A SCALABLE QOS ADAPTATION SERVICE FOR MOBILE MULTIMEDIA APPLICATIONS

Cristian Hesselman Henk Eertink
Telematica Instituut
P.O. Box 589, 7500AN Enschede, The Netherlands
e-mail: {hesselman, eertink} @telin.nl

ABSTRACT

Mobile multimedia applications typically operate in an environment that consists of a variety of different types of mobile hosts and wireless networks with different capabilities and resource availability. This heterogeneity makes it difficult to scale mobile multimedia applications up to large numbers of participants. In this paper, we propose a system that uses a limited set of domain-specific quality levels per service category to ensure scalability. The system is based on the use of IP multicast groups and proxies. We present the dynamic aspects of our approach as well as an initial implementation. We use an application that distributes a TV channel to mobile users to explain our work.

1. INTRODUCTION

Multimedia multiparty applications that integrate fixed and wireless communications typically run on a wide variety of hosts. As a result, the components that make up these applications usually have to rely on a wide variety of computing and communications capabilities [7, 12, 13]. This is particularly true for application components that run on mobile hosts. Some of them for instance reside on high-end laptop computers with lots of processing power, high quality presentation capabilities and megabit network connectivity. Others reside on low-end PDAs with limited processing power, limited presentation capabilities and a low-speed network connection. This heterogeneous operating environment forms a problem for sessions in which a large number of distributed application components participate (e.g. a session that distributes a TV channel). In such cases, it usually becomes unfeasible to deliver a multimedia stream to each receiving application component at a Quality of Service (QoS) level that is fine-tuned to the capabilities and the current resource availability (e.g. in terms of network bandwidth) of the mobile host. An extreme solution to this problem is to deliver the same QoS to every application component in a session. This approach is feasible and even desirable for some classes of applications, but it typically results in a large number of users receiving a suboptimal QoS.

In this paper, we propose a middleware[1] platform that strikes a balance between the above two extremes. It narrows the 'QoS spectrum' that a heterogeneous environment creates down to a limited number of application-level service classes and admits only a few of such classes to a session. A service class in this context defines a (perceptual) QoS level of the raw multimedia stream that an application component receives. We believe that this approach allows the platform to scale sessions up to large numbers of application components in highly heterogeneous environments. The price that we pay is that there will typically exist application components that receive their multimedia streams at a QoS level that closely matches the capabilities and the current resource availability of the hosts that they use but do not exactly match them. Our platform realizes service classes mainly in terms of IP multicast groups and proxies. The IP multicast groups interconnect mobile hosts that have similar capabilities and similar resource availability; the proxies bridge the differences that exist between hosts that connect to different multicast groups.

We also present the dynamic aspects of our approach, in particular the dynamic activation and deactivation of service classes and the adaptation of QoS levels by

[1] We view 'middleware' as a collection of generic distributed services that are application-independent.

transferring application components between service classes.

The rest of this paper is organized as follows. In Section 2 we consider the architecture of our platform. In Section 3 we present the kind of sessions that the platform supports and discuss how the platform realizes them. In Section 4 we consider the dynamic aspects of our approach. In Section 5 we take a look at a system that implements a part of our approach. In Section 6 we describe related work and we present our conclusions in Section 7.

2. ARCHITECTURE

The architecture of our middleware platform is based on the QoS framework proposed in [21]. Figure 1 shows the part of the framework that is relevant for the material discussed in this paper.

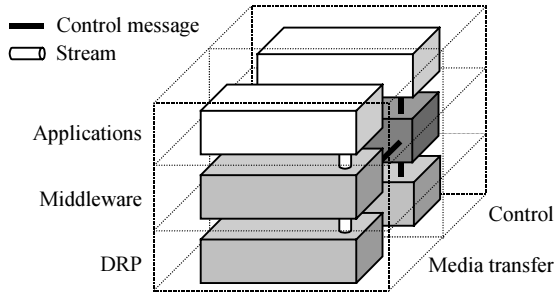

Figure 1. QoS framework.

