Proposed Mbone/Xunet Extensions
Revision 1.7.3

John Lockwood
Xunet Project
AT&T Bell Labs
Murray Hill, NJ

August 27, 1993

Abstract

This document outlines the plans to support for IP-based multicast applications on Xunet. This extension would allow multicast–based multimedia programs, including `nevot` (network voice terminal), `vat` (visual audio tool), `nv` (network video), and `wb` (whiteboard), to run unmodified over Xunet.

These plans involve extending the Mbone across Xunet. By surrounding Xunet with resonable thresholds, it is possible to confine high-bandwidth traffic within Xunet.

With an expected Mbone load of 0.5 Mbps, the multicast traffic will occupy only a small portion of Xunet's available bandwidth and network resources. In addition to allowing audio and video conferencing between Xunet sites, usage measurements could provide interesting statistics on multicast traffic and traffic patterns.

By setting the metrics on the links near unity, many Xunet links will become primary links for multicast traffic originating near Xunet routers. Larger metrics will cause the shortest path algorithm to use the exisiting Internet links to carry non-Xunet traffic. Because of the nature of the truncated broadcast algorithm, there would be no additional traffic due to the lower metrics. Initial tunnels and metrics are suggested in this paper. A tool is provided to update metrics across the sites as needed.

Please feel free to send comments and suggestions to: `lockwood@research.att.com`

Thanks are due to Chuck Kalmanek, Srinivasan Keshav, Steve McCanne,
Pat Parseghian, and Henning Schulzrinne for reviewing this document and offering
numerous suggestions and corrections

An updated postscript version of this document is available via
anonymous ftp from `ipoint.vlsi.uiuc.edu` as `/pub/ipoint/Documents/xbone.ps`

Contents

1 Xunet

Xunet is an experimental Asynchronous Transfer Mode (ATM) network that connects AT&T Bells Labs, University of Illinois, University of California at Berkeley, University of Wisconsin, Rutgers University, Lawrence Livermore National Laboratories and Sandia National Laboratory using a combination of DS3 and optical links, as shown in Figure 1.1 [1]. Routers, consisting of a Silicon Graphics 4D/310S with a custom Hobbit processor-based ATM interface, provide IP connectivity between the Xunet routers [2].

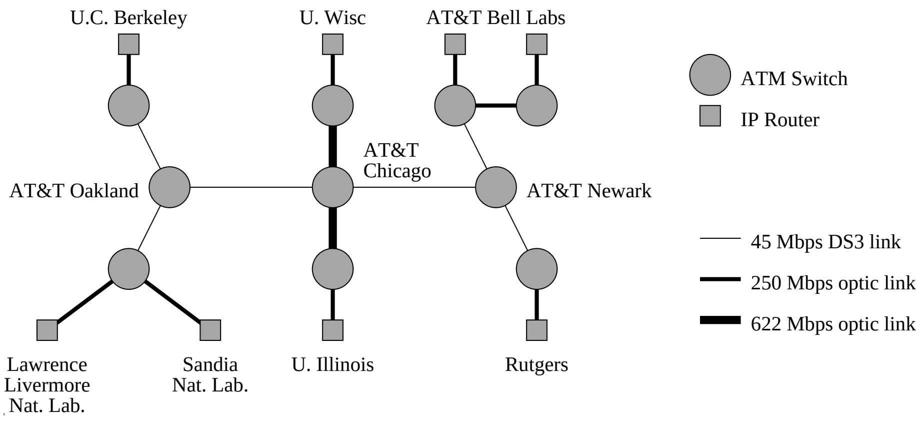

Figure 1.1: Xunet (Aug. 1993)

2 The Mbone

The Multicast Backbone (Mbone) is a virtual network that runs on top of the Internet. The Mbone is implemented by multicast routers that are scattered throughout the Internet.

Using the Mbone, it has become possible to provide audio and low-bandwidth video conferencing on almost all workstation platforms [3].

Most Mbone routers are actually workstations running a program called `mrouted`. When multicast routers are separated by unicast routers, the multicast packets are encapsulated in unicast IP packets. These "tunnels" allow the multicast packets to be routed between "multicast regions" over a standard IP network.

A postscript map of the Mbone topology can be obtained via anonymous ftp from `parcftp.xerox.com` in the `pub/net-research/` directory as `mbone-map-small.ps`.

2.1 Multicast IP Datagrams

The structure of the IP datagram remains unchanged for Multicast. Multicast IP packets are identified by a class D IP addresses, which have the first nibble set to `1110`. Thus, multicast IP packets have addresses ranging from `224.0.0.0` to `239.255.255.255`. Unlike unicast IP addresses, multicast addresses are typically dynamic—hosts may join and leave multicast groups. A few multicast addresses are reserved:

- `224.0.0.1`: Membership Query: From multicast router to all hosts
- `224.0.0.4`: Membership Report: From host to multicast router, Neighbor probe
- `224.0.2.2`: Neighbor probe: From multicast router

2.2 Example

An example Mbone topology is illustrated in Figure 2.1. The darkened nodes represent a multicast router (implemented as a unicast router in parallel with a workstation running `mrouted`, while the white nodes represent unicast IP routers. The tunnels (indicated by dotted lines) channel the multicast IP packets between multicast routers using the intermediate unicast routers. The numbers represent metrics and thresholds, which are described in the following two sections. A multicast region (indicated by the shaded box) is a portion of the Mbone surrounded by high thresholds that can confine high-bandwidth multicast traffic. This document describes how Xunet will become a multicast region of the Mbone.

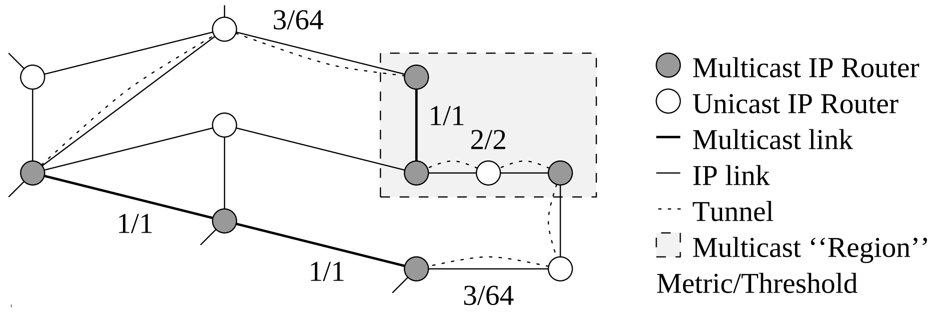

Figure 2.1: An example Mbone topology

2.3 Metrics

Each link has a static value assigned as a metric. The routing algorithm uses the metric to determine the shortest path between multicast routers. To reduce the time required to calculate the shortest path distances, the current routing algorithm has the limitation that the sum of the metrics between any two points must not exceed `31`. Most primary Mbone metrics are assigned a value of one. Backup links typically have a metric of two or three. [4].

2.4 Thresholds

Each link is assigned a static threshold value. If the packet's time-to-live is less than the threshold, the packet is not forwarded. Thesholds are used to confine traffic within a local region. By placing large thresholds on the links surrounding a multicast region, traffic can be contained. Typical threshold assignments are listed below: [4].

1	Local machines
32	Separate sites
64	Separate regions
128	Separate continents

2.5 Initial time-to-live (ttl)

High bandwidth traffic is assigned an initial ttl value that would prevent it from crossing boundaries. Applications choose their initial ttl by a *setsockopt* just after opening the socket. Each packet's ttl field is decremented by one for each hop. When the metric reaches zero, the packet is quietly discarded. Typical initial ttl settings are listed below [4]:

31	High bandwidth local traffic (> 200 kbps)
63	Video/Voice conference
127	64kbps audio
255	Low bandwidth voice/`wb` traffic

2.6 Your Responsibilites: A warning from Van

Users of the Mbone must control themselves. Due to the nature of the truncated broadcast algorithm, multicast traffic with high initial ttl's will propagate beyond Xunet and reach the entire world. The following note was posted to the Mbone mailing list from Van Jacobson to an Mbone user who accidentically ran `nv` with a high metric and high bandwidth:

> Why are you sending 350-450Kb/s video at multicast TTL of 127? Even if the subject matter had been interesting (it's a totally dark room), this is a completely unreasonable bandwidth for such a large ttl – You are totally saturating most of the international links and have completely destroyed the audiocast of the Summer School on Telecomm being broadcast from Lappeenranta University in Finland. Some of us are coming in in the middle of the night to watch this & weren't terribly pleased to learn that you had knocked it off the air. Last week you did essentially the same thing and wiped out the SigKids feed from SigGraph. This is very antisocial behavior. - Van Jacobson

3 Multicast Host Extensions

Kernel modifications are necessary to allow a host to support IP multicast. These changes have been programmed for the Sun and have been ported to SGI and other platforms. A multicast host is only concerned with transmitting and receiving the datagrams—it does not maintain the routing tables nor state information for the multicast network. Multicast routing is performed by multicast routers, as will be described in later sections.

