# Lightweight Application Level Multicast Tunneling using mTunnel*

Peter Parnes, Kåre Synnes and Dick Schefström

*Department of Computer Science/Centre for Distance-spanning Technology, Luleå University of Technology, 971 87 Luleå, Sweden.*
{Peter.Parnes,Kare.Synnes,Dick.Schefstrom}@cdt.luth.se

**Abstract**

This paper presents a system, called mTunnel, for application level tunneling of IP-multicast traffic in a lightweight manner, where the end-user is responsible for deciding which MBone-sessions and which IP-multicast groups to tunnel. mTunnel is primarily designed for easy deployment and easy-to-manage tunneling. Information about currently tunneled sessions and control of mTunnel is provided through a Web-interface. To save bandwidth, tunneled streams can be transcoded on the media level and traffic sent through the tunnel can be compressed by grouping several packets together and using statistical compression. The overall bandwidth reduction achieved without modifying the media itself have been measured to be between 5 and 14% depending on the traffic and media type.

*Key words:* Internet, IP-multicast, tunneling, packet and header compression, rate control, automatic media transcoding, MBone, Java

## 1 Introduction

On the Internet today, more and more applications are starting to use *IP-multicast* [3] for distribution of primarily real-time data such as audio and video and a growing number of more conventional applications are starting to utilize the power of IP-multicast. In the rest of this paper IP-multicast will be referred to as multicast.

* This work was supported by the Centre for Distance-spanning Technology (CDT), Luleå, Sweden.

Multicast is special form of addressing and sending data on the Internet where the sender sends the data to a so called multicast group instead of sending the data directly to a particular host. Receivers that want to receive data from the sender joins the multicast group and the network routing protocols makes that traffic available to the new receiver. When the receiver no longer wants to receive the data its host will notify the network and the data will no longer be forwarded to that part of the Internet (with the exception when, there are other receivers on the same local network that want to continue to receive the data). Note that the sender does not need to keep track of who wants to receive the data, but this is done by the network itself.

Multicast has a number of advantages, such as its inherent scalability of data propagation where data is only copied in the network where needed. It also has built in robustness as it does not depend on any central resources such as reflectors or mirrors for data propagation. However, a major problem with multicast is that it is not part of the core IP version 4 family and by that currently only a smaller part of the Internet actually supports it.

To ease the development of multicast related protocols, a virtual network called the Multicast Backbone, MBone [9] was developed. It consists of tunnels between nodes that act as virtual routers, exchanging multicast packets encapsulated in unicast packets. Encapsulation means that multicast packets are received at a node that is connected to a multicast enabled network. Repackaged into a unicast packet and sent over the part of the Internet that does not support multicast to a node that is part of a multicast network. This second node removes the unicast information and resends the packet on its multicast network.

The MBone has existed for several years now and is slowly being deployed in production networks. A remaining problem is that unfortunately most dial-up links do not support multicast. This means that there is still a large need for tunneling of multicast over such links. Characteristics of these links are that they usually only connect a single host or a single local network with a few hosts and that these dial-up links most often only have a limited amount of bandwidth available.

Another reason for using tunneling of multicast is firewalls. At some companies, experiments have shown that it can be very hard to convince the system administrators to open the company's firewall for multicast traffic, but interestingly easy to convince them to open a few ports for TCP and UDP traffic to a specific host behind the firewall. There are a number of problems and questions related to tunneling of multicast, which are:

- How should the tunneled packets be encapsulated when sent through the tunnel to waste as little bandwidth as possible?

- How much control information is needed to be exchanged to maintain the tunnel operational?
- Can the tunneled data be transformed on the *IP level* (i.e. without any knowledge about the content) before being sent through the tunnel to lower the bandwidth needed for the tunnel?
- Can the tunneled data be transformed on *media level* (i.e. with knowledge about the content) before being sent through the tunnel to lower the bandwidth needed for the tunnel?
- Can the deployment of tunnels be made simpler and more lightweight than with earlier available software?