QoS Framework

The framework of Figure 1 consists of an application layer, a middleware layer and a Distributed Resources Platform (DRP). The framework is furthermore made up of a media transfer plane and a QoS control plane.

The *media transfer plane* contains components that forward the data units of an audio-video stream. The components in the *application layer* encapsulate media presentation and capturing devices. These devices form the end-points of audio video streams. Examples are cameras, microphones and displays. The components in the media transfer plane of the *middleware layer* encapsulate media processing resources. They perform transport-independent as well as transport-dependent stream processing. Examples of the former are encoders, transcoders and mixers; an example of the latter is an RTP [16] packetizer that adapts an MPEG-4 [10] encoded stream for transmission over UDP. The components in the media transfer plane of the *DRP* encapsulate resources that provide end-to-end connectivity. Examples are IP (multicast) routers, network interface cards and bridges.

The components in the *control plane* of the framework encapsulate the logic that governs the QoS of the streams that flow through the media transfer plane. The control plane components for instance negotiate acceptable QoS levels, query and configure media transfer plane resources, and adapt QoS levels in response to host mobility.

The *interfaces* between the layers in the media transfer plane consist of *streams* with different characteristics. Streams of raw audio-video information for instance pass through the application-middleware interface, whereas encoded and packetized versions of those same streams pass through the middleware-DRP interface. The interfaces between the media transfer plane and the control plane take the form of *control messages*. The components in the media transfer plane define and publish these messages so that the components in the control plane can query and configure them. The middleware control components, for example, typically configure an MPEG encoder component in the media transfer plane through control messages that the encoder provides for this purpose. The interfaces between the different layers in the control plane also consist of messages. For example, the middleware's control components typically use messages to request a certain QoS from an RSVP-enabled DRP.

Architecture

The architecture of the system that we discuss in this paper is a narrowed-down version of the QoS framework of Figure 1. Specifically, we assume that the DRP's media transfer plane provides *UDP over IP multicast* capabilities. We furthermore assume that the control plane of the DRP allows us to query and configure the QoS characteristics of IP (typically through RSVP [20]). At the middleware level, we focus on a limited set of media processing resources, in particular on *encoders*, *decoders* and *transcoders*. We assume that *RTP* (de)packetizes encoded audio and video streams.

In this paper, we concentrate on the middleware layer. We first present the middleware's media transfer plane in Section 3. We then consider its control plane in Section 4.

3. MEDIA TRANSFER PLANE

The middleware's media transfer services allow two or more distributed application components to exchange streams of raw audio-video information by means of a *session*. Consider for example an application that distributes a TV channel to mobile clients. Figure 2 shows the application components

that are involved in the TV session.

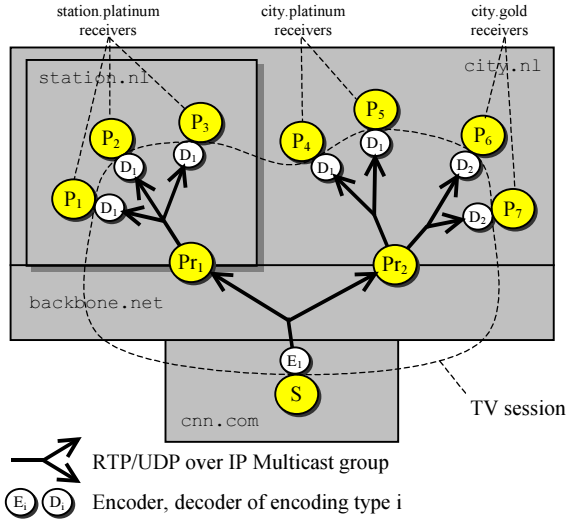

Figure 2. Multiparty TV session.

