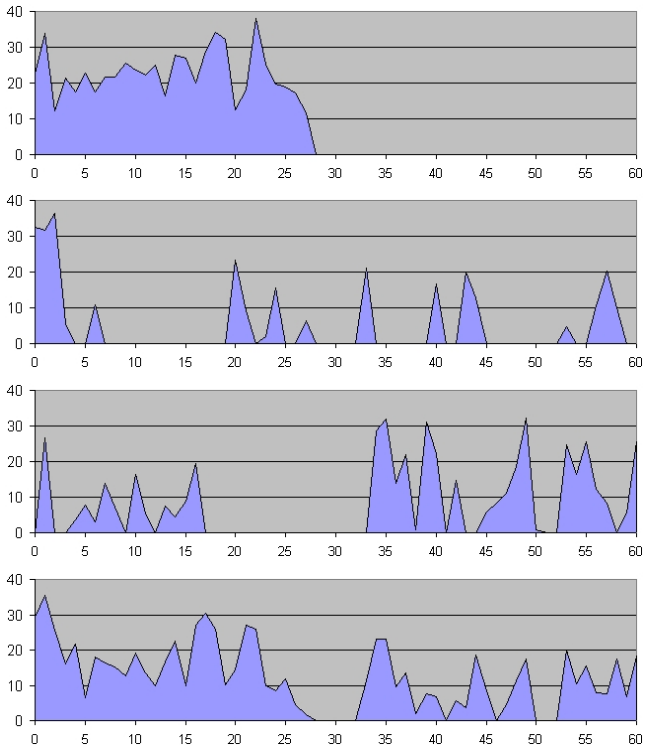


Figure 6: Audio network traffic generated during the sound mixer experiment. The horizontal axes represent the time (in seconds), the vertical axes the bit count (in Kilobits).

avatar. In this latter case, the avatar consists of a low-resolution and low-framerate live video stream of the participant's face. The fourth avatar in figure 5(a) belongs to a remote participant and is video-based. Notice that the video avatars are always displayed on the shared whiteboard, while the visualization of a remote participant's video conferencing stream can be turned on and off dynamically by the meeting moderator (i.e. the person standing at the whiteboard).

The video avatars representing remote participants are not sent out directly by these users but instead are automatically generated by the iConnect plug-in for our overlay network. In particular, this plug-in intercepts the video conferencing streams transmitted by remote participants, decodes them and applies face detection on them. The result is subsequently encoded at a much lower resolution and framerate (compared to the original video stream), and transmitted to the shared whiteboard in the iConnect meeting room. The differences between a remote participant's video conferencing stream and his video avatar, both in terms of encoding parameters and of image content, are illustrated in figure 5(b). The iConnect plug-in exploits the algorithms and functionality provided by the Open Computer Vision Library [11] to perform the face detection, and uses FFMPEG's codec library libavcodec [18] to decode incoming video streams and to encode the video avatars.

4. EVALUATION

4.1 Sound mixer plug-in

To demonstrate the sound mixer plug-in, we performed an experiment which involved three audio transmitting clients. The audio network traffic generated by these clients is shown in the first three graphs in figure 6. The first audio transmitting client was a continuous audio source which was shut down approximately 30 seconds after the experiment had started. The two other audio streams were generated by two verbally communicating NVE users, which resulted in alternating bursts of voice data (see graphs 2 and 3). The audio output of these three clients was processed by the sound mixer plug-in and the mix result was subsequently transmitted to a fourth client. The audio network traffic received by this latter client is shown in the bottom graph in figure 6, which indicates that the sound mixer plug-in successfully combined the audio output of the three audio transmitting clients into a single stream. Notice that the bit counts in figure 6 were measured per one-second time interval, which is rather coarse. Consequently, narrow peaks at the sender, caused by a burst of audio data in between the bounds of one time interval, often resulted in smaller but wider peaks at the receiver if the data reception crossed the time interval boundary.