To target these problems and questions an application called the *multicast Tunnel, mTunnel* was designed and developed. This application is the focus of this paper.

### *1.1 Related work*

The primary and most popular application for tunneling multicast traffic on the MBone is the *MRouted* [5] application. This application acts as a virtual multicast router and includes a lot of routing functionality. The application has to have access to privileged information within the operating system it runs on and it also needs special routing support in the kernel. This means that the application can only be used on machines that contain the required routing functionality (such as hosts running variants of the UNIX operating system.) [1] Since, MRouted is a virtual router it implements and utilizes multicast routing protocols to control its tunnels. These protocols often exchange a lot of control and routing information which can be a problem on low-bandwidth links. Note that the mTunnel application presented in this paper is by no mean a replacement for the core MRouted tunnels in the MBone. Instead it should be seen as a complement to allow more users to use multicast applications.

liveGate [10] is another application for tunneling of multicast traffic, which was developed concurrently and independently of mTunnel. liveGate supports basic lightweight deployment of multicast tunnels and tunneling of multicast traffic, but does not currently support any traffic modification to lower the bandwidth.

A number of different tools and gateways for modification of real-time multicast traffic also exist, such as the *Robust Audio Tool - RAT* [7] which can convert audio data between different formats and the *RTPGW* [1] which focuses on that bandwidth is not uniformly available on the Internet for audio

[1] Microsoft Windows NT5 is supposed to support routing and contain the needed functionality to run MRouted.

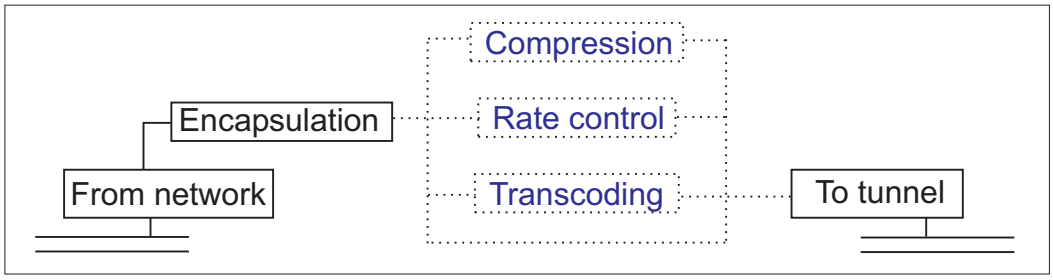


Fig. 1. Overview of the data path of an incoming packet through mTunnel. The three boxes in the center represent optional traffic modifiers.

and video transmissions. It can convert video between different formats and it also supports some basic audio transcoding.

The rest of the paper is structured as follows: Chapter 2 gives an overview of the mTunnel application; Chapter 3 discusses the use of non-lossy compression schemes for lowering the bandwidth; Chapter 4 describes rate control mechanisms for data flows; Chapter 5 examines extra delay introduced by the mTunnel application; Chapter 6 presents the implementation, status and evaluation of the problems and questions presented earlier and with Chapter 7 we conclude the paper and give a summary.

## 2 The mTunnel application

Tunnels always have two endpoints between between which traffic is tunneled. For that reason there must always be two instances of mTunnel running, one at each end of the tunnel. The basic functionality of mTunnel is very simple: listen to a number of multicast sessions, encapsulate all traffic received on these sessions and send the traffic through the tunnel. On the other end decapsulate the traffic and resend it locally using multicast. Fig. 1 gives a general overview of the data path of packet through the mTunnel application.

mTunnel runs as a user process at application level to allow easy and lightweight deployment. This means that the user does not need any special privileges to run the program. All the mTunnel application requires from the operating system is that it supports multicast. The main user interface to mTunnel is through a built in Web-server which allows users to get feedback about currently tunneled sessions and easily start tunneling of either publicly announced MBone sessions or private sessions where the user specifies the needed session information manually.