The broadcaster's server component (S) in Figure 2 produces a raw (i.e., uncoded) audio-video stream. The session delivers the stream to the player components P_1 through P_7 on mobile clients C_1 through C_7. Each player component P_i encapsulates the media presentation resources (display and speakers) on client C_i and consumes the stream.

The mobile clients are distributed over local area *domain* station.nl and over metropolitan area domain city.nl. They are equipped with media presentation resources (display and speakers), media processing resources (a decoder and an RTP depacketizer) and with at least one wireless network interface. The capabilities of the clients' presentation resources, media processing resources and network interfaces vary widely. The clients that use the networks of local area domain station.nl, for instance, typically use a network technology that has a greater capacity than the network technologies that are available in metropolitan area domain city.nl. The common capability of all clients is that they use RTP to depacketize an incoming encoded audio-video stream and that they are all IP multicast-enabled.

Service Classes

Our approach restricts the amount of 'QoS spectrum' available to the players in the TV session of Figure 2 to a few service classes. A *service class* defines a (perceptual) *QoS level* of the raw audio and video information that a player receives. The capabilities and the current resource availability of the client on which a player runs largely determine the service class that it will get. The *limited number* of service classes ensures that our system only has to deal with a few QoS levels. We believe that this allows our system to scale sessions up to large numbers of clients in highly heterogeneous environments. The downside of our approach is that there will typically exist players that receive their audio-video stream at a QoS level that is not fine-tuned to the capabilities and the current resource availability of the mobile clients on which they run. The QoS level that these players receive is therefore suboptimal. In short, our approach can be characterized as striking a balance between high scalability and delivering optimal per-client QoS levels. Observe that we define *application level* service classes. This is unlike DiffServ [20] that defines IP-level service classes.

In the TV session of Figure 2, the players receive their stream at one of two service classes: platinum or gold. Player P_3 could for instance be subject to a suboptimal QoS because it receives its stream at a platinum QoS level while the traffic situation in C_3's cell is such that C_3 would be able to consume more bandwidth and thus obtain a higher quality.

We propose to make the *QoS level* of the raw audio and video information associated with a service class *domain-specific*. The underlying reason is that domains have to realize service classes with different types of mobile clients and with different types of wireless networks. Domain station.nl can for instance realize its platinum class using high-speed wireless local area networks, whereas domain city.nl has to realize its platinum class over relatively low-speed metropolitan area wireless networks. As a result, the platinum class' QoS level will generally be higher in station.nl than in city.nl. The description of a service class therefore needs to include the domain that defines it. We accomplish this by prefixing a class' name with a domain name. In the example of Figure 2 we thus end up with service classes station.platinum, city.platinum and city.gold.

The consequence of using domain-specific service classes is that there generally exists a *well-defined ordering relation* between the service classes that a single domain supports. Such a relation does usually not exist between the service classes of different domains. It will for instance be hard to say that class station.platinum is 'better than' class city.platinum.

We furthermore suggest to also make the *number of service classes* domain-specific. Domain station.nl can for instance support three service classes (say silver, gold and platinum), while city.nl supports only two (say gold and platinum).

Realizing Service Classes

Our middleware realizes a service class in terms of the media processing and end-to-end connectivity resources that it has at its disposal. Specifically, the middleware defines the *QoS level* of a service class in terms of an audio and a video codec type (e.g. MPEG-4), a set of codec QoS characteristics, a packetizer type (RTP in this case) and a set of IP-level QoS characteristics. The QoS level of class station.platinum could for instance look like this:

```
station.platinum = {
  codecQoS = {
    videoCodec = {
      type = "mpeg4";
      chars = {
        // platinum characteristics
      }
    }
    audioCodec = {
      type = "mpeg4";
      chars = {
        // platinum characteristics
      }
    }
  }
  packetizer = "RTP";
  ipQoS = {
    chars = {
      // platinum characteristics
    }
  }
}
```

The codec and IP QoS characteristics of class platinum predominantly determine the QoS level of the raw audio and video stream that the players associated with this class receive. To realize the platinum QoS level, codecs must be configured to `codecQoS` and IP must be configured to `ipQoS`. The latter can be accomplished through a QoS aware IP-layer (e.g. an RSVP-enhanced one).

