# Measurement of Join Latency on the MBone

Abhinav Garg [†],Sneha Kumar Kasera[†], Rohan Kumar[‡], Don Towsley[†]
[†]Department of Computer Science, University of Massachusetts, Amherst, USA
[‡]Microsoft Corporation

**CMPSCI Technical Report TR 99-47**

August 5, 1999

**Abstract**

Protocols and services using multiple multicast groups depend on multicast join/leave latencies, and join/leave processing overheads being low. Hence it is extremely important to study these latencies and processing overheads. In this document, we propose experimental approaches to measure join latencies on the IP Multicast Backbone, MBone. Using one approach, we measured join latencies at a receiver on the Mbone. We found that pruning was not taking place properly in the current MBone. We also found that existing IGMPv2 protocol fails to take into account all the interoperability scenarios with IGMPv1 protocol which might result in higher leave latencies. A solution to correct this problem is proposed which requires modifications to IGMPv2 protocol at the multicast router only.

## 1 Introduction

Recently there has been a surge of proposals for protocols and services that use multiple multicast groups for reliable multicast, flow and congestion control [9], [10], [13] and Video-on-Demand services [6], [8], [7].

The success of these protocols and services depends on multicast join/leave latencies, and join/leave processing overheads being low. Hence it is extremely important to determine these latencies and processing overheads.

To the best of our knowledge, there has been no experimental study of join and leave operations. In this document, we propose experimental approaches to measure join latencies on the IP Multicast Backbone, MBone. We measured join latencies at a receiver on the Mbone. We found that pruning was not taking place properly in the current MBone. We also found that existing IGMPv2 protocol fails to take into account all the interoperability scenarios with

IGMPv1 protocol which might result in higher leave latencies. We propose a solution to correct this problem which requires modifications to IGMPv2 protocol at the multicast router only.

The rest of the report is organized as follows. In Section 2, we present related work. Section 3 provides describes IGMPv1 and IGMPv2. In Section 4, we give two possible definitions of Join Latency and identify the one we have used. Section 5 describes the experiment setup and some implementation aspects. The interoperability problem between IGMPv1 and IGMPv2 and its solution is presented in Section 6. We discuss the problems we faced in measuring the join latency values and findings in Section 7. The results of our experiments and some observations are presented in Section 8. In Sections 9 and 10, we present our conclusions from observations and talk about the future work.

## 2 Related Work

[12] proposes a mechanism for reducing the leave latency when a host leaves a multicast group. The mechanism is based on predicting the multicast group membership using previous group membership information and is similar to those used in RISC processors for optimizing jump performance. [14] discusses the overheads involved due to the leave latency for Point-to-Point (PtP) links and presents an alternative mechanism to current IGMP query/reply model for keeping track of group memberships over PtP networks. A PtP link connects only one end host to a router. To the best knowledge of the authors, no work has been done to measure the actual join/leave latencies on the MBone.

## 3 Background

In the current IP multicast model, a multicast router on a LAN uses a group membership protocol to find out which multicast hosts on the same LAN are members of a multicast group. The router also uses a multicast routing protocol to communicate with other attached multicast routers and receive/send multicast traffic. The multicast routing protocol used is independent of the group membership protocol being used. From this point on, we use the terms multicast router and router interchangeably.

We have measured join latency values on the current MBone. DVMRP (Distance Vector Multicast Routing Protocol) [11] is the multicast routing protocol which is widely deployed on the current MBone and IGMP protocol (IGMPv1 or IGMPv2) is the group membership protocol. In the next two subsections, we describe the IGMPv1 and IGMPv2 protocols.

### 3.1 IGMPv1

In IGMPv1 [1], a multicast router on a LAN sends out a query, with a maximum response time of 10sec, no more than once a minute to determine the group memberships. If no response to a sequence of queries is received for a particular group, the router assumes the absence of a local group member. The exact number of such queries after which a router assumes there is no local group member is not specified in the RFC 1112.

## 3.2 IGMPv2

In IGMPv2 [5], a router sends out a General Query every 125sec, with a maximum response time of 10 sec. The router maintains a list of all the multicast group memberships on the network and a timer for each membership. This timer is set to the Group Membership Interval whose value is equal to the (((the Robustness Variable) times (the Query Interval)) plus (one Query Response Interval)). The router should receive an IGMPv2 Report message as a response to the General Query message. If no Report for a group is received within the Group Membership Interval, the router assumes that there is no member of the group present on the subnetwork.

The Robustness Variable is used to handle IGMP packet losses on the subnetwork and its default value is 2. If the subnetwork is expected to be lossy, the value of the Robustness Variable can be increased. The Query Interval is the interval between General Queries sent by the router. Its default value is 125 seconds and a system administrator can vary it to tune the rate at which IGMP messages are generated on a network. A Query Response Interval is the Max Response Time that an IGMP host is allowed during which it can send a IGMPv2 Report message as a response to the General Query message. The default value of Query Response Interval is 10 seconds. Using these default values, the default value of Group Membership Interval is 260 seconds. The Robustness Variable can also be viewed as the number of General Queries which are sent out during the Group Membership Interval. Thus, if no host responds to two General Queries, the router assumes that that there is no local group member.

Also, in order to reduce leave latency, IGMPv2 requires a group member to send out a Leave Group message when it is the last member to leave a group. When a router receives such a message, it sends several Group-Specific Queries to find out whether any host is still a part of the group or not. By default, a host which is still a part of the group must respond to a Group-Specific Query within 1 sec. of the time the query was sent out and the number of such Group-Specific Queries is set to 2. If no response to the total number of [Last Member Query Count] Group-Specific Queries is received by the router, then the router assumes that there is no local group member. By default, the value of Last Member Query Count is equal to the Robustness Variable.