The tunneled data is sent over a UDP unicast “connection” between the two endpoints and all data is sent over the same port number. This allows for easy tunneling through firewalls (as the systems administrator only has to open one

port in the firewall). Control information is exchanged on a TCP connection between the two tunnel ends for reliable messaging.

Each multicast packet is encapsulated by adding a special trailer. The reason for using a trailer instead of a header (which is the case in most protocols) is twofold: it minimizes the number of required copy operations per tunneled packet (a header would impose that the incoming data would need to be copied to make room for a header) and that the IP/UDP/RTP [2] headers remain intact, which in turn allows lower level header-compression schemes to be utilized [8].

To lower the amount of encapsulation data sent over the tunnel, *flow identifiers* are used instead of sending the multicast group and port number with each tunneled packet. By using a flow identifier of size 1 byte, the trailer is reduced by 5 bytes which lowers the bandwidth with e.g. 1000 bps for a constant 40ms PCM audio flow (71 Kbps).

### *2.1 Data Translation*

The required bandwidth for a tunneled session may be lowered by transcoding the data. mTunnel includes a *translator*, that may translate the traffic in various ways. Currently the translator includes a *recoder*, that translates between different audio encodings; a *mixer*, that mixes several active streams into one single stream; a *switch*, which only forwards packets from a certain source based on another stream (for instance, video packets are only tunneled for the group member that currently is speaking - voice switching). The last part is the *scaler*, that rescales the traffic by dropping packets based either on just packet count or by taking media information into account. The latter makes a big difference on the result if a video encoding such as H.261 is used where one frame often spans more than one RTP packet. The different modes of the translator can also be pipelined, e.g. an audio session can be both mixed and recoded. All these different translation methods are lossy meaning that they actually remove information from the data stream. This transcoding allows users behind low bandwidth links to join sessions with a total higher bandwidth than the one available on the low bandwidth link.

[2] RTP, Real-Time Transfer Protocol [17] is the main protocol used on the MBone for transmitting real-time data.

The MRouted application (as presented in Sect. 1) connects multicast capable islands of networks to form the MBone. These network become an integrated part of the MBone virtual network and all traffic that is requested is tunneled to and from them. The decision of which sessions to tunnel, is based on requests made by the multicast aware applications. The messaging between hosts and multicast routers is done using a protocol called the *Internet Group Management Protocol - IGMP*[2]. As soon as a multicast aware application starts and requests data for a specific multicast group an IGMP message is sent to the nearest multicast router which in turn makes the corresponding multicast traffic available to that particular network.

This model works very good if the bandwidth is not limited, as several sessions can be tunneled at the same time. However, if the bandwidth is limited (like over analog modems or ISDN-links) the users have to quit their MBone applications to stop a session from being tunneled (i.e. if an application no longer "wants" multicast traffic for a special multicast group, usually the only way is to quit the application to stop the traffic from being tunneled). Also, if several users share the same narrow link, it might be complicated or even impossible to coordinate which sessions to tunnel (several users join different sessions at the same time).

mTunnel instead uses a *user decided session selection model* where users explicitly have to choose which sessions to tunnel. This has several advantages, such as making the end users aware of other currently tunneled sessions and removing the need for users to quit their multicast tools to stop the tunneling of specific multicast groups.

### *2.3 Transmission Loops and Time To Live*

mTunnel is designed to connect a network or a single host that is currently isolated from the MBone, to the MBone. However, if both ends of a tunnel are directly connected to the MBone, a transmission loop can occur if the tunneled MBone-sessions are not chosen carefully. mTunnel therefore does not forward packets through the tunnel if the sender matches the other end of the tunnel. Unfortunately, if two separate tunnels are deployed that together create a loop, packets will be forwarded over and over again.