An example configuration is illustrated in Figure 3.1. Here, the multicast router periodically sends a `query` IGMP message on the Ethernet. Each of the hosts responds with a `report` IGMP message, minimizing the traffic by the algorithm described in Section 3.3 [5].

3.1 Sending datagrams

A multicast packet is transmitted in much the same way as that of a unicast IP packet. New *setsockopt*'s have been provided to allow the application to specify the initial ttl as well as the outgoing network interface. If the host has more than one interface, and if the outgoing interface is unspecified, a default interface is selected. If the ttl is left unspecified, it defaults to one, forcing the packet to be transmitted only on the local network.

To transmit the multicast packet on an Ethernet, the low-order 28 bits of the destination address are mapped on the low-order 23 bits of the Ethernet broadcast address. The multicast Ethernet packet has a header of `01-00-5E-XX-XX-XX`.

3.2 Receiving datagrams

To receive multicast datagrams, the application must first join the multicast group using the `IP_ADD_MEMBERSHIP` and `IP_DROP_MEMBERSHIP` *setsockopt*'s to add and delete entries in the kernel's multicast group list. This downcall may, in turn, trigger a downcall to the physical network device, allowing the hardware to assist in packet filtering.

Because Ethernet does not have enough bits to uniquely identify each of the 2^{28} multicast groups, and because the Ethernet hardware may support only a small, fixed number of group addresses, some packet filtering must occur in software.

3.3 IGMP Host-specific Messages

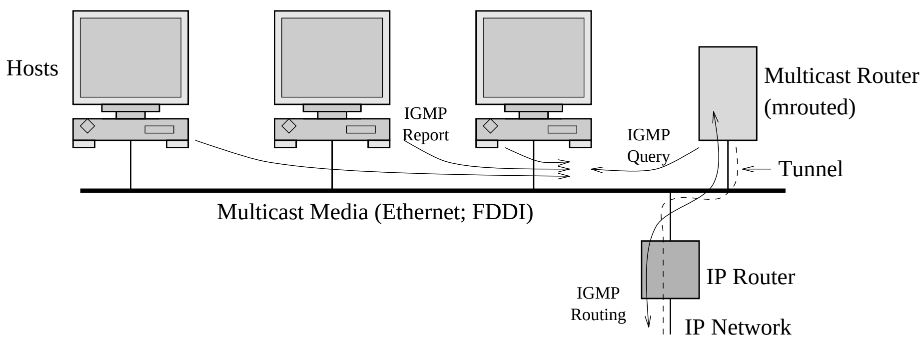

Figure 3.1: Hosts attached to the Mbone

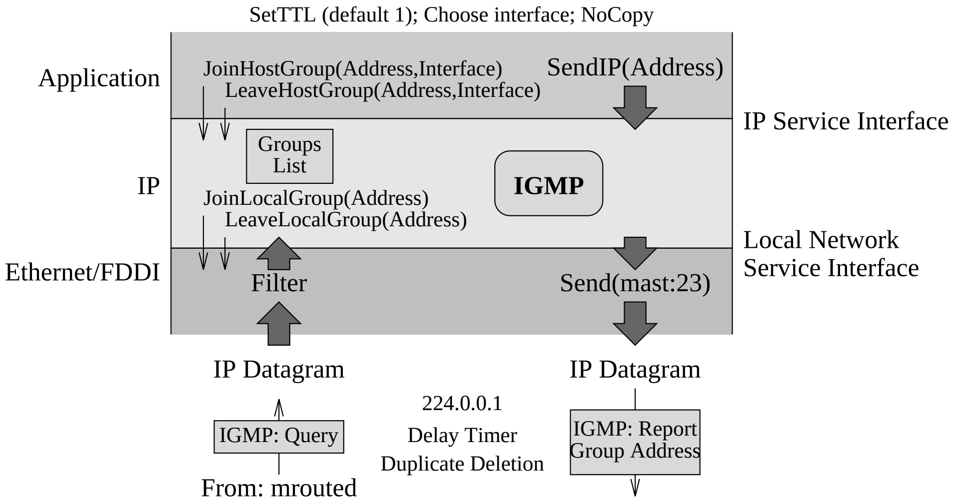

Figure 3.2: Multicast Host Extensions

Internet Group Management Protocol (IGMP) messages are exchanged between hosts and multicast routers. These messages are transmitted on the reserved "all-hosts" group address of `224.0.0.1`. There are two types of IGMP messages relevant to hosts:

- Host Membership Query: Router asks which multicast addresses are in use
- Host Membership Report: Hosts respond with addresses

The `query` message is periodically broadcast from the multicast router to all hosts on the local network. Each host responds by sending a `report` message that contains exactly one of the group addresses in use by that host. To avoid generating a rapid burst of traffic for each `query` message, timers are used to randomly delay the time before the `report` message is sent. Additionally, if a host sees that another host on the local network has already sent a `report` message, it will not send its `report` message for that group. The kernel-level host extensions are illustrated in Figure 3.2.

4 Multicast Routing: `mrouted`

For the Mbone, multicast routing takes place in a program called `mrouted`. This program listens to and generates IGMP messages to communicate with other `mrouted`'s and multicast hosts. This is a user-level program that uses `setsockopt` downcalls to set the routing tables, which are maintained in the kernel. Mbone routing is based on the Dalal and Metcalfe *reverse-path broadcast* Distance Vector Multicast Routing Protocol (DVMRP) algorithm [6].

Multicast packets are routed by their source address and the interface from which the packet arrived. Presently, this program uses broadcast trees to distribute multicast traffic throughout the Mbone. Leaves that are not participating in a multicast session are truncated [7], which helps to reduce the traffic seen by the attached hosts. Thresholds and time-to-live values also help reduce excessive multicast traffic.

To give an example of some of the state information maintained in each router, the text displayed in Sections 4.1 and 4.2 was extracted from `/usr/tmp/mrouted.dump` on *peters.research.att.com* (July 2, 1993), just after issuing a `killall -USR1 mrouted` [8].

4.1 Virtual Interface Table

`mrouted` must maintain information about each of its multicast interfaces. A virtual interface can be either a physical multicast interface or a tunnel to another `mrouted`. When `mrouted` is started, it first adds all physical interfaces (like Ethernet or FDDI) to the virtual interface (vif) table. A physical interface can be disabled, if desired, by adding a `disable` option in the `mrouted.conf` file.

```
Vif  Local-Address                           Metric  Thresh  Flags
 0   192.20.225.14    subnet: 192.20.225        1       1    querier
                      groups: 224.0.0.4
                              224.0.2.2
 1   192.128.161.11   subnet: 192.128.161       1       1    querier
                      groups: 224.0.2.2
                              224.0.0.4
 2   192.20.225.14    tunnel: 192.12.211.10     1      32
                      peers : 192.12.211.10
```

In this example, the first virtual interface (vif 0) refers to the Ethernet (subnet `192.20.225`), while the next (vif 1) refers to the FDDI ring (subnet `192.128.161`). The final virtual interface (vif 2) refers to the tunnel that connects peters to the Internet Mbone through *jvnc-net*. The `querier` flag indicates that this interface is a leaf, causing IGMP messages to be transmitted and received to determine which multicast groups are active.

4.2 Multicast Routing Table

The multicast routing table is the information maintained in the kernel for routing incoming multicast packets. The incoming vifs correspond with the vifs listed above.

Origin-Subnet	From-Gateway	Metric	In-Vif	Out-Vifs
128.109	192.12.211.10	3	2	0* 1*
128.231.128	192.12.211.10	5	2	0* 1*
140.181.64	192.12.211.10	14	2	0* 1*
192.16.184	192.12.211.10	11	2	0* 1*
192.20.225		1	0	1* 2
192.128.161		1	1	0* 2
192.231.8.128	192.12.211.10	8	2	0* 1*
...				