# 4 Join Latency

In this section, we introduce two definitions of join latency. Consider Figure 1. A host H on the LAN wishes to join group G and issues an IGMP Request (to the LANs multicast router R1) at time $T_1$ . If there are no other existing members of group G in the LAN, R1 sends a DVMRP graft message [11] for group G to its upstream router in the direction of the sender. The graft message propagates towards the sender until it reaches either a branching point (router forwarding packets addressed to group G) or the sender's subnet multicast router. Let R2 be the branching point. Let the local multicast router R1 send a DVMRP graft message for group G to its upstream neighbour at time $T_2$. Let R2 receive the graft message (which originated at R1) at time $T_3$ and H receive the first packet after joining G at time $T_4$. We refer to the reverse path from H to R2 in which the graft propagates as the **slow path** and the forward path from

R2 to H in which packets travel downstream as the **fast path**. Figure 2 shows the timeline of sequence of operations in a join latency experiment.

The above process would still be the same if the multicast routing protocol being used is PIM-DM (Protocol Independent Multicast - Dense Mode) [2]. In case of PIM-SM (Protocol Independent Multicast - Sparse Mode) [3], [4], the process would be similar for the shared distribution tree except that router R1 would send explicit join towards the Rendezvous Point (RP) and a graft would propagate towards the RP. If R1 wants to switch from the shared tree to Shortest Path Tree (SPT), it would send a join towards the source.

Join latency can be defined in two ways:

- 1. $S_1$ = ($T_4$ - $T_1$): This is the way that multicast routing protocols define join latency [1]. We call it the Application Specific Latency. This is the latency which we have measured.
- 2. $S_2$ = ($T_3$ - $T_1$): This is the time that is actually taken to graft to the nearest multicast router R2. This can be approximated by ($T_3$ - $T_2$).

Initially, we wanted to measure $S_2$. However, this requires that we be able to monitor the multicast router R2. R2 can either be a sender's subnetwork router or a multicast router belonging to an ISP.

If the graft has reached the sender's subnetwork router, then R2 is the sender's subnetwork router, and there is no member belonging to the multicast group outside the sender's subnetwork and the source traffic has been limited to the source subnetwork after the pruning of the distribution tree. By using netstat -C menu option 8 on a sender's subnetwork router, we can collect multicast forwarding statistics and determine when that router forwards a packet downstream. Netstat is a standard tool which displays information regarding a networking subsystem.

Since, we did not have sufficient privileges to monitor the sender's subnetwork router when the sender was running on a host machine outside UMass and also when R2 belonged to an ISP, we could not determine $S_2$.

Also, unfortunately, due to tunneling, one cannot accurately measure the delay between the routers and the end points. So, in the rest of the report, the join latency refers to the join latency $S_1$, as in the first definition. Also, the terms graft and join will be used interchangeably.

According to the first definition, the join latency essentially consists of delays incurred on the slow path, the waiting time at R2, and delays incurred on the fast path. The waiting time at R2 is incurred when the graft reaches R2 and R2 has to wait for a packet from the upstream router towards the sender, before R2 can forward that packet downstream. This waiting time depends upon the sender's rate. If the sender is transmitting at a high rate, this waiting time would be less and if the sender is transmitting at a slow rate, the waiting time would be high.

## 5 Experimental Setup

The experimentation process involved the following steps:

- Implementing a multicast sender
- Implementing a multicast receiver
- Collecting the join latency data

