Exploiting Proxy-Based Transcoding to Increase the User Quality of Experience in Networked Applications

Maarten Wijnants Patrick Monsieurs Peter Quax Wim Lamotte
Expertise Centre for Digital Media
Limburgs Universitair Centrum
Universitaire Campus, B-3590 Diepenbeek, Belgium
{maarten.wijnants, patrick.monsieurs, peter.quax, wim.lamotte}@luc.ac.be

Abstract

In this work we describe how we extended the functionality of our previously introduced application and network aware proxy system with transcoding capabilities. Based on their compound awareness, our proxies devise a strategy for every connected client that intelligently distributes the client’s available bandwidth over the different application data streams. The proxies subsequently put each computed bandwidth allocation strategy into effect by managing and transcoding data streams inside the network before they reach the client. We also present results from the integration of our work into an existing Networked Virtual Environment. These results clearly indicate that our intelligent proxy, enhanced with its novel transcoding functionality, can increase the user Quality of Experience considerably in networked applications.

1. Introduction

The last few years, we are witnessing an ever increasing heterogeneity in the end-user device space. As a result, users can nowadays choose from a multitude of devices, each with their own specific capabilities and constraints, to connect to the Internet. Examples include desktop PCs, laptops, Personal Digital Assistants (PDAs) and smartphones. While there are many advantages associated with this increased client diversity, it also complicates multimedia content delivery. After all, multimedia content suitable for presentation on a desktop PC often cannot be displayed efficiently on a mobile device.

This problem can be resolved in two different ways. The first solution consists of supplying multiple versions of the same content and providing users with the version that best suits the capabilities of their client device and network connection. For example, content providers could provide both low and high resolution versions of a video fragment. Whenever a wireless PDA user requests this video fragment, he should be served the low resolution version due to the form factor of his device and the low throughput of his network link. On the other hand, whenever a PC user with a broadband Internet connection requests this particular video fragment, he should be served the high quality version. The main drawback of this approach is its lack of flexibility. If a new device is released which has capabilities unlike any of the devices already on the market, this device cannot be served an optimal version of the content unless a new version is explicitly supplied by the content provider. Furthermore, this approach makes content management difficult and error-prone since multiple versions of the same content reside on the origin server.

A second, more dynamic solution consists of transcoding the content on-the-fly to a convenient format before it reaches the end-user. A transcoder can take the constraints of both the transportation network and the receiving end-user device into account when transforming content on behalf of a client. As a result, there is no need to store multiple versions of the same content on the origin server. Furthermore, this approach is also flexible in the sense that content can be adapted in real-time to optimally match the capabilities and constraints of any device. On the other hand, an important disadvantage of transcoding is its computational complexity. Transcoding is a very processor intensive task, which can result in scalability issues in case large numbers of simultaneous users need to be supported. Despite this drawback, we believe transcoding to be a very powerful technique whose usability is not confined to content delivery scenarios.

In [21], we discussed the implementation of an extensible software proxy that is both application and network aware. This means the proxy on the one hand has high-level knowledge of the application it is serving. In case of a Networked Virtual Environment (NVE) or Massively Multiplayer Online Game (MMOG), this application related
knowledge could for instance include information about the positions and orientations of users in the virtual world. On the other hand, the intelligent proxy periodically probes the underlying transportation network to obtain network related information such as the current throughput and packet loss rate of individual network links. In the original version of the proxy, this compound awareness was exploited by a rather basic quality selection plug-in to intelligently manage and control data streams destined for connected clients [21]. This plug-in had a very coarse granularity, since it was only capable of turning data streams on and off.

In this paper, we describe the implementation of a much more flexible plug-in that adds transcoding functionality to our intelligent proxy. Based on its compound awareness, the proxy is now capable of intelligently transcoding data streams to lower quality versions as they become needed by connected clients. Furthermore, we investigate whether our transcoding-enabled proxy could positively affect the user Quality of Experience (QoE) in networked applications such as NVEs and MMOGs. QoE is a subjective measure that relates to the experience people have when they use an application or a product.

The remainder of this paper is organized as follows. In section 2 we review related work on the distributed proxy network architecture and on content-based transcoding (with an emphasis on video transcoding). Section 3 describes the implementation of our intelligent proxy and its video transcoding plug-in. The NVE framework we integrated our work in is introduced briefly in section 4. In section 5 we present some experimental results which clearly demonstrate that our intelligent proxy, enhanced with transcoding functionality, can increase the user QoE considerably in networked applications. This section also describes the approach we employed to maximize the proxy’s scalability. Finally, we pose our conclusions and suggest possible future research directions in section 6.