From this table, we can see that any multicast packet arriving with a source address of `128.109.X.X` from the *jvnc-net* multicast gateway via the tunnel (vif 2) would be forwarded to both the Ethernet (vif 0) and the FDDI (vif 1). Any packet arriving from the local Ethernet will be sent through the tunnel (if the ttl is above the tunnel threshold) and will be forwarded to the FDDI (if there are any listeners). The `*` indicates that these vifs are leaves of the multicast tree, and traffic will only be forwarded to that vif if there are hosts that have joined the multicast address of the packet in question.

4.3 IGMP Router-specific Messages

In addition to the `Query` and `Report` IGMP messages used between the router and attached hosts, there are a few router-to-router IGMP messages. The `IGMP_DVMRP` messages are as follows:

`PROBE`	Neighbor Probe
	Request information
`REPORT`	Route Report
	Returns a list of {origin, metric}
`ASK_NEIGHBORS`	Neighbor Request
	Request neighbor information
`NEIGHBORS`	Neighbor List
	Returns a list of neighbor routers
`ASK_NEIGHBORS2`	Neighbor Request
`NEIGHBORS2`	Neighbor List

5 Multicast Applications

This section examines existing multicast applications and gives a programming example of the multicast–specific network application code.

5.1 Existing Multicast Applications

The following programs use the multicast extensions to send multicast data over the Mbone. All software is available via anonymous ftp on various sites. Source code is available as noted.

5.1.1 Session Directory: `sd`

Session Directory is like a "TV Guide". It is the mechanism used to coordinate multicast audio/video/whiteboard conferences [9]. `sd` is used on a voluntary basis. When someone starts a conference, they often post the multicast address and the port number to `sd` so that others can participate in the conference. `sd` is often used to launch the other multicast applications described in this section, alleviating the user of manually entering the address and port values. The binaries for `sd` are available at `ftp.ee.lbl.gov`.

5.1.2 Visual Audio Tool: `vat`

`vat` provides audio conferencing on a network [10]. Time stamps and sequence numbers are used to reassemble audio packets into a sequential stream of data suitable for an audio device. `vat` has many features, including automatic silence suppression (transmitting audio only when there is a signal), a mechanism to display all listeners as well as the active speaker, and support for multiple types of audio encoding. Unfortunately, the source code for `vat` has not yet been released. Binaries are available at `ftp.ee.lbl.gov`.

5.1.3 Network Voice Terminal: `nevot`

`nevot` is another tool for audio conferencing [11]. It supports the `vat` audio packets and provides additional features. The source code for `nevot` is available at `gaia.cs.umass.edu` in `/pub/nevot`. With a modular design, trace and statistics tools, and available source code, `nevot` makes for a good research tool.

5.1.4 Network Video: `nv`

`nv` is a tool that can send compressed video images across the network [12]. A dial is provided to adjust the maximum transmit bandwidth—a mechanism for a crude type of network admissions control. Audio/Video conferences typically run `nv` simultaneously with `nevot` or `vat`. The bandwidth control allows `nv` to co-exist with audio tools. Source code is available for `nv` from `parcftp.parc.xerox.com` in `/pub/net-research`.

5.1.5 Image Multicaster Client: `imm`

This program recives multicasted JPEG images [13]. Typically, these images are displayed in the root window. This program is usually invoked from `sd`. Binaries are available from `ftp.hawaii.edu` in `/paccom/imm`.

5.1.6 Audio/Video recorders

`vat` and `nv` sessions can be recorded for later use. `vat_nv_record.tar.Z`, available at `sics.se` in `/archive`), contains `vat_record`, `vat_play`, `nv_record`, `nv_play`, and `au_play`, as well as other utilities. Source code is included. Be warned that the distribution code uses the `usleep` command, therefore cannot be compiled on the SGI without slight modification. A modified version of these recording programs will be made available for the Xunet distribution.

5.1.7 Whiteboard: `wb`

The whiteboard program allows multiple users to draw and view graphical objects on a shared slate. Overheads can be shared among the members of the multicast group—a useful feature for a lecture. Changes and updates to the slate can be made and efficiently distributed in real–time. Postscript is used to represent the graphical and text objects.

`wb` is a unique multicast application in that it requires a *reliable* underlying IP multicast mechanism. Because source code is not available for `wb`, this section can only postulate on the underlying mechanism used to provide reliable multicast over a non–reliable multicast channel. By including a sequence number in each transmitted message, it is possible to determine when a transmitted message has been lost. If any of the receivers detects a missing packet they can request a retransmission (NACK) after a random timeout. If another receiver detects a NACK, it can cancel its timer to reduce the network traffic.

The binaries for `wb` are available from `ftp.ee.lbl.gov` in the `wb` subdirectory. Source code is not available.

5.2 Potential Multicast Applications

In addition to the existing applications listed above, below are some additional ideas for multicast applications that could be written.

5.2.1 Distributed File System

Reliable file transfer in a multicast environment could be implemented by a mechanism similiar to `tftp`, but with random delays on retransmission timers and suppressed acknowledgements, much like the reliable mechanism discussed in Section 5.1.7. By recursively transmitting files in a directory, it would be possible to efficiently transmit an entire directory tree of a file system, which would be an efficient mechanism for distributing source code, binaries, and documentation on a distributed system.

5.2.2 Network News

Rather than using NNTP over TCP to transport network news, the multicast mechanism could be used to quickly distribute the news to other news databases. These multicast groups would have the property of long group membership. Sequence numbers would be needed to ensure reliable delivery of the news items.

5.2.3 Distributed databases

Multicast is an efficient mechanism for searching distributed databases. One could imagine that a query would begin with a broadcast with a short time-to-live. If no database responds with an answer, the time-to-live field is increased, and the scope of the query is broadened. This process continues until a query can be resolved [14].

6 Suggested Xunet Multicast Channels

To standardize a common meeting place for Xunet discussion, I suggest the following multicast addresses be used for internal Xunet video and audio conferences. Applications can be invoked using `sd` as well as from the command line. Note that a `ttl` of 64 may be necessary to reach machines not directly connected to Xunet.

6.1 Video channel (nv)

Purpose: Open video conference among Xunet sites
Multicast address: `224.7.7.1`
Port: 1111
Initial ttl: 31
Suggested bandwidth: 128-512 kbps
To Invoke: `nv -ttl 31 224.7.7.1 1111`, or click from `sd`

6.2 Audio channel (vat, nevot)

Purpose: Open audio conference among Xunet sites
Multicast address: `224.7.7.2`
Port: 1112 *(Note: Port 1113 used for control)*
Initial ttl: 31
Audio encoding: 8-bit μ-law (default)
To Invoke: `vat -t 31 224.7.7.2/1112`, or click from `sd`

6.3 Whiteboard channel (wb)

Purpose: Common interactive whiteboard
Multicast address: `224.7.7.3`
Port: 1113
Initial ttl: 31

6.4 JPEG images (imm)

Purpose: Shared Still Images
Multicast address: `224.7.7.4`
Port: 1114
Initial ttl: 31

6.5 Radio Free Xunet (vat, nevot)

Purpose: Audio Streams
Multicast address: `224.7.7.5`
Port: 1115 *(Note: Port 1116 used for control)*
Initial ttl: 31
Audio Encoding: As Specified in `sd`

6.6 Experimental Xunet channels

Purpose: Testing and Debugging of new multicast applications
Multicast addresses: `224.7.7.7` or `224.7.7.8`
Initial ttl: Less than or equal to 31

7 IP Multicast over Xunet (Software-only Extensions)

Xunet sites could benefit greatly by using the ATM backbone for multicast traffic. The limited Internet bandwidth and congestion between Murray Hill and Berkeley, for example, cause noticable glitches in `vat` and `nv` sessions. The North American portion of the Mbone could greatly benefit if the decision were made to allow Xunet to act as a primary route for all Internet multicast traffic.

Without modifications to Xunet hardware or source code, the program `mrouted` would be run on each of the unicast Xunet routers. Implementing the IP multicast in software has a penalty, of course. Roughly, the multicast throughput, B_m, would be limited to the router throughput, B_r, divided by the number of tunnels, n. A reasonable topology could limit the fanout to approximately four tunnels ($n = 4$). Thus, the best–case throughput would be $B_m = B_r/n \approx$ 30Mbps/4 $\approx$ 8Mbps. Thus, Xunet could offer an eventual speedup of 16 over the original design goal of 500kbps for the Mbone.