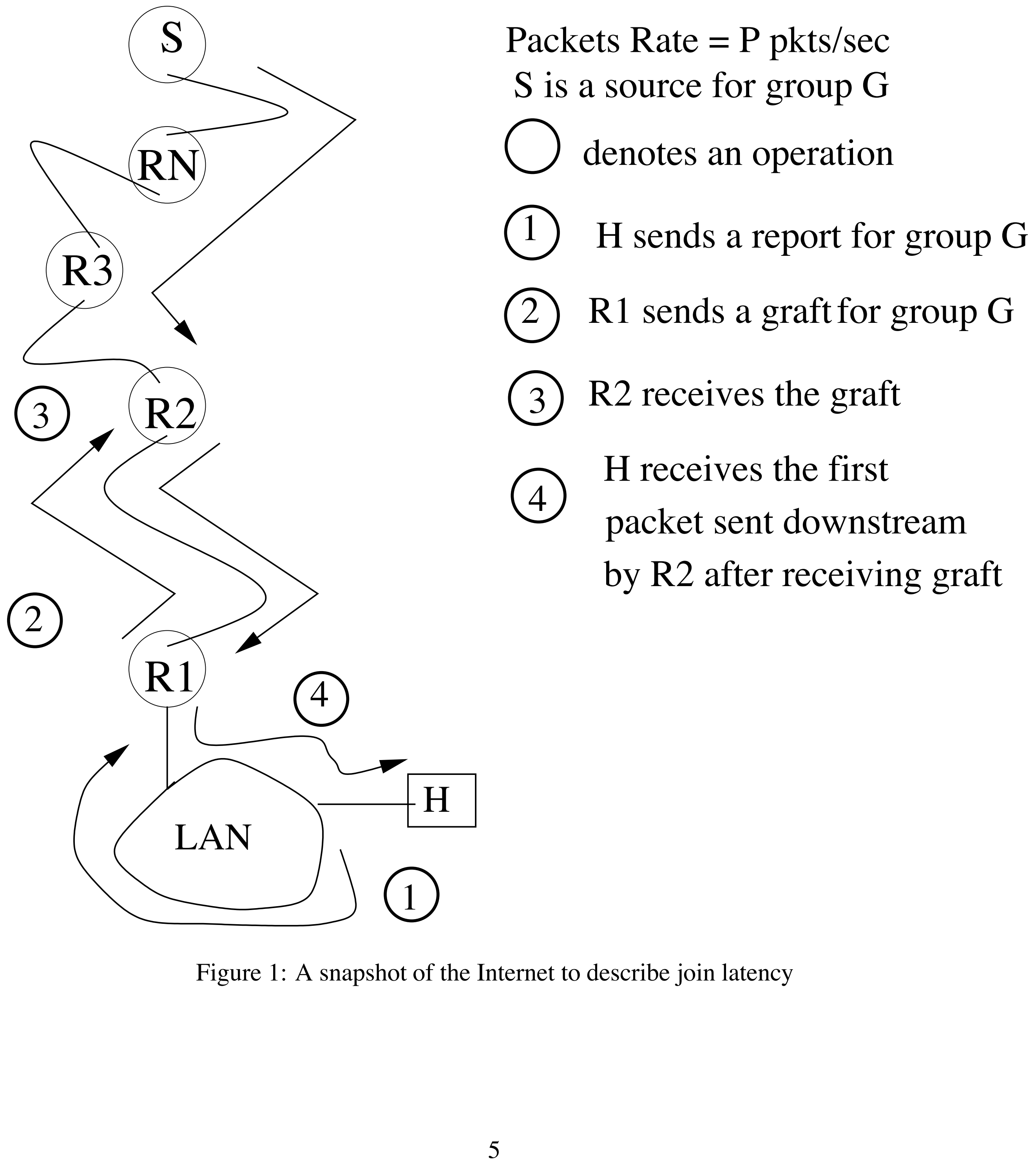

S
RN
R3
R2
R1
LAN
H
3
2
4
1
Packets Rate = P pkts/sec
S is a source for group G
denotes an operation
1 H sends a report for group G
2 R1 sends a graft for group G
3 R2 receives the graft
4 H receives the first packet sent downstream by R2 after receiving graft

Figure 1: A snapshot of the Internet to describe join latency

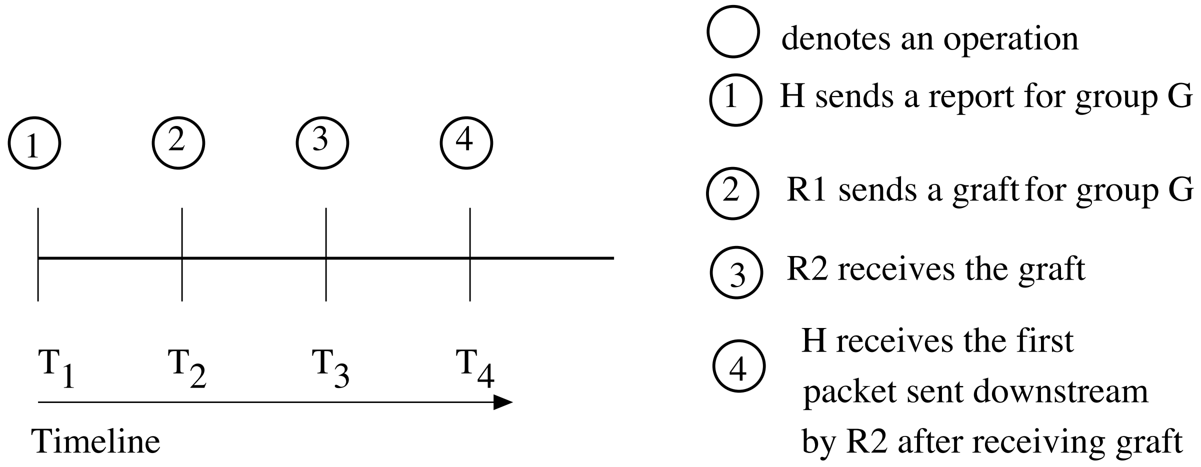

denotes an operation
1 H sends a report for group G
2 R1 sends a graft for group G
3 R2 receives the graft
4 H receives the first packet sent downstream by R2 after receiving graft
1
2
3
4
T1
T2
T3
T4
Timeline

Figure 2: Timeline of operations in a join latency experiment

### 5.1 Multicast sender

We used the multicast options of Unix sockets to implement the multicast sender and receiver. The command line parameters of the sender which could be varied for our experiments are:

- The period ($p$) between packets transmission by the sender
- The multicast group, TTL and the port number to which the sender sends packets

We start a sender which sends out packets to a particular multicast group and a particular port at a particular period $p$. The multicast group and port number were obtained using sdr. Sdr is a standard utility for obtaining unique multicast group and port numbers for a multicast session. Once the sender begins to execute, it keeps sending packets to the specified group, until the sender process is explicitly killed.

For our experiments, the period $p$ between packets transmission is 5 seconds. The usage of 5 seconds as inter-packet time ensures that the interval is long enough so that join latency values are not affected by transient network conditions in the MBone and also that not a lot of bandwidth is consumed on the MBone.

### 5.2 Multicast receiver

The command line parameters of the receiver which can be varied are:

- The multicast group which the receiver joins
- The port number at which it receives packets addressed to the group
- The file in which the receiver collects the join latency data