If mTunnel suspects that a loop has occurred (the packet rate through the tunnel suddenly raises dramatically), it stops tunneling traffic, sends out a special probe-packet and waits for the probe-packet to be received again. If the probe-packet *is* received, all current tunneling remains stopped and users

of the system are notified through the Web-interface. If the probe-packet *is not* received, the process is repeated a number of times (with exponential back-off), as the probe-packet could have been lost on the way due to the best-effort nature of UDP-packets.

When packets are sent on the MBone, their reach is limited by a so called *Time To Live (TTL)* value. For instance, if a user wants to send multicast packets to the local network only, the packets are sent with a TTL of 1. In the current standard version of the sockets interface[3] under Unix and Windows, there is no way for a user application to get information about the TTL of an incoming packet. Due to the socket interface, and the fact that mTunnel runs as a user-application, mTunnel can only forward packets based on the TTL value specified when the session was created. Unfortunately, this means that if a user sends traffic in an announced MBone-session with a lower TTL than the announced TTL, the local packets will be "amplified" and retransmitted with a higher TTL than intended by the original sender.

## 3 Non-Lossy Compression of Tunneled Data

The amount of data sent through the tunnel can be lowered by compressing it using a non-lossy compression scheme. Normally, this is not done on single real-time flows as the encoding used to encapsulate the data usually already includes some kind of compression scheme. However, in the case where several different data flows are concentrated at a single point in the network (such as the tunnel ends), the redundancy between the different flows can be used for compressing the data. This section discusses how data compression have been utilized in the mTunnel application and how much the bandwidth is reduced for different kinds of data flows.

The more uncompressed data available, the more it can be compressed due to the higher probability of similar and redundant parts. This is generally not a problem when compressing data files as all the required data is already there, however, when compressing real-time data flows the goal is to keep the delay low (i.e. not to buffer the packets too long). Another goal is to keep each resulting compressed packet independent, so if it gets dropped in the tunnel, following packets may still be decompressed and do not depend on previous packets. To be able to compress the data flows, packets have to be buffered before being sent through the tunnel. Secondly, there must exist an efficient way of recovering from unsuccessful compression attempts, as a group of packets might actually generate more compressed data than they contained from the beginning.

[3] The way an application speaks with the operating system and the network.

To allow for compression, incoming packets are grouped together. This grouping can be seen as a first step of compression as each tunneled packet has a cost in number of bytes sent over the tunnel due to the IP/UDP header each packet include (i.e. by grouping several packets together only one set of headers is needed per sent grouped packet). The cost of the per packet header is a minimum of 28 bytes and by grouping $n$ (2 or more) packets together, $(n-1)*28$ bytes may be saved. This might not sound as much, but remember that most data flows consist of very small packets, e.g. a 40 ms PCM audio flow consists of packets containing 320 bytes of data plus headers. This first step of the compression is referred to as *header suppression*. If the available grouped data cannot be compressed (i.e. compression generates more data than the original grouped packets), then it is sent grouped as one large uncompressed packet. The algorithm used is:

```
for each packet do
  if not packer.willFit(packet)
    packer.generateAndSend()
  packer.add(packet)

generateAndSend:
  if compressed_size < orginal_size
    sendGroupedAndCompressed()
  else
    sendGroupedAndUncompressed()
```

### *3.1 Measurements and Compression Evaluation*

Five different sessions (including several different data flows) have been examined with mTunnel to see how good they can be compressed. The measurements were done by saving information about each compressed packet within the mTunnel application and the measurements were done over a time period of 1 hour. In none of the tests lossy transcoding was performed and to be able to repeat the exact traffic, sessions were recorded earlier and played back using the multicast Media-On-Demand system - mMOD [13]. Table 1 presents how much bandwidth can be saved by using tunnel compression on the different data flows. The table shows that for normal data flows a compression ratio of about 5-9% is achieved. The reason for the fourth session getting a higher compression ratio (14%) than the rest is that it includes a whiteboard stream which consists of many very small packets which in turn contain a lot of redundant information. In session 5, hardly any compression is achieved due to the fact that the data flow consists almost entirely of packets with a constant size (1024 bytes + headers) containing already heavily compressed