The same topology information in Figure 7.1 is illustrated in Figure 7.2, in a more concise representation compatible with Mbone topology maps.

The following lists the initial suggested entries in the `/etc/mrouted.conf` at each of the specific Xunet routers.

7.1 mrouted.conf: peters.research.att.com

```
tunnel 192.20.225.14  192.12.211.10 metric 2 threshold 64 # jvnc-net
tunnel 192.128.57.1   192.128.57.5  metric 2 threshold 16 # uiuc
tunnel 192.128.57.1   192.128.57.8  metric 2 threshold 16 # rutgers
phyint 192.20.225.14  metric 1 threshold 16 # ethernet
phyint 192.128.161.11 metric 1 threshold 1  # fddi
```

7.2 mrouted.conf: morphius.cs.uiuc.edu

```
tunnel 192.128.57.5   192.128.57.1  metric 2 threshold 16 # peters
tunnel 192.128.57.5   192.128.57.3  metric 2 threshold 16 # berkeley
tunnel 192.128.57.5   192.128.57.4  metric 1 threshold 16 # wisc
tunnel 128.174.240.25 192.17.2.3    metric 2 threshold 64 # mbone-uiuc.cic
phyint 128.174.240.25 metric 1 threshold 16 # ethernet
phyint 192.17.240.3   metric 1 threshold 1  # fddi
```

7.3 mrouted.conf: r-ru.rutgers.edu

```
tunnel 192.128.57.8   192.128.57.1  metric 2 threshold 16 # peters
phyint 128.6.37.76    metric 1 threshold 16 # ethernet
phyint 128.6.91.7     metric 1 threshold 1  # fddi
```

7.4 mrouted.conf: sgi-xu.cs.wisc.edu

```
tunnel 192.128.57.4   192.128.57.5  metric 1 threshold 16 # uiuc
phyint 198.51.254.130 metric 1 threshold 16 # ethernet
```

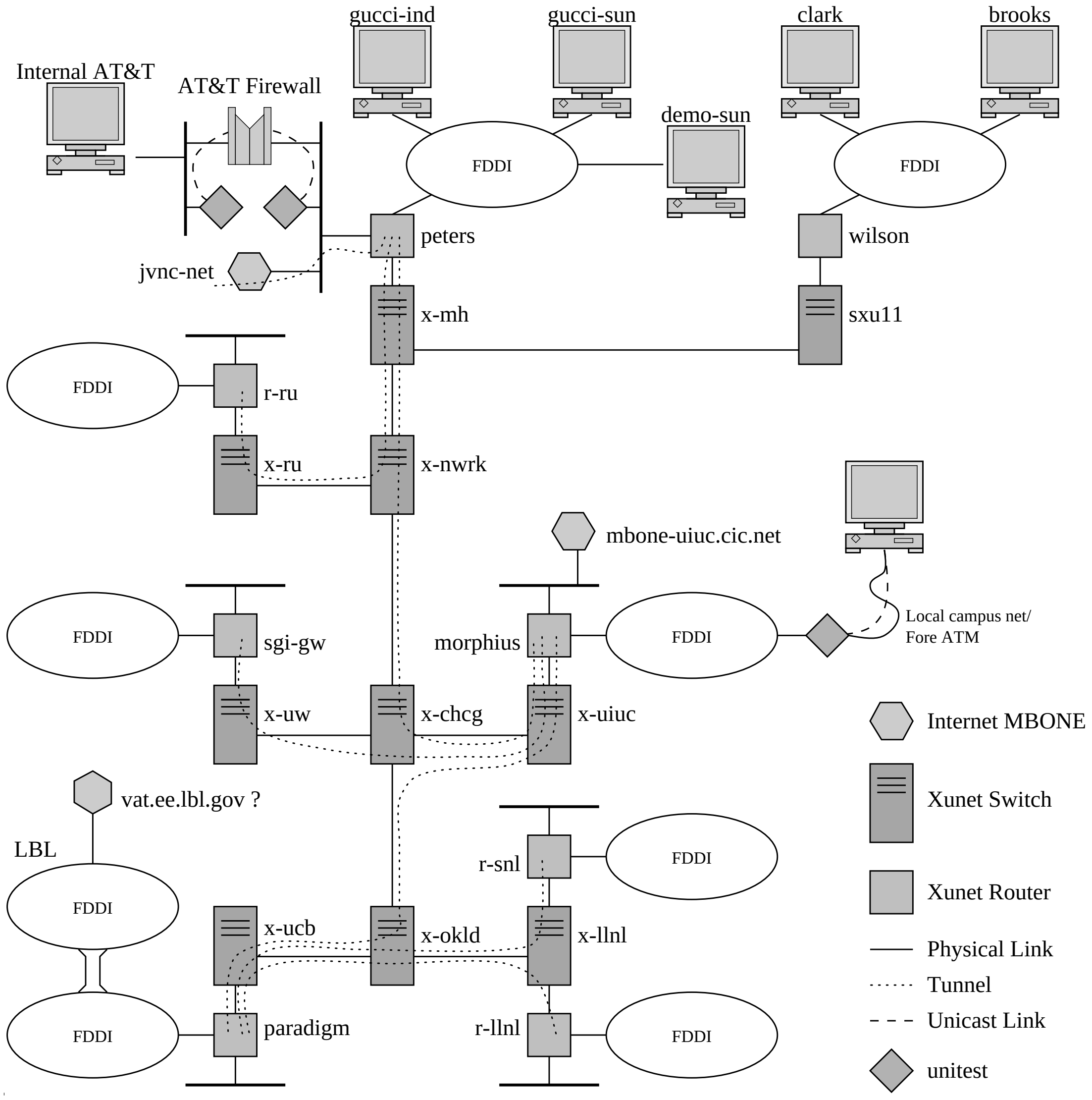

Figure 7.1: Possible Xunet Mbone Topology

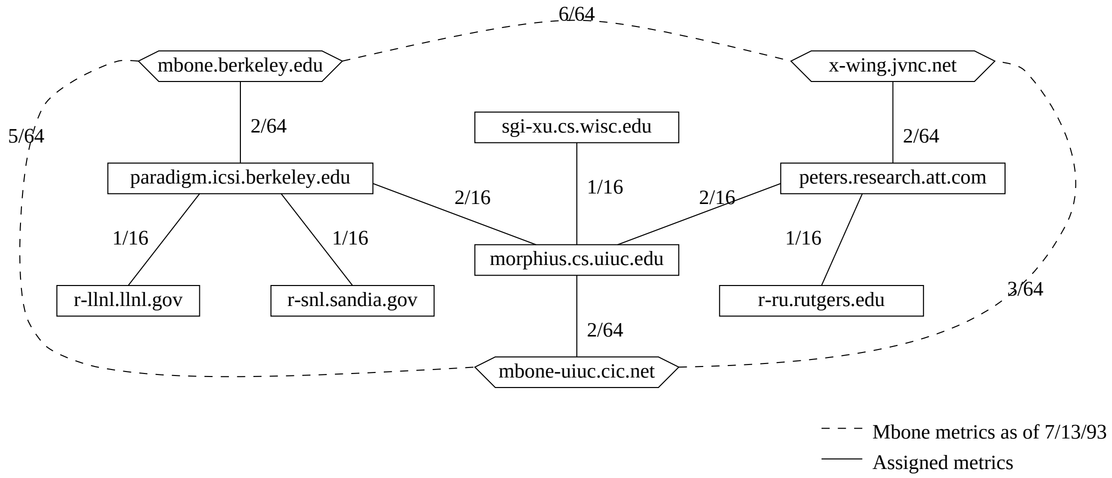

Figure 7.2: Mbone represenatation of Xunet topology

```
phyint 198.51.254.65  metric 1 threshold 1  # fddi
```

7.5 mrouted.conf: paradigm.icsi.berkeley.edu

```
tunnel 192.128.57.3  192.128.57.5  metric 2 threshold 16 # uiuc
tunnel 192.128.57.3  192.128.57.6  metric 1 threshold 16 # snl
tunnel 192.128.57.3  192.128.57.7  metric 1 threshold 16 # llnl
tunnel 128.32.201.88 192.31.161.1  metric 2 threshold 64 # mbone.berkeley
phyint 128.32.201.88  metric 1 threshold 16 # ethernet
phyint 192.107.102.66 metric 1 threshold 1  # fddi
```

7.6 mrouted.conf: r-llnl.llnl.gov

```
tunnel 192.128.57.7   192.128.57.3  metric 1 threshold 16 # llnl
phyint 128.115.37.101 metric 1 threshold 16 # ethernet
phyint 192.12.137.1   metric 1 threshold 1 # fddi
```

7.7 mrouted.conf: r-snl.sandia.gov

```
tunnel 192.128.57.6   192.128.57.3  metric 1 threshold 16 # snl
phyint 146.246.250.2  metric 1 threshold 16 # ethernet
phyint 192.160.228.1  metric 1 threshold 1  # fddi
```

8 Installation Guide

This section describes the modifications needed to support multicast on hosts and routers.