The codec and IP QoS characteristics of a service class can be described as a set of *parameter-value pairs* (see for instance [18, 2]). Examples of codec related QoS parameters include audio sampling size, audio sampling rate, video sampling rate, and so on. Typical IP-level QoS parameters are minimum and maximum bandwidth, jitter, etc.

The definition of a service class ensures that clients can communicate with each other without additional involvement of the middleware if the players that they host receive the same service class. Such clients can therefore be interconnected directly. Our middleware uses *site-local* [22] *multicast groups* for this purpose. In general, we can use one or more multicast groups to realize a service class. We can for instance use one multicast group to carry the audio portion of a stream and one to carry the video. Alternatively, we can use an RLM-like [3, 4] approach and realize a service class as a set of multicast groups with each multicast group carrying a layer of the audio-video stream. For the sake of simplicity, however, we will realize a service class as one multicast group in this paper. In the example of Figure 2 this for instance means that clients C_1 through C_3 are interconnected by a single multicast group because players P_1, P_2 and P_3 receive the same service class.

Clients that host players that receive different service classes cannot communicate with each other directly. This is because these clients use different codecs, or because they have configured the QoS of their codecs and IP service to be significantly different. Active involvement of the middleware is therefore required to bridge the differences between service classes (or, equivalently, between site-local multicast groups). Our middleware uses *proxies* [5] for this purpose. Proxies are middleware level components. They connect to a site-local multicast group for communications with mobile clients, and to a global multicast group to communicate with fixed clients. Proxies perform functions such as rate adaptation, transcoding [11], audio and video filtering, and so on. For example, proxy Pr_2 in Figure 2 transcodes between MPEG-2 [9] and MPEG-4 if the encoding of class station.gold is MPEG-4 (D_2 in Figure 2) and the encoding that the broadcaster server S uses is MPEG-2 (E_1 in Figure 2). Proxies typically run on *gateway* hosts in an *access domain* such as station.nl.

4. CONTROL PLANE

Each player component P_i of Figure 2 resides on mobile client C_i and is part of the application layer's media transfer plane. We associate each P_i with a *user agent* [21] component UA_i in the control plane of the application layer (cf. Figure 1). A user agent controls the *local* presentation QoS of the associated player. It uses the query and configuration control messages that the player provides for this purpose. A user agent for instance uses these messages to alter the volume of an audio-video stream that its player is presenting. The user agent furthermore makes use of the membership and QoS adaptation services that the middleware provides. The middleware makes these services available in the form of control messages.

Membership Control

The *membership* service allows a user agent to join a player to or remove it from a session.

As an example, assume that a new user agent UA_8 wants to *join* its player P_8 to the TV session of Figure 2 in domain station.nl. To accomplish this, UA_8 sends

a join *request* control message to the middleware's membership service. One of the parameters that this message takes is an application-level session identifier such as "CNN".

The middleware components that provide the membership service consume the request and examine the capabilities of mobile client C_8 on which AU_8 and P_8 reside. The middleware components then determine the capabilities and the current resource availability of the client's media processing components (decoder and RTP depacketizer) and its IP-level service. Based on this information, the middleware components determine a list of service classes that player P_8 can receive. The list will typically contain one entry. In some situations the list may however contain more than one entry. This for instance occurs when client C_8 is in range of both the station.nl domain and the city.nl domain. These domains both provide access to the TV session of Figure 2, but at different service classes. As a result, player P_8 will be able to join the TV session at class station.platinum or at class city.platinum. The list of possible service classes may also contain more than one entry if C_8 has alternative capabilities such as multiple decoders.