2. Related work

In comparison with the client/server and peer-to-peer architectures, the distributed proxy architecture is a relatively new network topology. In this architecture, a number of interconnected proxies are placed at the edge of the network, and clients only communicate with the proxy nearest to them. This is illustrated in figure 1. The main advantages of the distributed proxy architecture are increased system robustness (there is no single point-of-failure in the system) and reduced packet delay (proxies can cache data locally, this way decreasing the amount of time needed for data to reach requesting clients) [2][10]. Furthermore, proxies can perform valuable services such as packet filtering and packet aggregation inside the network, thereby making the network more intelligent. In [3], the results of porting a popular multi-player FPS game to the distributed proxy architecture are described. Nguyen et al. discuss in [12] how the distributed proxy architecture can be employed to provide immersive audio communication to users of online games. Finally, a flexible and highly modular proxy platform called AMPS is introduced in [22]. The platform supports an extensible set of streaming services and has been specifically designed with scalability in mind. However, in contrast to our intelligent proxy, the AMPS proxies at the moment still lack transcoding functionality.

On-the-fly transcoding of content to a convenient format is steadily receiving more and more attention from the research community. This is largely due to the popularization of mobile devices. In [17], an early content-based image transcoder is described which takes the capabilities of the receiving client device and the type and purpose of the image into account. The authors demonstrate clear improvements in both image delivery speed and image accessibility. A transcoding and caching intermediary for web content called TranSquid is discussed in [9]. TranSquid tries to increase the user satisfaction by servicing client requests from its cache as much as possible, hereby transcoding the cached content to an appropriate format if necessary. Knutsson et al. introduce in [6] the concept of server-directed transcoding, apply it to web content, and show how it can be integrated into the HTTP protocol. In the server-directed transcoding approach, the origin server guides the transcoder to make sure the content does not lose its semantics when it is being adapted to match the capabilities and preferences of the requesting client. Finally, a content transcoding middleware that aims at enabling universal access to military geospatial information (such as maps) is presented in [8]. The middleware represents the geospatial data as Scalable Vector Graphics, an XML-based W3C standard, and uses Java plug-ins and XSL stylesheets to...
transcode these SVG files to a format that suits the capabilities and preferences of the requesting client.

The advent of broadband access networks has led to a serious increase in the number of video files accessible through the Internet. This in turn boosted video transcoding research. Vetro et al. give in [20] an excellent overview of video transcoding architectures and techniques for block-based video coding schemes. Bit-rate reduction, spatial and temporal resolution reduction and error-resilience transcoding of video streams are discussed. In [1], a video gateway is presented that addresses the problem of heterogeneous client environments by intelligently transcoding and rate-controlling video streams. Another video transcoding gateway is described in [7]. The specific focus here is to enable video access on mobile devices. Finally, Shen et al. discuss in [15] and [16] their Transcoding-enabled Caching proxy (TeC proxy). The TeC proxy performs transcoding as well as caching to improve the efficiency of video delivery to heterogeneous end-users with various network conditions and device capabilities. Furthermore, the authors also introduce three novel caching algorithms which take the transcoding functionality of the proxy into account. The presented results demonstrate that these new algorithms are capable of seriously outperforming traditional caching strategies.

There is a large resemblance between the work we present in this paper and the systems described in [1],[7],[15],[16]. Some of these systems are even more sophisticated in terms of transcoding functionality and transcoding performance. However, the main difference between our work and the referred systems is that the latter focus solely on content delivery. While our intelligent proxy architecture can also be used in content delivery scenarios, it can just as well be integrated in more complex networked applications such as NVEs and videoconferencing programs. This is mainly due to the fact that our proxy, in contrast to the other systems referred in this section, also leverages application related information when making routing and transcoding decisions. We believe application awareness is crucial to provide users of networked applications with a maximal QoE, and we will demonstrate this in section 5.