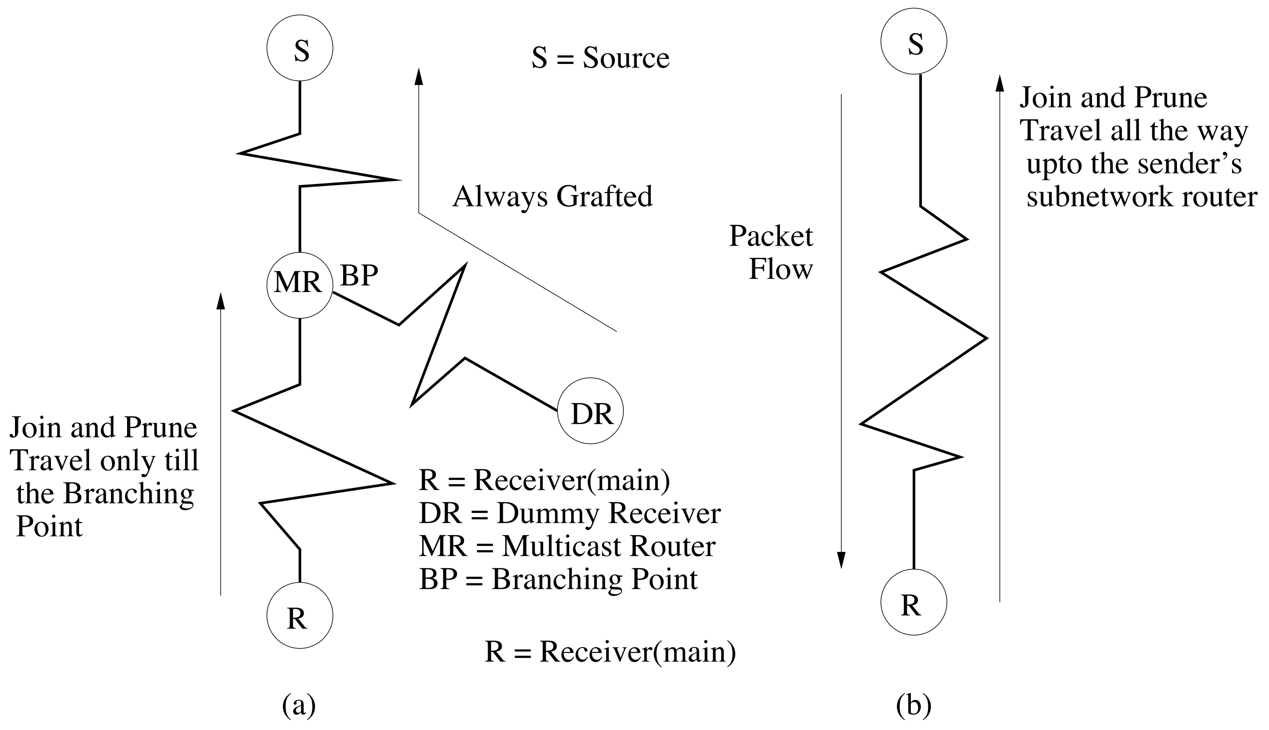


Figure 3: Experiment Description

## 5.3 Experimental Methodology

Consider Figure 3. Our experiments consisted of a single sender sending to a multicast group G and a single receiver receiving packets addressed to group G. Whenever a receiver joins/leaves the group, the graft/prune travel all the way to a multicast router upstream in the distribution tree. There are two possibilities according to whether there are any additional members of the group. Figure 3(a) illustrates the case, where a dummy receiver has not sent a prune to the branching point and is always a member of the group G. As a result of which, the sender's traffic always comes up to this branching point. Figure 3(b) illustrates the case, when there is no additional member of the group.

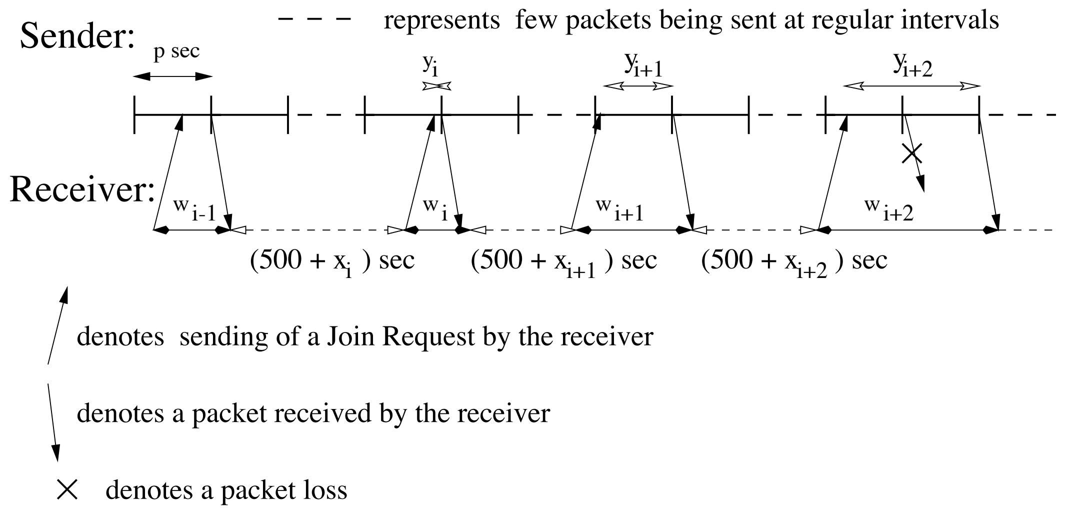
For all the experiments we conducted, we measured the join latency values between a receiver and a branching point on the distribution tree (Figure 3(a)). The branching point for the experiment was determined by running mtrace with the sender and receiver host names as arguments after every ten minutes and examining its output.

Mtrace is a standard tool which reports the reverse multicast path, if present, from a multicast host to another multicast host and traffic information between them. If a host is an active receiver, mtrace output shows the reverse multicast path from a sender to the host. If a host is not an active receiver and the prune message sent by the host's subnetwork router is still cached, then the mtrace output displays a reverse path between the source and the host. The mtrace output for a particular source/receiver pair and a multicast group, shows sender's traffic come how far downstream in the distribution tree towards the receiver and prune message, if sent, propagates how far upstream.

Figure 3(b) illustrates the case whenever a receiver joins or leaves the group, the graft and prune travel all the way to the multicast router attached to the source if there is no other member of the group outside the sender's subnetwork. For all our experiments, since there was only one receiver, we should have observed the prune/graft travel up to the multicast router attached to the source in a mtrace output. Instead, we noticed that the prune travelled only few hops towards the source. This is possibly due to either a version of mrouted running on a intermediate multicast router which does not support pruning or that the intermediate router has not received a prune on all the interfaces downstream.