Table 1
Overview over compression results for different sessions using non-lossy compression.

| Session | Original Bandwidth Kbps | Total saved % Kbps | Saved due to compression | Saved due to header suppression |
|---|---|---|---|---|
| **Local lecture** audio, video, HTML using mWeb [14] | 198 | 6.11% 12.9 | 3.05% | 3.06% |
| **Electronic corridor** audio, video | 170 | 9.23% 17.3 | 4.60% | 4.62% |
| **Meeting** audio, video, mDesk [15] | 229 | 5.15% 12.5 | 2.08% | 3.08% |
| **Global lecture** audio, video, WB | 192 | 14.3% 31.9 | 10.9% | 3.32% |
| **Constant MPEG** using mIR[11] | 131 | 0.128% 0.164 | 0.0363% | 0.0920% |

MPEG coded audio data [11].

## 4 Priority and Rate Control of Data Flows

As mTunnel is designed to run over links with narrow bandwidth it is important to support rate control for the total tunnel and for each tunneled session. This can be used for e.g. prioritizing audio over video by reducing the amount of bandwidth available for video. By reducing the total available bandwidth for the tunnel, other types of traffic might be given more room on the bandwidth restricted link. A higher priority means that if congestion occurs, the traffic belonging to the session with higher priority will be forwarded and traffic for a session with a lower priority might get dropped.

Each tunneled session can have its own forwarding priority and by default, all sessions have the same priority. Using the Web-interface, the priorities can be changed individually. Priorities for one or several sessions can also temporarily be locked, meaning that no other session can be given a higher priority than that session as long as it is locked (note that locking does not mean that the session gets a higher priority, only that its priority can not be exceeded by another future session). This is useful if an important electronic meeting is conducted over the tunnel and the participants do not want to be disturbed by another user who wants to watch some other MBone-session.

Priorities, can also be configured in mTunnel, based on a number of different

Table 2
Mean delay ($RTT/2$) of traffic for the different measured cases (ms)

| Network type | Direct contact | Tunneling without compression | Tunneling with compression |
|---|---|---|---|
| Ethernet | 2.00 | 14.4 | 74.5 |
| ISDN | 52.0 | 74.1 | 138 |

variables in the session: the media type, the multicast address and port, the used bandwidth, the name and the description. This allows for advanced selection of priority schemes, that enables a user to participate in sessions even if the total needed bandwidth is not available.

To control the data flows, a so called *bound token bucket* [16] scheme is used. The main idea of this scheme is to have a bucket of tokens that is filled up regularly depending on the available bandwidth for that flow. Each token corresponds to one byte of available bandwidth and when a packet is forwarded the number of tokens are reduced by the corresponding size of that packet. The bucket has a limited size, meaning that there can never be more than a certain amount of tokens available. An infinite number of tokens would make it possible for a bursty data flow to overrun the available bandwidth. This might still happen in the bounded bucket case, but the impact is much less severe if the max size is relatively low. (A hard limit against overruns can achieved by adding a *leaky bucket* [16]).

## 5 Data delay

The use of tunneling with its different schemes for reducing the bandwidth and the bursts in bursty flows has the side effect of introducing extra jitter and delay in the tunneled traffic. The introduced delay was measured on the global lecture (row 4 in Table 1). The measurements were done by measuring the round-trip-time ($RTT$) of a data flow between two hosts on different networks, one *sender* and one *echo-host* which bounces the received traffic back to the sender. Tests were conducted over two types of networks, Ethernet and ISDN.

For each type of network three tests were conducted: one where the two hosts had direct connectivity through a router (using only unicast in the ISDN case), a second one where the traffic was tunneled between the two networks using mTunnel with no compression or transcoding and a third one using mTunnel with compression turned on. For each of the tests the round-trip-time ($RTT$) was measured. Table 2 shows the mean delay ($RTT/2$) for the six cases. As seen in the table, the delay gets quite high when using mTunnel with ISDN and compression turned on. This high delay could make synchronous conversations

uncomfortable as the $RTT$ would get up to around 276 ms which is above the usually recommended 150-250 ms maximum for conversations.