The middleware components inform UA_8 of the list of available service classes by sending a join *response* control message to it. For the sake of this example, we assume that C_8 can only receive class station.platinum and that the list therefore only contains one entry. User agent UA_8 next selects the class from the list and uses a *confirm* control message to inform the middleware of its selection. Service class station.platinum is already active, so it suffices for the middleware components to initialize C_8's decoder, RTP depacketizer and IP service to the QoS characteristics defined by station.platinum. The middleware can then simply subscribe client C_8 to the multicast group associated with class station.platinum.

In case the middleware components determine that C_8's capabilities are inadequate to receive class station.platinum, they must join P_8 to the TV session at another service class, say station.silver. This class is however not yet active. The middleware components must therefore *activate* it first. They do this by creating a site-local multicast group and by creating and starting a proxy. Next, the middleware components initialize C_8's decoder, RTP depacketizer and IP service to the QoS characteristics defined by station.silver. The middleware components complete the join by subscribing C_8 to the site-local multicast group and by subscribing the new proxy to the site-local multicast group and the global multicast group.

When a UA_8 wants player P_8 to *leave* the TV session, it sends a leave *request* control message to the membership service. The middleware components that implement the leave request consume the message and unsubscribe C_8 from the multicast group that it uses. The middleware components destroy the multicast group and the associated proxy if C_8 is the last member of the multicast group. This effectively *deactivates* the service class that P_8 received. The middleware *confirms* the leave by sending an appropriate control message to UA_8.

QoS Adaptation

The *QoS adaptation service* allows a user agent to be kept informed about QoS adaptations that occur as a result of host mobility and to request a different service class on behalf of the end-user.

In the example of Figure 2, the middleware adapts the QoS of an audio-video stream that a player receives by transferring the player from one service class to another. We call this a *service class handoff*.

The middleware accomplishes a service class handoff by having the client on which the player runs leave the multicast group associated with the player's current service class and joining it to the multicast group that represents the player's new service class. The middleware furthermore configures the client's decoder and IP services to the QoS level of the target service class. Observe that the handoffs that we use occur at the IP-level. This is unlike the network level handoffs that are required to transfer a mobile client from one base station to the next.

Service class handoffs usually occur as a result of host mobility. The middleware will for instance handoff P_3 from station.platinum to city.platinum if client C_3 roams from station.nl to city.nl. We call this an *inter-domain* service class handoff. There also exist *intra-domain* service class handoffs. In the example of Figure 2, this means that a player switches between different service classes while the client on which it runs roams within the same domain. Player P_3 may for example at some point need to switch from class platinum to class silver when C_3 roams within station.nl. The reason may be that C_3 roams from a lightly loaded cell to a more heavily loaded cell where there is not enough bandwidth to support class platinum. The middleware notifies a user agent of a service class handoff (inter- or intra-domain) by sending a *notification* control message to it.

Finally, a player may also need to be transferred to another service class *on request of the end-user*. This

for instance happens when the end-user who carries client C_3 decides to shrink-wrap the window that presents the TV channel because he wants to check his email. In this particular case, user agent UA_3 would send a handoff request message the middleware to hand player P_3 off to another service class. The middleware confirms the handoff by sending a *confirmation* control message to UA_3.

The middleware's control components must implement the membership and QoS adaptation services as one or more *protocols*. These protocols run between the mobile clients and the gateways that host proxies. They furthermore run between gateways.

5. IMPLEMENTATION

The goal of our implementation is to check if it is possible to handoff a player on a roaming client from one service class to another without serious hick ups in the QoS of the audio-video stream that it receives. In terms of the example of Figure 2, we can say that we want to validate if it is possible to handoff a client from one multicast group to another. We specifically want to check if such a handoff is possible in an inter-domain mobility scenario. We therefore implemented the scenario of Section 4 in which client C_3 roams between domains station.nl and city.nl (see Figure 2).

Figure 3 shows the organization of our test bed. It also illustrates how the proxy and player components of Figure 2 are distributed over the machines in the test bed.