The experimental results presented in the previous paragraph demonstrate that the sound mixer plug-in is capable of improving the audio experience provided to mobile NVE users. In particular, instead of being limited to receiving the audio stream of a single remote client, mobile users now receive a single audio stream which contains all audio data they are interested in. This means that the sound mixer plug-in successfully eliminates the difference that existed between the audio experience available to respectively desktop and mobile users of the NVE framework. Furthermore, no additional bandwidth or processing requirements are introduced at client-side, since mobile users still need to receive and process only a single audio stream. A minor disadvantage of the sound mixer service is that it introduces a small amount of additional delay since the plug-in needs to buffer incoming audio streams for a

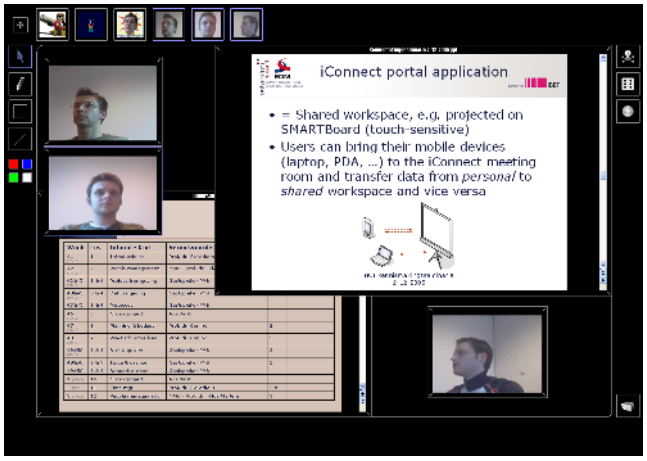


Figure 7: Visualizing the video streams of remote participants can consume a lot of space on the shared whiteboard.

short period of time so that they can be mixed together. This extra delay (less than 1 second) will however be negligible in most networked applications. Also note that although the sound mixer plug-in was described and explained from the point of view of mobile users, it can just as well be exploited by desktop users of the NVE framework.

4.2 iConnect plug-in

The iConnect plug-in for our overlay network in many cases eliminates the need for displaying the video conferencing stream of remote participants on the shared whiteboard. This is mainly true for situations in which only limited cooperation is required from these participants. In such situations, visualizing their full-blown video stream adds very little value, since the feedback provided by the video avatars will normally suffice to guarantee meeting effectiveness. As an example, consider the use case in which the iConnect meeting system is used to give a presentation or a lecture. The speaker, who is standing next to the shared whiteboard, has direct face-to-face contact with the other physically attending participants. The video avatars on the other hand also provide the speaker with facial feedback from remote participants. Consequently, displaying the video streams of remote participants becomes superfluous since the speaker can already quickly determine whether his presentation is understood by meeting participants, either attending or remote.

By eliminating the need for displaying the video streams of remote meeting participants, the iConnect plug-in saves screen space on the shared whiteboard. This is an important advantage, since whiteboard screen space is a scarce resource. After all, although the whiteboard has a large form factor, content needs to be displayed on it large enough so that it can be perceived clearly by all participants in the meeting room. For instance, figure 7 illustrates the screen space required to visualize the video streams of three remote participants. This is already a substantial amount, which will only increase as the number of remote participants rises. The iConnect plug-in is capable of freeing up some of this screen space, which can subsequently be used to display other content.

It is important to note that the service provided by the iConnect plug-in could also be implemented at client-side or at server-side. However, one of the design goals for the iConnect project was to keep the client software as lightweight as possible in order to maximize the number of devices through which users can remotely connect to the iConnect system. Although the client software currently only runs on PCs, the iConnect research group is investigating whether it can be ported to handheld devices. By supporting handheld devices, the iConnect system could address new user groups like, for instance, people on the move carrying a PDA. However, handheld devices are limited in terms of processing power, which makes it impractical to implement services such as the one provided by our iConnect plug-in at client-side. The iConnect server on the other hand already has a large number of responsibilities, which results in a high server processor load. Consequently, we decided an in-network location was the best position to implement the iConnect service.

Table 1: Performance evaluation of the implemented services.

	Average CPU utilization	Maximum CPU utilization
Sound mixer plug-in	1,5 %	4 %
iConnect plug-in	7,5 %	20 %
Both plug-ins loaded simultaneously	9,5 %	22 %

4.3 Performance evaluation

We tested the performance of the two described plug-ins by installing them on a single proxy server and measuring both the average and the maximum proxy CPU usage when performing the experiments described in sections 4.1 and 4.2. To recapitulate, the sound mixer experiment involved mixing together the audio streams of three NVE clients, while during the iConnect experiment the video avatars of three remote iConnect participants were generated. Furthermore, we also registered the proxy CPU utilization when both experiments were performed simultaneously. The results, measured on an Intel Pentium D 3 Ghz without any dedicated hardware and running linux kernel 2.6.14, are summarized in table 1. This table illustrates that when both plug-ins were active simultaneously, the proxy CPU usage approximately equalled the sum of the CPU usages when the two plug-ins were active separately. This indicates that the simultaneous loading of plug-ins introduces nearly no overhead.