## 6 Implementation, Status and Evaluation

The current prototype is implemented in the platform independent Java language (version 1.1), except some parts of the audio transcoder that are implemented in C for efficiency reasons. The non-lossy data compression is done using the built in version of the ZLIB [4] compression library in the Sun Java environment. The audio recode functionality is currently only available on Sun/Solaris, but the rest has been tested to work under both Unix and Windows95/NT4. All tests and measurements mentioned in this paper were done under Sun/Solaris.

### *6.1 Further issues*

There are a number of open security issues not yet addressed such as encryption of the tunneled data and IP-spoofing. [6] proposes a solution for the IP-spoofing problem which should be further examined. The user interface currently does not include any user authentication, meaning that any user can configure mTunnel and its tunneled sessions. The translator should also be extended to support a larger variety of encodings. Another issue currently not addressed, is the possibility of sharing media translators between two or more concurrently running instances of mTunnel to save CPU usage by only translating the same stream at one host. Other types of non-lossy compression schemes should be investigated as well.

A limitation in the current implementation is that a TCP connection is used for exchanging control information and a UDP "connection" for the actual tunneled data. If these two could be combined deployment together with firewalls would get easier as only an UDP port would have to be opened for mTunnel and not both a TCP and a UDP port as it is the case today.

### *6.2 Evaluation*

In the introduction a number of questions and problems were introduced. This section presents how mTunnel target these.

- How should the tunneled packets be encapsulated when sent through the tunnel to waste as little bandwidth as possible?

The encapsulation of the packets in mTunnel is done by only including minimal extra information such as multicast group and port. To lower the overhead flow-identifiers and packet grouping is used.

- How much control information is needed to be exchanged to maintain the tunnel operational?

  mTunnel tries to minimize the amount of control information sent through the tunnel and no periodic update messages are sent.

- Can the tunneled data be transformed on the IP level (without any knowledge about the content) before being sent through the tunnel to lower the bandwidth needed for the tunnel?

  mTunnel can compress traffic using statistical compression without knowledge about the type of traffic currently being tunneled.

- Can the tunneled data be transformed on media level (with knowledge about the content) before being sent through the tunnel to lower the bandwidth needed for the tunnel?

  mTunnel allows traffic to be translated in a number of different ways such as: media can be recoded to another encoding; several active streams can be mixed; stream can switched based on activity in another stream and the streams can be rescaled by dropping parts of the traffic based on the type media in the stream.

- Can the deployment of tunnels be made simpler and more lightweight than with earlier available software?

  mTunnel only require that the host operating system supports multicast and can therefore be run on almost any operating system. The system includes a Web-interface which allows users to easily configure mTunnel and select which sessions to tunnel. mTunnel is implemented in Java which means that it runs on any platform that supports Java.

## 7 Summary and Conclusions

This paper presents a system for allowing users to easily connect to the MBone infrastructure and to connect different isolated multicast capable networks. mTunnel gives users easy access to information about currently tunneled sessions through a Web-interface, which also allows for easy configuration of existing and future tunneled sessions. mTunnel does not start tunneling of MBone-sessions based on current multicast group activity and IGMP messages, but instead makes the user responsible for deciding which MBone-sessions are to be tunneled. This allows for a *user decided session selection model* where tunneling decisions are left explicitly to the user.