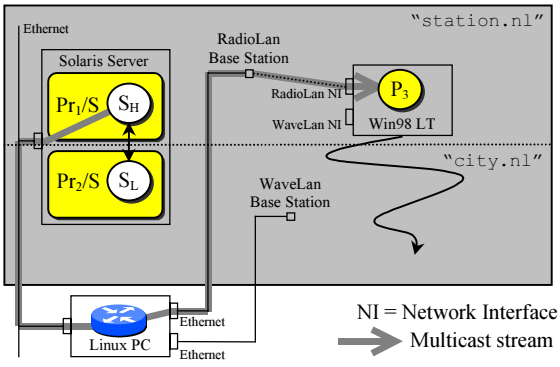

Figure 3. Test bed.

The Solaris machine hosts proxies Pr_1 and Pr_2. For reasons of simplicity, we have implemented proxies Pr_1 and Pr_2 to also act as broadcasting servers. That is, they generate the stream containing the TV channel locally rather than from a stream coming from the broadcasting server S (cf. Figure 2). Pr_1 and Pr_2 each consist of a QuickTime Darwin streaming server [23] for this purpose: Pr_1 consists of server S_H; proxy Pr_2 consists of server S_L.

S_H and S_L run synchronously as indicated by the arrow between them in Figure 3 and loop continuously. S_H locally reads a high quality movie from a hinted (i.e., encoded) QuickTime file and transmits it onto the multicast group that represents class station.platinum. Similarly, S_L locally reads a low quality version of the same movie from a different hinted file and transmits it onto the multicast group that represents class city.platinum.

The Solaris server connects to a Linux PC through an Ethernet network. The Linux PC acts as a multicast router. It routes the traffic that it receives on its Ethernet network interface to one of two base stations. One base station uses a pre-802.11 version of Lucent's WaveLan [27] technology. It provides a gross over-the-air bandwidth of 1 Mbps. The WaveLan base station operates at a frequency of 2.4 GHz and has an indoor range of approximately 30 meters. The second base station in the test bed is based on a proprietary technology of RadioLan [28]. This base station offers a gross bandwidth of 10 Mbps. It operates in the 5.8 GHz band and has an indoor range of approximately 15 meters. The base stations are positioned such that the WaveLan cell overlays the RadioLan cell.

The two networks that we use mimic the local and metropolitan area networks of the station.nl and city.nl domains of Figure 2. The RadioLan network represents station.nl's local area network (high capacity, short range) whereas the WaveLan network mimics city.nl's metropolitan area network (medium capacity, medium range).

The multicast group that S_H uses represents the multicast group of class station.platinum. We have configured the multicast router such that it transmits the data that this multicast group carries onto the RadioLan network. This means that the IP-level QoS of this class has a best-effort QoS with a gross bandwidth of 10 Mbps. Similarly, the multicast group that S_L uses represents the multicast group of class city.platinum. The multicast router transmits the traffic of this multicast group onto the WaveLan network. This also gives class city.platinum a best-effort IP-level QoS but with a gross bandwidth of 1 Mbps.

A Windows98 laptop (marked LT in Figure 3) represents client C_3 of Figure 2. The laptop is equipped with a RadioLan and a WaveLan network interface and runs the QuickTime client software package [26]. The QuickTime package covers the application components (in this case player P_3) and

the middleware layer components in the media transfer plane of Figure 1. We are currently building several middleware control components around the QuickTime client package. These components determine the network that provides the best IP-level QoS and join the laptop to the multicast group associated with it. As an example, assume that the location of the laptop is such that the RadioLan network provides the best IP-level QoS. In this case, the middleware control components initialize the necessary QuickTime components and join the multicast group that S_H uses. As a result, the client receives the steam that S_H transmits over its RadioLan network interface and the end-user sees the high quality version of the movie. When the laptop gets out of range of the RadioLan network, it must hand off to the WaveLan network. The middleware control components then unsubscribe the laptop from the multicast group that S_H uses, reinitialize the QuickTime components and join the laptop to the multicast group that S_L uses. As a result, the laptop now receives the low quality version of the movie that S_L transmits and the user sees a degradation in the QoS of the movie. The middleware components go through this behavior in reverse order when the laptop roams back into range of the RadioLan network.