A source is transmitting packets with a period $p$. After a receiver receives a packet, it records one sample of the join latency value and leaves the group. It then waits for some sufficient amount of time so that a prune is sent upstream by the receiver's multicast router and then the receiver joins the group again, waits for a packet, records another sample of join latency value and the whole process is repeated. We chose the waiting time before a receiver sends the next join request to be a uniformly distributed value between 500 sec to (500 + $p$) sec. In our case, the inter-packet time ($p$) was always 5 sec. The minimum waiting time of 500 sec. was chosen to ensure that the prune messages are actually generated and sent upstream. The use of 500 sec. as the minimum waiting time is justified in the "Problems Faced" section.

We added a random value between 0 and $p$ to the minimum waiting time so that the receiver sends a join request anytime between 0 and $p$ inclusive with respect to the sender's sending out a packet. Consider Figure 4. The sender keeps sending out packets at regular intervals of $p$ sec. The receiver issues a Join request anytime during an interval and starts a timer. The timer is stopped only when a packet is received by the receiver. This Join request results in a graft



For a measured sample value $w_i$ :

$x_i$ is a uniform random value between 0 and p sec.

$y_i$ is the waiting time incurred at the sender's router

$w_i = y_i$ + delays incurred on the reverse path and forward path

Figure 4: A snapshot of the experiment

message being generated. This graft message travels upstream up to the nearest branching point. Figure 4 corresponds to the case that the sender's subnetwork router is the branching point. The branching point then waits for the next packet from the source before forwarding it downstream. Since, we don't have any control at the branching point, we have no way of determining when the graft is received at the branching point and, exactly, how much time is there before the next packet is received by the branching point. As shown in Figure 4, we are unable to determine the exact value of $y$ (the waiting time) for a measured join latency value.

If the next packet forwarded downstream by the branching point is lost, the receiver's timer is still on and stops only when it receives the next packet. Thus, a measured join latency value may also include delays incurred due to the lost packets.

Assuming the next packet sent by the sender's router on receiving a graft always makes it to the receiver, the waiting period is then a uniform random value between 0 and inter-packet time and the measured join latency value would consist of delay on the slow path, a uniform random value between 0 and inter-packet time inclusive due to the wait at the branching point and delay on the fast path. As shown in Figure 4, if the next packet is lost and the subsequent packet makes it to the receiver, then the waiting period consists of a random value between 0 and $p$ plus $p$. Also, a sample of measured join latency value would now consist of delays on the slow path and the fast path and waiting time equal to a random value between 0 and inter-packet time plus an integral number of inter-packet times (which accounts for integral number of consecutive packet losses).

Of all the samples of the join latency values we collected, we ignore the samples with values greater than 6.5 seconds. We don't know whether the samples with values greater than 6.5 seconds were due to packet losses or congestion in the network. It may be possible to infer losses from the measured join latency values. However, this is topic for future investigation.

We first calculated the mean measured join latency value by taking all the measured join latency values less than or equal to 6.5 seconds. We subtracted the mean waiting time from the mean measured join latency value in order to obtain the mean join latency value which consists of average delays on the slow path and the fast path only.

The usage of 500 sec. as the minimum waiting time before another join latency value can be measured makes it difficult to generate a large number of samples and introduced the problem of less join latency values being measured. We had to let our experiments run for a week or so in order to collect 1000 join latency value samples. Care was taken to ensure that the routes between a sender and a receiver remain the same while the experiment was running. This was done by running mtrace as a separate process with the sender and receiver host names and group address and a TTL (Time-To-Live) value as arguments after every ten minutes.

## 6 Interoperability problems between IGMPv1 and IGMPv2 and its solution

When a router runs IGMPv1 or a IGMPv2 router is on the same subnetwork with a IGMPv1 router, then Leave Group messages of IGMPv2 are ignored by the router. Essentially IGMPv2 router behaves like a IGMPv1 router; the IGMPv2 router should time out as a IGMPv1 router before it can detect that there is no local group member.

When a IGMPv1 host is a member of a group, an IGMPv2 router ignores any Leave Group messages for that particular group and then group membership for the group is primarily determined by the response to the General Query messages of IGMPv2. If an IGMPv1 host was the last member to leave the group, an IGMPv2 router can detect that there is no local group member only when it times out after the Group Membership Interval has passed. Since, the Group Membership Interval is equal to 260 seconds by default, the Leave latency for IGMPv2 has now increased significantly compared to the Leave latency for IGMPv1 and the very purpose of IGMPv2 to reduce leave latency is defeated. The problem is further exacerbated if an IGMPv2 implementation after detecting the presence of an IGMPv1 host on the subnetwork, determines group membership for all the groups as if the IGMPv1 host is a member of all of them. In this pathological case, the IGMPv2 protocol behaves like a IGMPv1 protocol only with much higher leave latencies.

A solution to the above problem of increased leave latency would be to upgrade all the IGMPv1 hosts on the subnetwork to IGMPv2. This would ensure that the group membership is strictly determined according to IGMPv2.

A second solution requires a modification to IGMPv2 executing on the router so that it determines group membership as follows:

When a router receives an IGMPv1 Report message, it starts a v1 host timer and also stores the IP address of the v1 host. A running v1 host timer implies that there is a IGMPv1 host group member. An IGMPv1 Report message will be received when an IGMPv1 host joins the group or an IGMPv1 host responds to a General Query message. If IGMPv1 Report messages are received for the same group from different hosts, the IP addresses of all the IGMPv1 host members are also stored in a list. This ensures that the router knows the IP addresses of all the IGMPv1 host group members.

When a IGMPv2 Leave Group message is received for the group and a v1 host timer is running, the router does not ignore the Leave Group message unlike an unmodified IGMPv2 router which ignores the Leave Group message. It then sends a IGMPv2 Group Specific Query with a maximum response time of 1 sec and also a IGMPv1 General Query message with a maximum time response time of 10 sec encapsulated as a unicast message to all the IGMPv1 host group members using the list of IP addresses of the IGMPv1 host members. If no IGMPv1/v2 Reports are received, the router sends the Query messages again till the count of both IGMPv1 and IGMPv2 Query messages is equal to the value of the Robustness variable. If no report is received for all these messages, the router assumes that there are no IGMPv1/v2 host group members.