3. Implementation

3.1. Intelligent proxy implementation

As described in [21], our proxy system is both application and network aware. In order to gain application awareness, we proposed to insert a network intelligence layer (NI layer) between the transport layer and the application layer of the client software. The main responsibility of the NI layer consists of continuously querying the application’s awareness management model in order to obtain information regarding the relative importance of the different data streams present in the system. This information is subsequently transmitted to the proxy this client is connected to. For example, in case of an NVE, the awareness management model could specify that data streams belonging to objects which are located close to the local user in the virtual world have a higher significance than streams belonging to more distant objects. Since the NI layer is highly reusable, it should be possible to integrate our proxy architecture in nearly any networked application with minimal effort. Network awareness on the other hand is gained by periodically measuring the latency, throughput and packet loss rate of the network links going to the clients currently connected to the proxy. Furthermore, each proxy records the bandwidth usage of every data stream that passes through it.

Our proxies continuously combine their application and network awareness to devise a strategy for every connected client that intelligently distributes the client’s available bandwidth over the different streams present in the system. For example, when a client’s available downstream bandwidth no longer suffices to receive all streams at maximum quality, the proxy will determine which streams should be reduced in quality or even completely blocked. This decision will be based on the relative importance of the streams as well as their bandwidth requirements. As a result, whenever possible, less important streams will be transcoded or dropped by the proxy before more significant streams are altered, this way maximizing the user QoE. The algorithm used to compute the bandwidth allocation strategies is described in [11].

To implement the proxy’s packet filtering, routing and mangling functionality, we used the netfilter/iptables framework [19] which is part of the Linux kernel since version 2.4. Netfilter defines a set of hooks for a number of network stacks (for example, the five hooks defined for IPv4 are illustrated in figure 2) and allows kernel modules to register callback functions for these hooks. Whenever a packet arrives at a hook, netfilter checks if anyone has registered a callback function for it. If this is the case, the packet is transferred from the network stack to this callback function, which can then examine the packet and subsequently decide on its fate. Iptables on the other hand enables packet selection in userspace. More specifically, iptables allows users to
insert and delete rules from the kernel’s packet filtering table for any of the hooks defined by netfilter. A rule specifies what should be done with a packet if its header matches the conditions specified in the rule. Possible verdicts a rule can issue for a packet include ACCEPT (let the packet pass), DROP (discard the packet) and QUEUE (hand the packet over to a userspace application which can then decide what should be done with the packet). If a packet does not match the conditions of the current rule, the next rule for that hook is consulted.

Iptables is used by our proxies to do some trivial packet filtering directly (e.g. always issue an ACCEPT verdict for network probes), and to QUEUE packets belonging to data streams of the served application to a userspace program [21]. Possible data streams in networked multi-user applications include audio, video, geometry, and position and orientation updates. The userspace application subsequently consults the computed bandwidth allocation strategy for the client this stream applies to, and leverages this information to decide if the packet should be accepted, dropped or transcoded. If a packet does not match the conditions of the current rule, the next rule for that hook is consulted.

Iptables is used by our proxies to do some trivial packet filtering directly (e.g. always issue an ACCEPT verdict for network probes), and to QUEUE packets belonging to data streams of the served application to a userspace program [21]. Possible data streams in networked multi-user applications include audio, video, geometry, and position and orientation updates. The userspace application subsequently consults the computed bandwidth allocation strategy for the client this stream applies to, and leverages this information to decide if the packet should be accepted, dropped or transcoded. If a packet does not match the conditions of the current rule, the next rule for that hook is consulted.

Iptables is used by our proxies to do some trivial packet filtering directly (e.g. always issue an ACCEPT verdict for network probes), and to QUEUE packets belonging to data streams of the served application to a userspace program [21]. Possible data streams in networked multi-user applications include audio, video, geometry, and position and orientation updates. The userspace application subsequently consults the computed bandwidth allocation strategy for the client this stream applies to, and leverages this information to decide if the packet should be accepted, dropped or transcoded. If a packet does not match the conditions of the current rule, the next rule for that hook is consulted.

Iptables is used by our proxies to do some trivial packet filtering directly (e.g. always issue an ACCEPT verdict for network probes), and to QUEUE packets belonging to data streams of the served application to a userspace program [21]. Possible data streams in networked multi-user applications include audio, video, geometry, and position and orientation updates. The userspace application subsequently consults the computed bandwidth allocation strategy for the client this stream applies to, and leverages this information to decide if the packet should be accepted, dropped or transcoded. If a packet does not match the conditions of the current rule, the next rule for that hook is consulted.