6. RELATED WORK

Proxies are a common way of dealing with capability variations of networks and hosts. [1], [5], [11], [13], [14] and [15] are examples of initiatives that take this approach. In particular, [5], [13] and [15] use proxies to overcome capability variations for mobile networks and hosts.

[3] and [4] discuss an approach that uses multiple multicast groups to control the bandwidth of video streams that are destined for a large number of fixed clients in a heterogeneous best-effort Internet environment. This approach is known as Receiver-driven Layered Multicast (RLM). RLM exploits the fact that advanced coding schemes such as MPEG-2 and MPEG-4 allow information streams to be encoded at multiple quality levels. Each quality level produces a flow of a certain bandwidth each of which RLM transmits onto a separate multicast group. This allows receivers to 'tune in' to as many groups as they are capable of receiving given their current network capacity and 'add up' the quality levels they receive. We believe that RLM can be used to implement a service class and that it therefore fits into our architecture. Other areas that use multiple multicast groups to assure scalability are reliable multicast [17], multicast flow control [24], and distributed virtual environments [19]. The concept of handoffs between multicast groups also exists in this last application area.

We are also aware of work that combines proxies and multiple multicast groups, for instance [6], [7], [8] and [11]. [7] uses this combination in a mobile setting, but focuses on reliable communications rather than on multimedia streaming. [6], [8] and [11] also deal with the dynamic composition and configuration of proxies and multicast groups that is required in our approach, but in a fixed environment.

Mobile IP [25] supports host mobility in the Internet at the IP-level. Our platform, on the other hand, supports mobility at the middleware level.

7. CONCLUSIONS AND FUTURE WORK

We have discussed the architecture of our middleware platform. Our approach revolves around the notions of sessions, dynamic application-level service classes, service class handoffs, site-local multicast groups and proxies. We have claimed that this approach allows the platform to scale sessions up to large numbers of participants at the cost of delivering an audio-video stream at a close-to-optimal QoS level rather than at a level that is fine-tuned to the capabilities and current resource availability of a client. We have also discussed the control logic that is required to deal with the dynamic aspects of sessions. We exemplified our approach by means of an application that distributes a TV channel.

Our plans for the future are to develop the protocols that the middleware's control plane components must implement. In the short term, we will first focus on techniques to describe the capabilities and resource availability of mobile clients. In parallel to these developments, we plan to extend our test bed, for instance with transcoding proxies, rate adaptation proxies and other types of wireless networks.

ACKNOWLEDGEMENTS

The authors would like to thank Arjan Peddemors for reviewing the draft version of this paper.

REFERENCES

[1] S. Cho and Y. Shin, *"Multimedia Service Interworking over Heterogeneous Networking Environments"*, IEEE Network, March/April 1999

[2] M. Handley and V. Jacobson, *"SDP: Session Description Protocol"*, RFC 2327, April 1998

[3] S. McCanne, V. Jacobson and M. Vetterli, *"Receiver-driven Layered Multicast"*, ACM SIGCOMM, August 1996, Stanford, USA

[4] X. Li, M. Ammar and S. Paul, *"Video Multicast over the Internet"*, IEEE Network, March/April 1999

[5] E. Brewer et al., *"A Network Architecture for Heterogeneous Mobile Computing"*, IEEE Personal Communications Magazine, Oct. 1998

[6] E. Amir, S. McCanne, R. Katz, *"An Active Service Framework and its Application to Real-time Multimedia Transcoding"*, Proc. of ACM SIGCOMM'98, Vancouver, Canada, Sept. 1998