By sending a IGMPv1 General Query message encapsulated as a unicast message to a IGMPv1 host, we get the IGMPv1 reports from that host only. If we don't do this encapsulation, we will trigger IGMPv1 report messages from all the other IGMPv1 hosts also who haven't joined the group. If the number of IGMPv1 hosts who have joined the group is less than the total number of IGMPv1 hosts on the subnetwork, then unicasting the IGMPv1 Query message saves bandwidth. Otherwise, we can send a normal multicast Query message to all the hosts.

## 7 Problems Faced and Findings

One of the problems we faced was how to ensure that a prune is generated every time a receiver leaves the group by the receiver's subnetwork router and a graft is sent upstream to restore the path when the receiver joins the group again in order to measure the next join latency value. This was further complicated because of the way in which different versions of IGMP (IGMPv1 and IGMPv2) running on a host/router within the same subnetwork interact with each other.

In order to find out how a multicast router determines that there is no member of the group present on the subnetwork after a last member leaves the group, we had to first find out which version of IGMP was being used on the host/router. Also, we had to ascertain the exact number of query messages in IGMPv1 and general query messages in IGMPv2 are sent before a router assumes the absence of a local group member. One can determine which version of IGMP is running on a host/router by running tcpdump and taking a look at the IGMP messages. The version of IGMP running can also be determined by taking a look at the header files. The authors didn't have sufficient privileges to run tcpdump on host machines or take a look at the system files on all the host machines used for the experiments. By using netstat -C menu option 8 at a router repeatedly, one can determine how long it takes before a router assumes the absence of a local group member. Again, we didn't have access or enough permissions to use netstat options on all the routers attached to the host machines.

Since we could not ascertain for sure which version of IGMP was running on a host machine and the attached router, we had to figure out a waiting time before a receiver sends the next join request such that the router would have sent a prune already and now send a graft message upstream independent of the IGMP versions being used on the host/router.

As mentioned in the previous section, with an IGMPv1 host and IGMPv2 router on the same subnetwork, the router by default takes 260 seconds (Group Membership Interval) to decide that there is no group member on the subnetwork when a last member leaves a group. If the value of Robustness Variable is 3, then the Group Membership Interval is 385 seconds. Since we didn't know what was the value of the Robustness Variable for all the routers attached to the end hosts in our experiments, we chose a value of 500 seconds as the minimum waiting time.

Choosing a waiting time of at least 500sec after a member leaves a group and before it sends a Join request, ensured that prunes/grafts were actually generated and sent upstream. We might have collected few more samples in the same time period at few sites, had we chosen a smaller value, say 300 sec. But, in order to maintain uniformity, we stuck with 500sec.

When there is no local group member, the mtrace output with a local host as the destination parameter shows the prune message actually being generated and also how far it travels. By taking a look at such a mtrace output, one can identify whether pruning is taking place or not in the MBone. If its not, then which routers are not pruning properly can be identified.

At times, the sender had to be restarted when the sender process stopped running on a host machine outside authors' controls, ie, when the host machine was rebooted. The presence of the same routes between the sender and the receiver after a sender/receiver was restarted ensured the validity of the join latency values. Also, sometimes the Mbone was down or there was a loop between a sender and a receiver or no reverse path from the receiver to the sender could be found.

## 8 Results

In this section, we present the results of our experiments. In Section 8.1, we present the measured mean Join Latency values. Sample Cumulative Frequency and Frequency Distribution graphs are found in Sections 8.2 and 8.3 respectively. A sample mtrace output is presented in Section 8.4 .

### 8.1 Observations:

For these experiments, hosts `Eraser` and `Sahir` are located at UMASS (eraser.cs.umass.edu and sahir.cs.umass.edu). Hosts `Conviction` and `Bagpipe` are located at UCBerkeley (conviction.cs.berkeley.edu) and UKY (bagpipe.dcs.uky.edu) respectively.

A sender and receiver were started and samples of join latency values were collected at the receiver. A mtrace output was collected between the source and the receiver every ten minutes. All the observations are listed in the table below. Of the total samples (column "Total" in the table) collected, only those samples whose value was less or equal to 6.5 seconds (column "Used" Samples) were used in calculating the final Mean Join Latency values (Mean JL). The reason we chose the limit to be 6.5 seconds will be given in the next subsection. The Measured Mean Join Latency value for the experiment and its variance is also given. A Mean JL value is calculated by subtracting mean waiting time which is 2.5 sec in our case, from the measured Mean JL value.

The total number of hops between the source and the destination are listed in the column (Total Hops). This includes a hop from the sender to the sender's router and a hop from the receiver's router to the receiver. The number of hops between the multicast router attached to the receiver subnetwork and the branching point is listed in the column "Pruned" Hops. The branching point for an experiment was determined by taking a look at the mtrace output.

| #Samples | | Source | Destination | Hops | | Mean JL | Measured Mean JL | Variance |
|---|---|---|---|---|---|---|---|---|
| Total | Used | | | Total | Pruned | | | |
| 1042 | 1026 | Conviction | Eraser | 16 | 2 | 0.61709 | 3.11709 | 2.28731 |
| 833 | 703 | Bagpipe | Eraser | 9 | 2 | 0.84408 | 3.34408 | 2.44741 |
| 561 | 517 | Bagpipe | Sahir | 9 | 2 | 0.68579 | 3.18579 | 2.40611 |
| 336 | 311 | Bagpipe | Eraser | 9 | 2 | 0.69376 | 3.19376 | 2.50775 |
| 312 | 290 | Conviction | Eraser | 16 | 2 | 0.68285 | 3.18285 | 2.46441 |

Figure 5: Results: Measured Mean Join Latency Values

All the mean Join Latency values and the measured Mean JL values are in seconds. The row entries with the same source and destination pair denote that experiments were conducted between them at different times.