8.1 SGI Kernel-level modifications

The SGI IRIX kernel has multicast built-in. There were bugs in some of the existing kernel modules as supplied by SGI. The SGI multicast multicast extensions/bug fixes are located at `ftp.sgi.com` in `/sgi/ipmcast`. The Xunet router kernel includes the fixes to `ip_mroute.o` and `ip_input.o` that were current as of July 13, 1993.

The SGI kernel changes will be maintained in the master copy of the source code at Murray Hill in the `/usr/xunet/master/IRIX` directory. The default bootable image file will reflect the the changes listed above. Individual site administers need *not* make the changes described above.

8.2 User-level multicast programs program

The user-level multicast programs are available at `ftp.sgi.com` in `/sgi/ipmcast`. The binaries `mrouted`, `mrinfo`, and `map-mbone` should reside in the Xunet router's `/usr/etc` directory.

To start `mrouted` automatically whenever the system boots, change the contents of `/etc/config/mrouted` to `on`. By default, `mrouted` will expect to find its configuration file in `/etc/mrouted.conf`, and should be set according to Chapter 7.

These changes will be maintained on *peters* at Murray Hill and propagted to the remote Xunet sites. Individual site administers need *not* make the changes described above.

8.3 Xunet-specific multicast tools

The program `xumserv` should be started as root on each of the Xunet routers from the `hobbit.rc` initialization script. This program listens on a well-know, privileged TCP/IP port address for commands related to configuration of the multicast routing tunnels.

9 Debugging Guide

This section describes some tips on debugging multicast. The programs described in this section allow testing and debugging the multicast functionality at each site.

9.1 General Multicast Debugging Tools

The following tools have been created or modified to include multicast debugging operations [15]. These tools are available from `gregorio.stanford.edu` in the `vmtp-ip` directory. On a sun, These tools would *replace* your existing program of the same name. The standard SGI distribution includes versions of netstat and ping that already support the multicast options.

`netstat -M`	Print out a table similiar to `mrouted.dump`.
`mtest`	Interactive program for adding/deleting multicast groups
`ping`	Extra support for setting ttl, etc.
`map-mbone`	Map the Mbone connectivity
`mrinfo`	Get info about another router

9.2 Xunet-Specific Multicast Debugging Tools

The programs `xumrex` and `xumserv` have been written specifically to help manage multicast routing on Xunet. Xunet has the special property that it is necessary to be able to disable multicast routing at any time for demos, performance benchmarks, or testing. It is also necessary to provide a central coordinator with the ability to manage all the `mrouted` programs running on each of the multicast routers.

Because `mrouted` needs to run as root to manipulate the multicast routing tables, the process can only be restarted by the superuser. Because it is insecure to allow `rsh` for root across all of the Xunet routers, a a tool was desirable to allow a central coordinator to start, stop, and debug `mrouted` as well as to reconfigure the multicast configuration file (`mrouted.conf`).

`xumserv` listens on a privileged TCP/IP port address for commands. The program `xumrex` establishes a connection with `xumserv`, authenticates itself with a 16-byte cookie, then requests the execution of a multicast command. `xumrex` is started as: `xumrex machine command`, where `machine` is the name of the machine that should be contacted (such as *peters*), and `command` is one of the following:

```
 command: Description
--------  --------------------------------------------------
   mdown: Politely kill multicast routing daemon
     mup: Start multicast routing daemon
mrestart: Restart mrouted, useful after changing mrouted.conf
 mstatus: Print routing tables (mrouted.dump)
   medit: Set permissions on /etc/mrouted.conf to a+rw
 mnoedit: Write protect /etc/mrouted.conf
mgetconf: Print the configuration file (mrouted.conf)
mputconf: Write a new configuration file from stdin
```

9.3 Common problems

This section describes some common problems and how to fix them.

9.3.1 Multicast Application Problems

1. Verify program functionality: Check operation in *unicast* mode. (ie, try a point-to-point connection).
2. Ensure multicast host extensions have been made.
3. Try multicast to another machine on the same physical subnet.
4. If all of the above work properly, there may be something wrong with multicast routing. See below.

9.3.2 Multicast Routing Problems

1. Be sure the local router is alive (using `ping`).
2. Be sure the local router is running `mrouted` (using `ps`).
3. Use `netstat -M` to examine the multicasts vifs and routing table. If tunnels are down, see below.

9.3.3 Tunnel Problems

1. Be sure the remote machine is alive (using `ping`).
2. Be sure the remote machine running `mrouted` (using `ps`).
3. Check `/etc/mrouted.conf` on remote site. There must be a tunnel entry in both the local and remote sites before a tunnel can be configured.
4. Use `ping -T` *n dest-addr* with *n* equal to a ttl that can overcome the thresholds. (*Note: On the SGI use* `-T`*, while on the Sun use* `-t`).

9.3.4 Network Errors

1. If you get `Host Unreachable` messages from other machines on your network, it is likely that they are running an *outdated* operating system that does not support quitely dropping multicast packets that have had their `ttl` expire. The operating system on these machines should be updated.
2. If you have a new Proteon router that is attempting MOSPF routing, error messages will appear. MOSPF routing is *not* compatable with DVMRP routing. This problem is being addressed by John Moy at Proteon. For now, it is necessary to *disable* Proteon's multicast routing on the subnets interfacing with the Mbone.

10 Multicast Hardware Extensions

Implementing multicast in hardware would be optimal. Further, guaranteeing a Quality of Service (QoS) for the multicast traffic would make Xunet an optimal network for nationwide audio and video conferencing. Using 64 kbps `vat` audio and assuming a router throughput of 30 Mbps, Xunet could support approximately 468 simultaneous `vat` sessions (30Mbps/64kbps). Using 200kbps `nv` video combined with 64 kbps audio would allow approximately 113 simultaneous audio/video conferences (30Mbps/264kbps). Thus, Xunet could offer an eventual speedup of 60 over the original Mbone design goal.

The Xunet switches can support multicast. There is some work to be done, however. The following steps would need to be implemented:

- Switch Translation Card: Add broadcast address; Bypass VCI consistency check.
- Queue Module: Set RAT service list for non-participating sites.
- Signalling code: Add a protocol for multicast call setup.
- IP Routing code: Implement multicast host extensions for Xunet device.

A Multicast Programming Examples

The following programming examples demonstrate how to write a multicast application and provide a library of multicast procedures.