Iptables is used by our proxies to do some trivial packet filtering directly (e.g. always issue an ACCEPT verdict for network probes), and to QUEUE packets belonging to data streams of the served application to a userspace program [21]. Possible data streams in networked multi-user applications include audio, video, geometry, and position and orientation updates. The userspace application subsequently consults the computed bandwidth allocation strategy for the client this stream applies to, and leverages this information to decide if the packet should be accepted, dropped or transcoded. If a packet does not match the conditions of the current rule, the next rule for that hook is consulted.

Iptables is used by our proxies to do some trivial packet filtering directly (e.g. always issue an ACCEPT verdict for network probes), and to QUEUE packets belonging to data streams of the served application to a userspace program [21]. Possible data streams in networked multi-user applications include audio, video, geometry, and position and orientation updates. The userspace application subsequently consults the computed bandwidth allocation strategy for the client this stream applies to, and leverages this information to decide if the packet should be accepted, dropped or transcoded. If a packet does not match the conditions of the current rule, the next rule for that hook is consulted.

Iptables is used by our proxies to do some trivial packet filtering directly (e.g. always issue an ACCEPT verdict for network probes), and to QUEUE packets belonging to data streams of the served application to a userspace program [21]. Possible data streams in networked multi-user applications include audio, video, geometry, and position and orientation updates. The userspace application subsequently consults the computed bandwidth allocation strategy for the client this stream applies to, and leverages this information to decide if the packet should be accepted, dropped or transcoded. If a packet does not match the conditions of the current rule, the next rule for that hook is consulted.

Iptables is used by our proxies to do some trivial packet filtering directly (e.g. always issue an ACCEPT verdict for network probes), and to QUEUE packets belonging to data streams of the served application to a userspace program [21]. Possible data streams in networked multi-user applications include audio, video, geometry, and position and orientation updates. The userspace application subsequently consults the computed bandwidth allocation strategy for the client this stream applies to, and leverages this information to decide if the packet should be accepted, dropped or transcoded. If a packet does not match the conditions of the current rule, the next rule for that hook is consulted.

Iptables is used by our proxies to do some trivial packet filtering directly (e.g. always issue an ACCEPT verdict for network probes), and to QUEUE packets belonging to data streams of the served application to a userspace program [21]. Possible data streams in networked multi-user applications include audio, video, geometry, and position and orientation updates. The userspace application subsequently consults the computed bandwidth allocation strategy for the client this stream applies to, and leverages this information to decide if the packet should be accepted, dropped or transcoded. If a packet does not match the conditions of the current rule, the next rule for that hook is consulted.

Iptables is used by our proxies to do some trivial packet filtering directly (e.g. always issue an ACCEPT verdict for network probes), and to QUEUE packets belonging to data streams of the served application to a userspace program [21]. Possible data streams in networked multi-user applications include audio, video, geometry, and position and orientation updates. The userspace application subsequently consults the computed bandwidth allocation strategy for the client this stream applies to, and leverages this information to decide if the packet should be accepted, dropped or transcoded. If a packet does not match the conditions of the current rule, the next rule for that hook is consulted.

Iptables is used by our proxies to do some trivial packet filtering directly (e.g. always issue an ACCEPT verdict for network probes), and to QUEUE packets belonging to data streams of the served application to a userspace program [21]. Possible data streams in networked multi-user applications include audio, video, geometry, and position and orientation updates. The userspace application subsequently consults the computed bandwidth allocation strategy for the client this stream applies to, and leverages this information to decide if the packet should be accepted, dropped or transcoded. If a packet does not match the conditions of the current rule, the next rule for that hook is consulted.

Iptables is used by our proxies to do some trivial packet filtering directly (e.g. always issue an ACCEPT verdict for network probes), and to QUEUE packets belonging to data streams of the served application to a userspace program [21]. Possible data streams in networked multi-user applications include audio, video, geometry, and position and orientation updates. The userspace application subsequently consults the computed bandwidth allocation strategy for the client this stream applies to, and leverages this information to decide if the packet should be accepted, dropped or transcoded. If a packet does not match the conditions of the current rule, the next rule for that hook is consulted.