The average of mean JL values in the table is 0.704714 seconds for two Pruned Hops and the average of mean JL value per hop is 0.352357 seconds. It is evident from the row entries that the mean Join Latency value between any hosts is also dependent upon the network conditions. It is not clear to the authors whether higher Mean Join latency values can be attributed either to congestion in the network or packet losses in the network and this demands further investigation.

The author's speculation for the high mean Join Latency values between Bagpipe and Eraser is discussed in the next two subsections.

It is clear from the column "Pruned" Hops that not all the routers in the current Mbone are pruning properly. Otherwise, ideally the authors would have expected to see an entry in the column "Pruned" Hops to be equal to the corresponding "Total" Hops entry minus two (one hop for the source to the source subnetwork router and another for the receiver to the receiver's subnetwork router.)

## 8.2 Cumulative Frequency Graphs

The Cumulative Frequency Graph for the experiment with `Conviction` as the sender and `Eraser` as the receiver (Total Samples = 1042) is drawn in Figure 6. The measured Join

Latency value (in seconds) is on the $x$-axis and the cumulative frequency is on the $y$-axis. Since a line through the steep slope of the curve tapers towards the origin, this shows that there is non-negligible mean join latency value and its value can be interpolated by the point where the above line meets the $x$-axis. If the mean join latency value would have been zero, the line would have passed through the origin and the initial steep slope of the curve would have lasted till the point with value 5.0 on the $x$-axis. Thus, initial portion of the curve corresponds to the mean join latency.

The Cumulative Frequency Graph for the experiment with `Bagpipe` as the sender and `Eraser` as the receiver (Total Samples = 833) is drawn in Figure 7. The curve consists of three regions - the initial steep slope, the middle portion with a lower slope between 6 sec and 11 sec and the third portion with even less steep slope between 11 sec and 15 sec. The authors speculate that the middle portion is due to one packet loss and the third portion is due to two consecutive packet losses and the observed high mean Join Latency values for the experiment is due to the observed packet losses. This demands further investigation.

Also, the point on the curve in Figure 6 till which the steep slope lasts has an approximate value of 6.5 on the $x$-axis. The point till which the initial steep slope of the curve in Figure 7 lasts has an approximate value of 6.5 on the $x$-axis. This initial portion of the curve corresponds to the mean join latency. And this is the reason why we included only those samples whose value was less than or equal to 6.5 seconds in calculating measured Mean Join Latency values.