A.1 Multicast Libraries: mlib.c

The following are the basic multicast library calls.

```
/*************************************************/
/* mlib.c: Example multicast application         */
/*                                               */
/* John lockwood                                 */
/* Xunet Project: AT&T Bell Labs                 */
/* Inspirations from udp.c by Henning Schulzrinne */
/*************************************************/

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h> /* defines multicast structures */
#include <netdb.h>
#include <stdio.h>
#include <errno.h>
#include <arpa/inet.h>

extern int errno;

/*****************************************************************************/
int die(errormes)
  char *errormes;
{
  perror(errormes);
  /* exit(-1); */
}

/*****************************************************************************/
int UDP_RX_connect(maddr, recv_port, inputp)
unsigned long maddr;   /* Multicast Address (already in network byte order) */
int recv_port;         /* input port (host order) */
int *inputp;           /* socket for data input */
{
  int one = 1;               /* Used to pass "on" through setsockopt */
  struct sockaddr_in Sin;    /* binding info */
  struct ip_mreq mreq;       /* Multicast request message (from netinet/in.h) */

  /* create input socket */
  if ((*inputp = socket(AF_INET, SOCK_DGRAM, 0)) < 0)  /* Open socket */
    die("UDP_RX_connect: can't open socket");

  /* Allow address reuse (Not necessary for all applications) */
  if (setsockopt(*inputp,SOL_SOCKET,SO_REUSEADDR,(char *)&one,sizeof(one)) < 0)
    die("UDP_RX_connect:setsockopt(REUSE)");
```

```
  /* bind it */
  Sin.sin_family      = AF_INET;
  Sin.sin_port        = htons(recv_port); /* network format short */
  Sin.sin_addr.s_addr = IN_MULTICAST(ntohl(maddr)) ? maddr : INADDR_ANY ;
  if (bind(*inputp, (struct sockaddr *)&Sin, sizeof(Sin)) < 0)
    die("UDP_RX_connect:Binding");

  /* Tell the multicast extensions that we are adding ourselves */
  if (IN_MULTICAST(ntohl(maddr))) {
    mreq.imr_multiaddr.s_addr = maddr; /* multicast address */
    mreq.imr_interface.s_addr = htons(INADDR_ANY);  /* 0x00000000 */
    if (setsockopt(*inputp,IPPROTO_IP,IP_ADD_MEMBERSHIP,
        (char *)&mreq,sizeof(mreq)) < 0)
      die("UDP_RX_connect:setsockopt(IP_ADD_MEMBERSHIP)");
  }

  return(0);
} /* UDP_RX_connect */

/****************************************************************************/
int UDP_TX_connect(maddr, outputp, ttl)
  unsigned long maddr;   /* Multicast Address (already in network byte order)*/
  int *outputp;          /* socket for data output */
  u_char ttl;            /* initial time-to-live (but can be changed) */
{
  u_char zero = 0;       /* output loop back */
  struct ip_mreq mreq;   /* multicast group */
  int len;

  if ((*outputp = socket(PF_INET, SOCK_DGRAM, 0)) < 0)
    die("UDP_TX_connect: can't open socket");

  if (IN_MULTICAST(ntohl(maddr))) {
    /* disable loopback */
    if (setsockopt(*outputp,IPPROTO_IP,
        IP_MULTICAST_LOOP,(char *)&zero,sizeof(zero)) < 0) {
      if (errno == EINVAL)
        die("UDP_TX_connect:no multicast");
      else
        die("UDP_TX_connect:setsockopt(M_LOOP)");
    }

    len = sizeof(ttl);
    /* set initial time-to-live */
    if (setsockopt(*outputp,IPPROTO_IP,IP_MULTICAST_TTL,&ttl,sizeof(ttl)) < 0)
        die("UDP_TX_connect:setsockopt(TTL)");
  }
  return 0;
} /* UDP_TX_connect */

/****************************************************************************/
int UDP_RX(inputp, buf, len, from, fromlen)
  int inputp;               /* socket to read from */
```

```
  char *buf;                   /* buffer */
  int len;                     /* size of buffer */
  struct sockaddr *from;       /* source port, address */
  int *fromlen;
{
  int rcount;                    /* number of bytes received */

  *fromlen = sizeof(*from);
  rcount = recvfrom(inputp,buf,len,0,from,fromlen);
  return rcount;
}

/****************************************************************************/
int UDP_RX_printsource(from, fromlen)
  struct sockaddr_in *from;      /* source port, address */
  int *fromlen;

{
  printf("From: Address: %s, Port: %d\n", inet_ntoa(from->sin_addr),
    ntohs(from->sin_port));
}

/****************************************************************************/
int UDP_TX(outputp, maddr, port, buf, len)
  int outputp;                 /* socket to write to */
  unsigned long maddr;         /* Multicast Address (network order) */
  int port;                    /* output port (host order) */
  char *buf;                   /* data to be xmitted */
  int len;                     /* length of data to send */
{
  static struct sockaddr_in to;

  to.sin_family = AF_INET;
  to.sin_port = htons(port);
  to.sin_addr.s_addr = maddr;

  return sendto(outputp,buf,len,0,&to,sizeof(to));
}

/****************************************************************************/
int UDP_RX_close(inputp, maddr)
int inputp;           /* stream descriptor */
unsigned long maddr;   /* Multicast Address (already in network byte order) */
{
  struct ip_mreq mreq;      /* multicast group */

  mreq.imr_multiaddr.s_addr = maddr;
  mreq.imr_interface.s_addr = INADDR_ANY; /* Local IP address of Interface */

  if (setsockopt(inputp, IPPROTO_IP, IP_DROP_MEMBERSHIP,
      (char *)&mreq, sizeof(mreq)) < 0)
    die("UDP_RX_close:setsockopt(DROP)");

  if (close(inputp)<0)
```

```
    die("UDP_RX_close: close failed\n");

  return 0;
} /* UDP_RX_close */

/****************************************************************************/
int UDP_TX_close(outputp)
int outputp;            /* stream descriptor */
{
  if (close(outputp)<0)
    die("Error: UDP_TX_close: close failed");
  return 0;

} /* UDP_TX_close */

/****************************************************************************/
int UDP_TX_ttl(outputp, ttl)
  int outputp;   /* socket to be modified */
  int ttl;       /* new ttl value */
{
  u_char ttlc;
  int rc;
  int len;       /* length returned */

  if (ttl < 0 || ttl > 255) return -1;
  ttlc = ttl;
  printf("Setting ttl to %d\n",ttlc);
  if ((rc=setsockopt(outputp,IPPROTO_IP,IP_MULTICAST_TTL,&ttlc,sizeof(ttlc)))<0)
    die("UDP_TX_ttl: setsockopt(TTL)");
  ttlc = 0;
  len = sizeof(ttlc);
  if (getsockopt(outputp,IPPROTO_IP,IP_MULTICAST_TTL,&ttlc,&len)<0)
    die("UDP_TX_ttl: getsockopt(TTL)");
  printf("Returned length: %d, value: %d\n",len,ttlc);
  return(rc);
} /* UDP_ttl */
```

A.2 Multicast Testing Program: mcast.c

This section illustrates multicast programming from a user application. This program may be run as either a multicast transmitter or a receiver, depending on the command line arguments (`mcast -t` or `mcast -r`).

As a receiver, this program opens a datagram socket and receives multicast datagrams. The source (IP address and port number) is printed. The contents of the UDP message are printed as well (it is assumed that the message contains a text string).

As a transmitter, this program sends multicast datagrams at a rate of one per second. The time–to–live field `ttl` is incremented by eight for each datagram sent. The program ceases when the `ttl` would exceed 255.

This program simplifies debugging, as a descriptive error message is printed for any possible error condition (such as mis–installed multicast extensions). This program also allows testing multicast routers, in a simple manner similar to ping.

```
/************************************/
/* mcast.c:                         */
/*                                  */
/* John Lockwood                    */
/* Xunet Project: AT&T Bell Labs    */
/************************************/

#include "mlibkr.h"

#define TXMODE 1
#define RXMODE 2

main(argc, argv)
  int argc;
  char *argv[];
{
  char mcaststr[16];                /* IP address (dotted decimal string) */
  unsigned long maddr;              /* multicast address (network order)  */
  int port;                         /* Transmit/receive port              */
  int inputp, outputp;              /* transmit/receive sockets           */
  int ttl = 128;                    /* time-to-live                       */
  struct sockaddr from;             /* source of datagram (port, address) */
  int fromlen;                      /* length of above                    */
  int rcount;                       /* Bytes received from recvfrom       */
  char Rbuf[1000], Tbuf[1000];      /* Receive/Transmit buffer            */
  int mode;                         /* Set depending on arguments to prog.*/
  int i;

  if (argc != 2) {
    printf("Usage: %s [-t] [-r]\n",argv[0]); exit(-1); }

  if (strcmp(argv[1],"-t")==0) { printf("Transmitting: \n"); mode=TXMODE; }
  else {
    if (strcmp(argv[1],"-r")==0) { printf("Receiving: \n"); mode=RXMODE; }
    else { printf("You must specify either -t or -r\n"); exit(-1); }
  }

  port = 5555 ;                     /* Pick an arbitrary multicast port  */
  strcpy(mcaststr,"224.7.7.7");     /* Pick an arbitrary class D address */

  strcpy(Tbuf,"Hello, world\n");    /* message to trasmit                */

  if ((maddr=inet_addr(mcaststr))==-1)
    die("Invalid IP string");

  printf("Opening RX_connect\n");
  UDP_RX_connect(maddr, port, &inputp);
  printf("Opening TX_connect\n");
  UDP_TX_connect(maddr,&outputp, ttl);

  if (mode==TXMODE) {
    i=0;
```