Iptables is used by our proxies to do some trivial packet filtering directly (e.g. always issue an ACCEPT verdict for network probes), and to QUEUE packets belonging to data streams of the served application to a userspace program [21]. Possible data streams in networked multi-user applications include audio, video, geometry, and position and orientation updates. The userspace application subsequently consults the computed bandwidth allocation strategy for the client this stream applies to, and leverages this information to decide if the packet should be accepted, dropped or transcoded. If a packet does not match the conditions of the current rule, the next rule for that hook is consulted.
which the event originated. Each client has an Area of Interest (AoI) manager that is responsible for constantly determining the regions this user should be aware of. As the user moves around the world, the multicast groups associated with the regions selected by the AoI manager are dynamically joined and left, this way limiting the amount of information clients need to receive and process. However, the framework also divides the world into video regions. Each user represented by a video-based avatar in the virtual world sends out three distinct qualities (high, medium and low) of his video stream, one to each of the three multicast addresses associated with the video region he is currently located in. Table 1 shows the different quality parameters the framework utilizes to encode the three versions of a user's video stream. Like the AoI manager, the Video Area of Interest (V AoI) manager decides which video multicast addresses the client should subscribe to. One strategy could be to subscribe to the high quality multicast group of the video region the user currently is in, and to the medium or even low quality groups of adjacent video regions.

### 5. Experimental results

A detailed discussion of the integration of our proxy system into the multi-user framework described in the previous section can be found in [21]. However, the results presented in that paper were obtained by using a simple video quality selection plug-in for our proxies. This plug-in exploited the proxy's application and network awareness to improve the QoE for connected users by determining which quality (if any) they should receive from every video-based avatar in the virtual world. Based on the decisions taken by the plug-in, the proxy subsequently dropped certain video qualities before they reached the user. Note that since we position our intelligent proxies close to end-users, the complexity of stream management is moved to the edge of the network where the number of streams is relatively low.

Although the quality selection plug-in worked fine, users represented by video-based avatars were still required to send out three distinct qualities of their video stream. This requirement places an extra load on the processor of the client device, since the user's video stream needs to be encoded three separate times. More importantly however, this approach also requires a large amount of client up-stream bandwidth. Most Internet Service Providers (ISPs) provide asymmetric Internet subscriptions in which the up-stream capacity is merely a fraction of the offered down-stream bandwidth (typically 128 or 192 Kbps). As a result, forcing clients to send out three different qualities of the same video stream is something commercial applications just cannot afford at the moment. By equipping our proxy system with transcoding functionality, we can relieve framework clients from this burden. After slightly modifying the general transcoding plug-in described in section 3, we obtained the plug-in shown in figure 5. This plug-in intercepts high quality encoded video streams (which are now the only video streams clients need to send out), transcodes them to medium and low quality versions if at least one of the proxy’s connected clients is interested in that quality, and subsequently serves every client the appropriate version.

To assess the value our new plug-in adds to the framework, we performed an experiment which involved four clients running the unmodified version of the framework (clients C1 to C4 in figure 6(a)) and two clients connected through our transcoding-enabled proxy (clients PA and PB in figure 6(a)). Only clients C1, C2 and C3 were represented as video-based avatars. During the experiment, we artificially varied the available downstream bandwidth of clients...
PA and PB over time, while all other clients remained stationary in the virtual world. Notice that since in this experiment the virtual world was inhabited by both standard as well as proxy-enhanced clients, the video-based clients still needed to send out three distinct video qualities. If however the world would have been populated solely by clients connected through our intelligent proxy (which is actually the scenario we envision), it would have sufficed for video-based clients to only send out the high quality encoded version of their video stream.

At the beginning of the experiment, we set the bandwidth of clients PA and PB to 320 Kbps. Due to its network awareness, the proxy noticed this bandwidth did not suffice for PA and PB to receive the high quality video streams of all three video-based avatars. As a result, our proxy exploited its application awareness to transcode certain video streams to medium or low quality, and subsequently transmitted these transcoded versions to its clients instead of their high quality counterparts. After 30 seconds, we limited PA’s and PB’s bandwidth to 250 Kbps, and after 60 seconds we further decreased their bandwidth to 170 Kbps. Finally, after 90 seconds, we increased the downstream bandwidth of PA and PB back to 400 Kbps.