[7] Y. Chawathe, S. Fink, S. McCanne, E. Brewer, *"A Proxy Architecture for Reliable Multicast in Heterogeneous Environments"*, Proc. of ACM Multimedia'98, Bristol, UK, Sept. 1998

[8] K. Jonas, M. Kretschmer and J. Moedeker, *"Get a KISS — Communication Infrastructure for Streaming Services in a Heterogeneous Environment"*, Proc. of ACM Multimedia'98, Bristol, UK, Sept. 1998

[9] T. Sikora, *"MPEG-1 and MPEG-2 Digital Video Coding Standards"*, http://wwwam.hhi.de/mpeg-video/papers/sikora/mpeg1_2/mpeg1_2.htm

[10] R. Koenen, *"MPEG-4 — Multimedia for Our Time"*, IEEE Spectrum, Feb. 1999

[11] E. Amir, S. McCanne and H. Zhang, *"An Application Level Video Gateway"*, Proc. of ACM Multimedia, San Fransisco, USA, Nov. 1995

[12] A. Fasbender, F. Reichert, E. Geulen, J. Hjelm and T. Wierlemann, *"Any Network, Any Terminal, Anywhere"*, IEEE Personal Communications, April 1999

[13] A. Fox, S. Gribble, E. Brewer and E. Amir, *"Adapting to Network and Client Variability via On-Demand Dynamic Distillation"*, ASPLOS-VII, Oct 1996

[14] N. Yeadon, A. Mauthe, F. Garcia and D. Hutchison, *"QoS Filters: Addressing the Heterogeneity Gap"*, Proc. of the Interactive Distributed Multimedia Systems and Telecommunication Services (IDMS'96), Berlin, Germany, May 1996

[15] A. Balachandran, A. Campbell and M. Kounavis, *"Active Filters: Delivering Scaled Media to Mobile Devices"*, 7th International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV'97), May 1997, St. Louis, USA

[16] H. Schulzrinne, S. Casner, R. Frederick and V. Jacobson, *"RTP: A Transport Protocol for Real-Time Applications"*, RFC 1889, Jan. 1996

[17] S. Kasera, J. Kurose and D. Towsley, *"Scalable Reliable Multicast Using Multiple Multicast Groups"*, CMPSCI Technical Report TR 96-73, Oct. 1996

[18] *"CORBAtelecoms: Telecommunications Domain Specification"*, OMG, June 1998

[19] M. Macedonia, M. Zyda, D. Pratt, D. Brutzman and P. Barham, *"Exlpoiting Reality with Multicast Groups: A Network Architecture for Large-scale Virtual Environments"*, IEEE Computer Graphics and Applications, Sept. 1995

[20] X. Xiao and L. Ni, *"Internet QoS: A Big Picture"*, IEEE Network, March/April 1999

[21] C. Hesselman, I. Widya, A. van Halteren and L. Nieuwenhuis, *"Middleware Support for Media Streaming Establishment Driven by User-oriented QoS Requirements"*, accepted for publication at the 7th International Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services (IDMS2000), Oct. 2000, Enschede, the Netherlands

[22] D. Lee, D. Lough, S. Midkiff, N. Davis and P. Benchoff, *"The Next Generation of the Internet: Aspects of the Internet Protocol Version 6"*, IEEE Network, Jan/Feb 1998

[23] Apple Darwin Streaming Server, http://www.publicsource.apple.com/projects/streaming/

[24] S. Bhattacharyya, J. Kurose, D. Towsley and R. Nagarajan, *"Efficient Multicast Flow Control Using Multiple Multicast Groups"*, IEEE Infocom98, San Francisco, USA, April 1998

[25] J. Solomon, *"Mobile IP — The Internet Unplugged"*, Prentice Hall, 1998

[26] Apple QuickTime Client 4.1, http://developer.apple.com/quicktime/

[27] WaveLan homepage, http://www.wavelan.com

[28] RadioLan homepage, http://www.radiolan.com