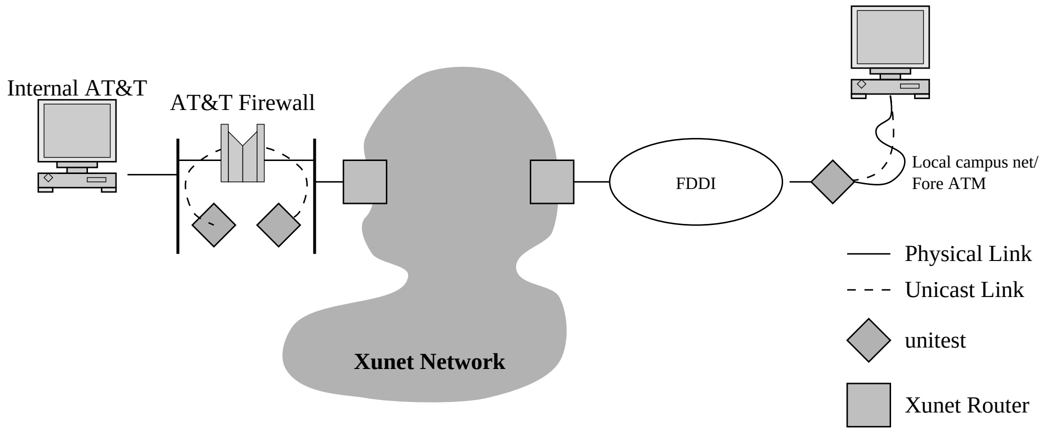

Figure A.1: Function of unitest.c

```
    while((i+=8) < 255) {
      printf("Sent datagram with ttl=%d\n",i);
      UDP_TX(outputp, maddr, port, Tbuf, strlen(Tbuf));
      UDP_TX_ttl(outputp,i);
      sleep(1);
    }
  }

  if (mode==RXMODE) {
    while(1) {
      rcount = UDP_RX(inputp, Rbuf, 1000, &from, &fromlen);
      printf("UDP_RX: %d bytes received\n",rcount);
      if (rcount<0)
        fprintf(stderr,"error: recvfrom errors");
      else {
        UDP_RX_printsource(&from, &fromlen);
      }
    }
  }

  UDP_RX_close(inputp,maddr);
  UDP_TX_close(outputp);

  printf("successful termination\n");
}
```

A.3 Multicast/Unicast Packet Forwarding: unitest.c

This program listens to an attached UDP port, and sends on another UDP port. This program can support all permutations of sending/receving on unicast/multicast addresses. This program allows users at University sites to set up a connection to their local Xunet router (or a machine on the FDDI ring), then participate in the multicast application (such as vat, nv, etc) from their office machine, which may be separated by IP routers and gateways, as shown in Figure A.1.

```
#include "mlibkr.h"

/************************************/
/* unitest.c:                       */
/*                                  */
/* John Lockwood                    */
/* Xunet Project: AT&T Bell Labs    */
/************************************/

#define MAXBUFSIZE 70000

main(argc, argv)
  int argc;
  char *argv[];
  {
  int rxfd, txfd;
  char rxaddrstr[16], txaddrstr[16];
  u_long rxmaddr, txmaddr;
  int rxport, txport;
  int txttl;
  char buf[MAXBUFSIZE];
  int buflen;
  struct sockaddr from;
  int fromlen;
  int i;

  printf("%d\n",argc);
  if (argc!=5) {
    printf("Usage: %s RXaddr RXport TXaddr TXport\n",argv[0]); exit(-1); }

  strcpy(rxaddrstr,argv[1]);
  rxmaddr=inet_addr(rxaddrstr);
  sscanf(argv[2],"%d",&rxport);
  strcpy(txaddrstr,argv[3]);
  txmaddr=inet_addr(txaddrstr);
  sscanf(argv[4],"%d",&txport);

  printf("RX: %s %d; TX: %s %d\n",rxaddrstr,rxport,txaddrstr,txport);

  txttl=200;

  UDP_RX_connect(rxmaddr,rxport,&rxfd);
  UDP_TX_connect(txmaddr,&txfd,txttl);

  for (i=0;; ) {
    buflen = UDP_RX(rxfd, buf, MAXBUFSIZE, &from, &fromlen);
    /* UDP_RX_printsource(&from, &fromlen); */
    UDP_TX(txfd, txmaddr, txport, buf, buflen);
  }

  UDP_RX_close(rxfd, rxmaddr);
  UDP_TX_close(txfd);
}
```

B Remote mrouted Management: xumrex and xumserv

This chapter provides the source code for **xumserv** and **xumrex**, the programs that allow remote management and configuration of multicast routing and tunnels. Each site is asked to run xumserv as root from their /usr/local/xunet/adm/hobbit.rc initialization scripts. A user's guide to these programs is provided in Section 9.2

B.1 Xunet Multicast Management Server: xumserv.c

```
/****************************************************/
/* xumserv.c:                                       */
/*                                                  */
/* Server portion of remote mrouted contorl program. */
/* This program should be run as root.              */
/* Provides security by limiting command set and    */
/* by a 16-byte cookie.                             */
/*                                                  */
/* John lockwood                                    */
/* Xunet Project: AT&T Bell Labs                    */
/* Inspirations from Srinivasan Keshav's serv.c     */
/****************************************************/

#include "xumserv.h"

struct mcommand mc;
char buffer[BUFLEN];
char sbuf[80];
int msgsock, sock;

sendfile(fname) /* Sends a file to remote socket */
  char *fname;
{
  int fd, len;
  if (( fd = open(fname,O_RDONLY) )<0) {
    sprintf(sbuf,"File %s could not be opened\n",fname);
    send(msgsock,sbuf,strlen(sbuf),0);
    }
  else {
    while (( len=read(fd,buffer,BUFLEN))>0)
      send(msgsock,buffer,len,0);
    }
  close(fd);
}

putfile(fname) /* Writes a file using data from a remote socket */
  char *fname;
{
  int fd, len;
  if (( fd = creat(fname,0666) )<0) {
    sprintf(sbuf,"File %s could not be opened\n",fname);
```

```
    send(msgsock,sbuf,strlen(sbuf),0);
    }
  else {
    len=1;
    buffer[0]=0x1;
    while (buffer[len-1] != 0x4) {
      len = recv (msgsock, buffer, BUFLEN, 0);
      write (fd, buffer, len);
      }
    close(fd);
    }
}


/* Supported Xunet Command */
/* ---------------------------------------------------------------------- */
void mdown() { /* Politely kill multicast routing daemon, stop mcast routing */
  system ("/etc/killall mrouted");
  sprintf(sbuf,"mdown: mrouted killed\n");
  send(msgsock,sbuf,strlen(sbuf),0);
  }
void mup() { /* Start multicast routing daemon */
  system ("/usr/etc/mrouted -c /etc/mrouted.conf");
  sprintf(sbuf,"mup: mrouted started\n");
  send(msgsock,sbuf,strlen(sbuf),0);
  }
void mrestart() { /* Restart mrouted, useful after changing mrouted.conf */
  mdown();
  mup();
  }
void mstatus() { /* Dump routing tables to /usr/tmp/mrouted.dump */
  unlink("/usr/tmp/mrouted.dump"); /* delete this file */
  system ("/etc/killall -USR1 mrouted");
  sleep(1);
  sendfile("/usr/tmp/mrouted.dump");
  }
void medit() { /* This file needs to be editable */
  system ("/bin/chmod a+rw /etc/mrouted.conf");
  sprintf(sbuf,"medit: Configuration file now editable\n");
  send(msgsock,sbuf,strlen(sbuf),0);
  }
void mnoedit() { /* Write protect the .conf file */
  system ("/bin/chmod og-w /etc/mrouted.conf");
  sprintf(sbuf,"mnoedit: Configuration file secured\n");
  send(msgsock,sbuf,strlen(sbuf),0);
  }
void mgetconf() { /* Get the contents of the configuration file */
  sendfile("/etc/mrouted.conf");
  }
void mputconf() { /* Write a new configuration file */
  putfile("/etc/mrouted.conf");
  sprintf(sbuf,"Configuration file written\n");
  send(msgsock,sbuf,strlen(sbuf),0);
  }
```

```
/* ------------------------------------------------------------------------ */

void docommand(command)
    char *command;
{
    if (strcmp(command,"mdown")==0)    mdown();    else
    if (strcmp(command,"mup")==0)      mup();      else
    if (strcmp(command,"mrestart")==0) mrestart(); else
    if (strcmp(command,"mstatus")==0)  mstatus();  else
    if (strcmp(command,"medit")==0)    medit();    else
    if (strcmp(command,"mnoedit")==0)  mnoedit();  else
    if (strcmp(command,"mgetconf")==0) mgetconf(); else
    if (strcmp(command,"mputconf")==0) mputconf(); else
      { sprintf(sbuf,"Unknown command\n");
        send(msgsock,sbuf,strlen(sbuf),0);
      }

}
/* ------------------------------------------------------------------------ */

main (argc, argv)
    int             argc;
    char          **argv;
{
    int             length;
    struct sockaddr_in server;
    int             rval;

    sock = socket (AF_INET, SOCK_STREAM, 0);
    if (sock < 0)
    {
perror ("opening stream");
exit (0);
    }
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = INADDR_ANY;
    server.sin_port = htons (PORT);

    if (bind (sock, &server, sizeof (server)))
perror ("binding");
    length = sizeof (server);
    if (getsockname (sock, &server, &length))
    {
perror ("getting sock name");
exit (0);
    }
    listen (sock, 5);
    while (TRUE)
    {
msgsock = accept (sock, 0, 0);
if ((rval = recv (msgsock, buffer, BUFLEN, 0)) < 0)
    perror ("reading msg");
else {
```

```
            bcopy(buffer,&mc,sizeof(mc));
    if (rval != sizeof(mc))
         perror ("wrong size command");
     else {
     if (strcmp(mc.cookie,COOKIE)!=0)
    perror("Invalid cookie");
else docommand(mc.command);
             }
        }
    buffer[0]=0x4;
    send(msgsock,buffer,1,0);
    close (msgsock);
    }
}
```