To save bandwidth, data streams can be transcoded in four different ways: audio can be recoded to an encoding that requires lower bandwidth, several simultaneous audio streams can be mixed into a single stream, streams can be switched based on another stream, and streams can be scaled by dropping certain parts of the traffic. The traffic can also be compressed on the IP level which saves about 5-14% bandwidth. Unfortunately, measurements show that the compression adds a high extra delay to the tunneled traffic which might become a problem on links with an already high delay.

mTunnel does not require any special features in the operating system other than general multicast support and it runs as a normal user application. This means that it is very easy to deploy. mTunnel also tries to minimize the amount of control traffic exchanged through the tunnel and to minimize the cost of encapsulation of data by using for instance flow identifiers instead of adding group and port information to each tunneled packet. Note that the target of mTunnel is by no mean to replace existing software for core tunneling in the MBone, such as MRouted, but should instead be seen as a tool for connecting more users to the global multicast network.

mTunnel is currently being used in three different ways: to connect different parts of a large software company's intranet, to connect computers at users homes, and to connect industry networks to the MBone. More information about the current version, status of mTunnel, earlier publications and the program itself can be found at [12]. The usage of mTunnel has shown and proven that it is useful and that there is a need for this kind of applications.

## Acknowledgement

Thanks to Mattias Mattsson, Fredrik Johansson, Serge Lachapelle, Håkan Lennestål, Johnny Widén and Ulrika Wiss, all at Centre for Distance-Spanning Technology, and R. P. C. Rodgers, U.S. National Library of Medicine, for interesting comments, encouragement and feedback.

This work was done partly within the Esprit project 20598 MATES, which in turn is supported by the Information technology part of the 4th Framework Program of the European Union. Support was also provided by the Centre for Distance-spanning Technology (CDT).

## References

[1] E. Amir, S. McCanne, and H. Zhang. An application level video gateway. In

*Proceedings of ACM Multimedia'95*, 1995.

[2] S. E. Deering. Internet Group Management Protocol - IGMP. IETF RFC1112.

[3] S. E. Deering. *Multicast Routing in a Datagram Internetwork*. PhD thesis, Stanford University, 1991.

[4] P. Deutsch. ZLIB compressed data format specification version 3.3, 1996. IETF RFC1950.

[5] B. Fenner. The multicast router daemon - MRouted. [4].

[6] R. Finlayson. The UDP Multicast Tunneling Protocol, 1997. Work in progress, draft-finlayson-utmp-01.

[7] V. Hardman, A. Sasse, M. Handley, and A. Watson. Reliable audio for use over the internet. In *Proceedings of ISOC INET'95*, 1995.

[8] V. Jacobsson and S. Casner. Compressing IP/UDP/RTP Headers for Low-Speed Serial Links, 1997. Work in progress, draft-ietf-avt-crtp-04.

[9] V. Kumar. The MBone information web. [5].

[10] Live Networks, Inc. The liveGate multicast tunneling server. [6].

[11] P. Parnes. The multicast Interactive Radio - mIR. [7].

[12] P. Parnes. The multicast Tunnel system - mTunnel. [8].

[13] P. Parnes. mMOD - the multicast Media-On-Demand system. extended abstract [9], 1997.

[14] P. Parnes, M. Mattsson, K. Synnes, and D. Schefström. The mWeb presentation framework. *Computer Networks and ISDN Systems*, September 1997.

[15] P. Parnes, M. Mattsson, K. Synnes, and D. Schefström. The WebDesk framework. In *Proceedings of The Seventh Annual Conference of the Internet Society (INET'97)*, 1997.

[16] C. Patridge. *Gigabit Networking*, pages 260–262. Addison Wesley, 1994.

[17] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A transport protocol for real-time applications, 1996. IETF RFC1889.

[4] <URL:ftp://ftp.parc.xerox.com/pub/net-research/ipmulti/>
[5] <URL:http://www.mbone.com/>
[6] <URL:http://www.lvn.com/liveGate/>
[7] <URL:http://www.cdt.luth.se/~peppar/progs/mIR/>
[8] <URL:http://www.cdt.luth.se/~peppar/progs/mTunnel/>
[9] <URL:http://www.cdt.luth.se/~peppar/progs/mMOD/>