Figures 6(b) and 6(c) show all video network traffic received by respectively client PA and client PB. A first important observation is that PA and PB both roughly stayed within their bandwidth limitations for the entire duration of the experiment. Secondly, these network traces also clearly indicate how our proxy exploits its application awareness to intelligently decide which video streams should be transcoded to a lower quality in case of insufficient client bandwidth. For example, since the distance between PA and C1 was much larger than the distance between PA and C2 and C3, the proxy allocated most of PA’s available bandwidth to the video streams sent out by C2 and C3 (see figure 6(b)). On the other hand, since PB was located very close to C1 in the virtual world, the majority of PB’s downstream bandwidth was allocated to C1’s...
video stream (see figure 6(c)). In contrast, figure 6(d) illustrates all video packets received by client C4. Since this client was not connected through a transcoding-enabled proxy, its VAOI manager was responsible for performing video quality selection. Unlike our proxy however, the VAOI manager lacks network awareness. As a result, client C4 received the same set of video streams during the entire experiment. If this client would have been subject to the same bandwidth fluctuations as clients PA and PB, its bandwidth would have been exceeded numerous times since no attempts would have been made to adjust the bandwidth usage of individual video streams. Exceeded client downstream bandwidth results in increased packet loss and higher video packet delay, which seriously deteriorates the video playback and consequently also the user’s immersive experience.

By comparing the network traces in figures 6(b) and 6(c) with the one in figure 6(d), the added value of our work becomes clear. Our transcoding-enabled proxy at all times exploits all available client bandwidth without ever exceeding it, and intelligently and dynamically distributes this bandwidth over all involved network streams in order to optimize the user QoE. Unsurprisingly, these results are almost identical to those presented in [21]. However, as stated before, scarce upstream bandwidth is no longer wasted since clients normally no longer need to send out different qualities of the same video stream. Furthermore, these results also illustrate the flexibility of our proxy system. By simply installing a new plug-in, the functionality of our proxy system was extended with transcoding capabilities.

A drawback of the new plug-in is its computational complexity. As stated before, video transcoding is very processor intensive, with the cascaded pixel-domain approach being the least efficient of all transcoding architectures. As a result, the transcoding plug-in is much less scalable than the previously developed quality selection plug-in. Although transcoding performance was not one of our main objectives, we nonetheless wanted the number of clients our proxy could support simultaneously to be as high as possible. We therefore incorporated an intelligent scaling mechanism in the transcoding plug-in. If over a short time interval the proxy’s average processor usage exceeds a predefined threshold, medium quality transcoding is disabled for all video streams and clients are served the low quality version instead. This has two advantages. First of all, low quality encoding is slightly less processor intensive than medium quality encoding. Secondly, by disabling medium quality encoding, transcoding results can be reused to a higher degree. For example, suppose two clients are connected to the same proxy, with one client being interested in the medium quality of a certain video stream while the other is interested in the low quality version. If the proxy has sufficient processor time available, it transcodes the high quality video stream to medium as well as low quality and serves both clients the quality they prefer. However, if the proxy’s threshold is exceeded, only low quality transcoding is enabled and both clients will consequently receive the low quality version of the video stream, resulting in a decrease in processor load by a factor of two approximately. This is illustrated in figure 7. Here, the threshold was set to an unusually low 20 percent processor usage for testing purposes only. Normally, the threshold value would be much higher, like for instance 80 or 90 percent processor load.

6. Conclusions and future work

In this paper, we have described the implementation of a video transcoding plug-in for our generic proxy system. Furthermore, we have also presented results from integrating our transcoding-enabled proxies in an existing NVE framework. These results are very similar to those achieved by a previously developed quality selection plug-in. More specifically, both plug-ins can equally improve the user QoE in networked multimedia applications. However, the transcoding plug-in relieves client machines from having to send multiple versions of the same video stream. We believe this to be an interesting advantage due to the asymmetric nature of most current Internet subscriptions. On the other hand, since transcoding is computationally expensive, the transcoding plug-in is less scalable than the quality selection plug-in. We have therefore equipped the transcoding plug-in with an intelligent scaling mechanism to push the number of clients our proxies can handle simultaneously to the limit. Finally, the results presented in this paper also clearly illustrate the flexibility of our generic proxy system. By simply installing a new plug-in, transcoding functionality was added to our proxies. By implementing and installing other novel plug-ins, their capabilities and range of application could easily be extended even further.
In the future we would like to improve our work in several ways. First and foremost, our intelligent proxy system at the moment still lacks device awareness. We envision the proxy gathering information regarding the capabilities of the receiving client device and combining this knowledge with its application and network awareness to take more complex transcoding decisions and to distribute the bandwidth of clients even more efficiently. Secondly, we would like to perform some more elaborate experiments, for instance by including mobile devices. Finally, we will keep looking at ways to increase the performance and scalability of our transcoding plug-in.

Acknowledgments

Part of this research was funded by the IBBT (Interdisciplinary institute for BroadBand Technology) and the Flemish Government. We also wish to thank ANDROME NV for the use of their video codec software.

References