5. CONCLUSIONS AND FUTURE WORK

We have presented our overlay network consisting of interconnected proxy servers. Each proxy server combines a generic base layer with a plug-in mechanism which enables the installation of additional functionality. However, besides ensuring extensibility, this plug-in mechanism also provides our overlay network with application awareness since plug-ins are application-aware and can thus exploit application-specific knowledge and functionality. This software architecture enables our overlay network to provide not only generic services but also high-performance services targeted at specific networked multimedia applications. As a result, the need for deploying and managing a separate overlay network for each distinct networked application is eliminated. This is confirmed by the experimental results produced by two example plug-ins that were also described in this paper. In particular, these plug-ins each provide a specific multimedia service for a completely different networked application.

It is important to note that our overlay network is a work in progress. As a result, we intend to improve it in several ways in the near future. For instance, we plan to implement an overlay routing plug-in which would enable our proxy servers to route network

packets around congested or broken network links. This plug-in could, for instance, be added at the end of the list of loaded proxy plug-ins and could subsequently register interest for every network stream that passes through the proxy server. As a result, the services provided by the other plug-ins would become less sensitive to network failure. Furthermore, we are going to continue integrating our work into the iConnect system to be able to perform some more extensive experiments.

6. ACKNOWLEDGMENTS

We wish to thank the members of both the NVE and the iConnect research group at the Expertise Centre for Digital Media for their help and support.

7. REFERENCES

[1] P. Aghera, A. Dixit, R. Oliveira, and V. Samanta. Wireless Middleware: Dynamic Video Transcoding. Project Report CS211, Computer Science Department, UCLA, 2003.

[2] M. Cardinaels, G. Vanderhulst, M. Wijnants, C. Raymaekers, K. Luyten, and K. Coninx. Seamless Interaction between Multiple Devices and Meeting Rooms. In *Proceedings of the CHI'06 Workshop on Information Visualization and Interaction Techniques for Collaboration across Multiple Displays (IVITCMD'06)*, Montreal, Canada, April 2006.

[3] M. Harville, M. Covell, and S. Wee. An Architecture for Componentized, Network-Based Media Services. In *Proceedings of the IEEE International Conference on Multimedia & Expo (ICME'03)*, pages 333–336, Baltimore, Maryland, July 2003.

[4] Intel News Release: New Intel Technology Speeds Delivery of Web Pages. `http://www.intel.com/pressroom/archive/releases/IN092597.HTM`.

[5] JVOIPLIB Homepage. `http://research.edm.uhasselt.be/jori/jvoiplib/`.

[6] Z. Lei and N. Georganas. Video Transcoding Gateway for Wireless Video Access. In *Proceedings of the IEEE Canadian Conference on Electrical and Computing Engineering (CCECE'03)*, Montreal, May 2003.

[7] C.-Y. Lin, A. Natsev, B. Tseng, M. Hill, J. Smith, and C.-S. Li. Content Transcoding Middleware for Pervasive Geospatial Intelligence Access. In *Proceedings of the IEEE International Conference on Multimedia & Expo (ICME'04)*, Taipei, Taiwan, June 2004.

[8] W. Y. Lum and F. Lau. A Context-Aware Decision Engine for Content Adaptation. *IEEE Pervasive Computing*, 1(3):41–49, 2002.

[9] 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'02)*, pages 50–59, San Jose, California, February 2002.

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

[11] Open Computer Vision Library Homepage. `http://sourceforge.net/projects/opencvlibrary/`.

[12] 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.

[13] 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)*, pages 412–417, Hawaii, USA, August 2005.

[14] R. Rejaie and J. Kangasharju. Mocha: A Quality Adaptive Multimedia Proxy Cache for Internet Streaming. In *Proceedings of the 11th International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV'01*, pages 3–10, Port Jefferson, NY, USA, June 2001.

[15] S. Rho, E. Hwang, and M. Kim. An Implementation of QoS Adaptive Multimedia Content Delivery. In *Proceedings of the 9th International Conference on Internet & Multimedia Systems & Applications (IMSA'05)*, pages 316–321, Hawaii, USA, August 2005.

[16] Second Life Homepage. `http://secondlife.com/`.

[17] Speex Homepage. `http://www.speex.org/`.

[18] The FFMPEG Homepage. `http://ffmpeg.sourceforge.net/`.

[19] M. Wijnants and W. Lamotte. Audio and Video Communication in Multiplayer Games through Generic Networking Middleware. In *Proceedings of the 7th International Conference on Computer Games (CGAMES'05)*, pages 52–58, Angoulême, France, November 2005.

[20] 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.

[21] 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)*, pages 116–121, Kinsale, Ireland, June 2004.