B.2 Xunet Remote Execution Client: xumrex.c

```
/*****************************************************/
/* xumrex.c:                                         */
/*                                                   */
/* Client portion of remote mrouted contorl program. */
/* This program can be run from any machine in the   */
/* network.  It contacts the remote xumserv, provides */
/* a cookie for security, then requests that a       */
/* command (from a subset specifed by xumserv) be    */
/* executed.                                         */
/*                                                   */
/* John lockwood                                     */
/* Xunet Project: AT&T Bell Labs                     */
/* Inspirations from Srinivasan Keshav's rex.c       */
/*****************************************************/

#include "xumserv.h"

main (argc, argv)
    int             argc;
    char          **argv;
{
    int             sock, len;
    struct sockaddr_in server;
    struct hostent *hp, *gethostbyname ();
    struct mcommand mc;
    char buffer[BUFLEN];
    int i;

    sock = socket (AF_INET, SOCK_STREAM, 0);
    if (sock < 0)
    {
perror ("opening socket ");
exit (0);
    }
```

```
    server.sin_family = AF_INET;

    /* On-line help menu */
    if (argc <3) {
      printf("Usage: xumrex machine command \n\n");
      printf(" command: Description\n");
      printf("--------  ---------------------------------------------------\n");
      printf("   mdown: Politely kill multicast routing daemon\n");
      printf("     mup: Start multicast routing daemon\n");
      printf("mrestart: Restart mrouted, useful after changing mrouted.conf\n");
      printf(" mstatus: Print routing tables (mrouted.dump)\n");
      printf("   medit: Set permissions on /etc/mrouted.conf to a+rw\n");
      printf(" mnoedit: Write protect /etc/mrouted.conf\n");
      printf("mgetconf: Print the configuration file (mrouted.conf)\n");
      printf("mputconf: Write a new configuration file from stdin\n\n");
      exit(0);
      };
    hp = gethostbyname (argv[1]);
    bcopy (hp->h_addr, &(server.sin_addr.s_addr), hp->h_length);
    server.sin_port = htons (PORT);

    if (connect (sock, &server, sizeof (server)) < 0)
    {
close (sock);
perror ("conn sock ");
exit (0);
    }

    strcpy(mc.cookie,COOKIE);
    strcpy(mc.command,argv[2]);
    bcopy(&mc,buffer,sizeof(mc));

    if (send (sock, buffer, sizeof(mc), 0) < 0)
perror ("writing");

    /* Read from stdin, write to remote /etc/mrouted.conf */
    if (strcmp(argv[2],"mputconf")==0)  {
      while(( len = read (0, buffer, BUFLEN)) > 0)
        send (sock, buffer, len, 0);
      buffer[0]=0x4;
      send(sock, buffer ,1,0);
      }

    printf("Return message from %s: \n",argv[1]);

    len=1;
    buffer[0]=0x1;
    while (buffer[len-1] != 0x4) {
      len = recv (sock, buffer, BUFLEN, 0);
      write (1, buffer, len);
      }

    write(1,"\n",1);
    close (sock);
```

```
}
```

B.3 Common Header: xumserv.h

```
/***************************************************/
/* xumserv.h:                                      */
/*                                                 */
/* common header for xumrex.c and xumerv.h         */
/*                                                 */
/* John lockwood                                   */
/* Xunet Project: AT&T Bell Labs                   */
/***************************************************/

#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <fcntl.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>
#include <unistd.h>

#define TRUE 1
#define FALSE 0
#define BUFLEN 4096
#define PORT 955
#define COOKIE "xmuunletti"
/* Not the actual cookie, of course */

struct mcommand {
  char cookie[16];
  char command[16];
  } mc;
```

B.4 Makefile for xumrex and xumserv

```
CC = cc
CFLAGS = -O
PROGS = xumserv xumrex

all: $(PROGS)

xumserv: xumserv.o
$(CC) $(CFLAGS) -o xumserv xumserv.o

xumrex: xumrex.o
$(CC) $(CFLAGS) -o xumrex xumrex.o
```

```
.c.o:
$(CC) $(CFLAGS) -c $*.c

xumserv.o: xumserv.c xumserv.h
xumrex.o: xumrex.c xumserv.h
```

Bibliography

[1] A. G. Fraser, C. R. Kalmanek, A. E. Kaplan, W. T. Marshall, and R. C. Restrick, "Xunet 2: A nationwide testbed in high–speed networking," in *INFOCOM*, pp. 582–589, 1992.

[2] A. Berenbaum, J. Dixon, A. Iyengar, and S. Keshav, "A flexible ATM-host interface for Xunet II," *IEEE Network*, pp. 18–23, July 1993.

[3] S. Casner and S. Deering, "First IETF internet audiocast," *ACM Sigcomm Computer Communications Review*, vol. 22, July 1992.

[4] S. Casner, "mbone FAQ." Anonymous ftp from `nic.sura.net` in `/pub/mbone`, May 1993.

[5] S. Deering, "Host extensions for IP multicasting." RFC 1112, Aug. 1989.

[6] S. Deering and D. Cheriton, "Multicast routing in datagram internetworks and extended LANs," *ACM Transactions on Computer Systems*, pp. 85–110, May 1990.

[7] S. Deering, "CERFnet seminar: MBONE—the mulicast backbone," 1993.

[8] S. Deering, "`mrouted` man page." Anonymous ftp as `ipmulti-sunos41x.tar`, July 1993.

[9] V. Jacobson, "Session directory (`sd`) README." Anonymous ftp from `ee.lbl.gov`, Mar. 1993.

[10] V. Jacobson and S. McCanne, "Visual audio tool (`vat`) man page." Anonymous ftp from `ee.lbl.gov`, Feb. 1992.

[11] H. Schulzrinne, "Voice communication across the internet: A network voice terminal." Anonymous ftp from `gaia.cs.umass.edu` as `/pub/nevot/nevot.ps.Z`, Aug. 1992.

[12] R. Frederick, "Network video (`nv`) man page." Anonymous ftp from `parcftp.xerox.com` in `/pub/net-research`, Mar. 1993.

[13] W. Dang, "Image multicaster client (`imm`) man page." Anonymous ftp as `/paccom/imm/imm.tar` from `ftp.hawaii.edu`, Aug. 1993.

[14] D. R. Boggs, "Internet broadcasting." Ph.D Thesis, Electrical Engineering Dept., Stanford University, Jan. 1982.

[15] S. Deering, "IP multicast extensions for 4.3BSD UNIX and related systems." Anonymous ftp from `gregorio.stanford.edu` as `vmtp-ip/ipmulticast.README`, June 1989.