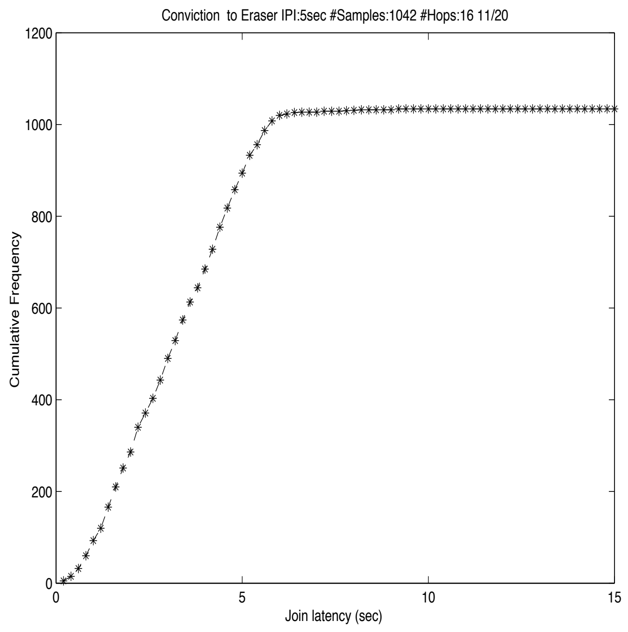


Figure 6: Cumulative Frequency Graph from Conviction to Eraser

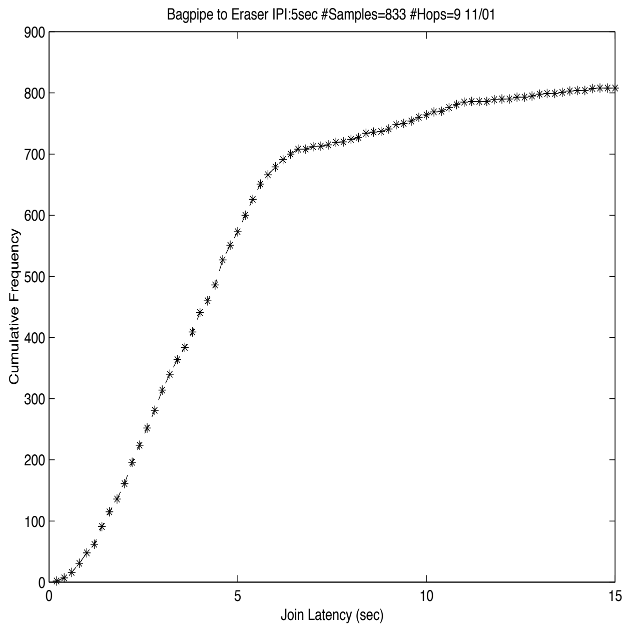


Figure 7: Cumulative Frequency Graph from Bagpipe to Eraser

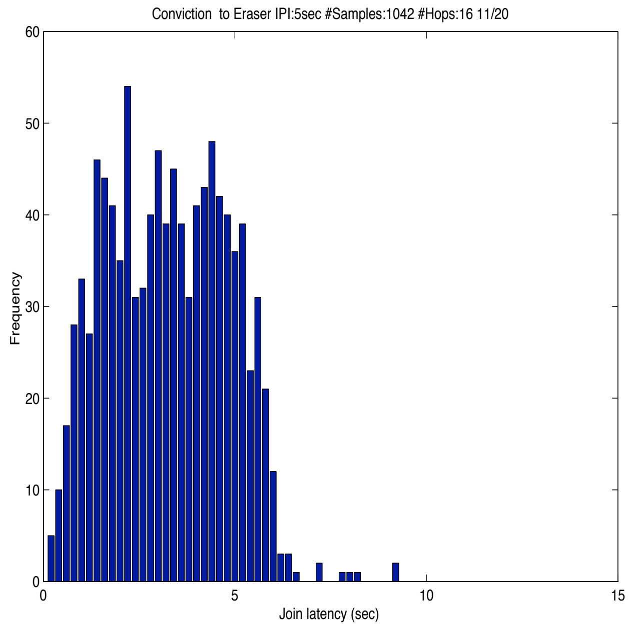


Figure 8: Frequency Distribution Graph from Conviction to Eraser

## 8.3 Frequency Distribution Graphs

Figures 8 and 9 are the frequency distribution graphs for the experiments from `Conviction` to `Eraser` (Total Samples = 1042) and from `Bagpipe` to `Eraser` (Total Samples = 833)

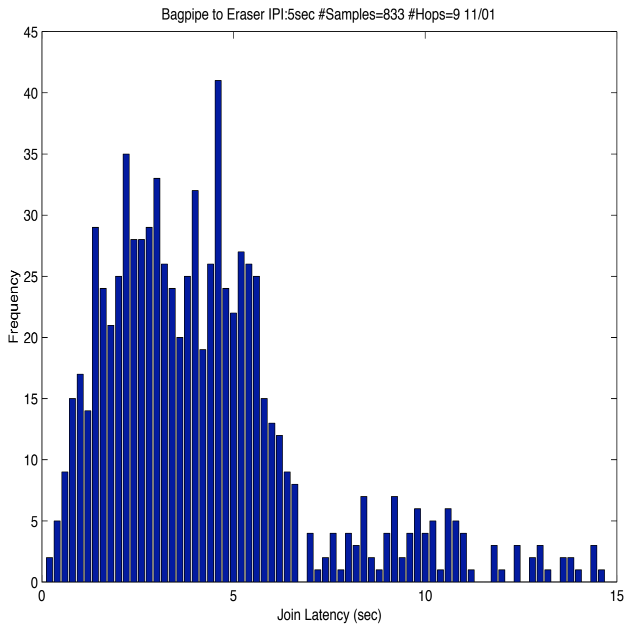


Figure 9: Frequency Distribution Graph from Bagpipe to Eraser

respectively. The measured Join Latency in seconds is on the $x$-axis and the frequency is on the $y$-axis.

Since the frequency of measured Join Latency values between 6 sec and 15 sec is non-zero in Figure 9, the authors speculate that this tail of the distribution is attributed to the packet losses and this demands further investigation.

## 8.4 A sample mtrace output

The sample mtrace output below was collected at the time of the experiment from `Conviction` to `Eraser` (Total Samples = 1042). From the mtrace output it is clear that a prune was sent upstream from the receiver's subnetwork router and it propagated only two hops upstream and this is the branching point for the experiment. The sender's traffic always flows upto this branching point and at this point the output is pruned and data propagates no further towards the receiver.

This also shows that pruning is not taking place in the current Mbone as expected. Otherwise, the mtrace output would have shown that the prune propagates upstream till the sender's subnetwork router. Similar mtrace outputs were collected for other experiments and all the mtrace outputs collected reflect the same bad pruning.

```
Mtrace from 128.32.33.103 to 128.119.41.176 via group 224.2.239.142
Querying full reverse path...
  0  eraser (128.119.41.176)
 -1  erlang (128.119.40.203)  DVMRP  thresh^ 1  Prune sent upstream
 -2  spare-gw.gw.umass.edu (128.119.2.9)  Unknown protocol code 5  thresh^ 32
 Prune sent upstream
 -3  dec3800-1-fddi-1.WestOrange.cw.net (204.70.64.45)  DVMRP  thresh^ 64
Output pruned
 -4  dec3800-1-fddi-0.Washington.cw.net (204.70.2.13)  DVMRP  thresh^ 1
 -5  dec3800-2-fddi-0.Washington.cw.net (204.70.74.61)  DVMRP  thresh^ 1
 -6  ? (204.70.176.23)  DVMRP  thresh^ 1
 -7  cs.res.vbns.net (204.147.128.189)  Unknown protocol code 8  thresh^ 0
 -8  cs-atm0-0-101.psc.vbns.net (204.147.130.242)  Unknown protocol code 8
 thresh^ 0
 -9  cs-atm0-0-8.hay.vbns.net (204.147.130.38)  Unknown protocol code 8
thresh^ 0
-10  BERK-vBNS.Calren2.net (198.32.251.2)  Unknown protocol code 6  thresh^ 0
-11  pos1-0.inr-000-eva.Berkeley.EDU (128.32.0.89)  PIM  thresh^ 0
-12  pos6-0-0.inr-002-eva.Berkeley.EDU (128.32.0.78)  Unknown protocol code 8
 thresh^ 0
-13  fast4-0-0.inr-666-eva.Berkeley.EDU (128.32.0.83)  Unknown protocol code 5
  thresh^ 0
-14  f1-0-0.inr-107-eva.Berkeley.EDU (128.32.2.1)  Unknown protocol code 5
 thresh^ 0
-15  f1-0.inr-180-soda.Berkeley.EDU (128.32.120.180)  PIM  thresh^ 0
-16  conviction.CS.Berkeley.EDU (128.32.33.103)
Round trip time 1165 ms

Waiting to accumulate statistics... Results after 10 seconds:

  Source        Response Dest    Packet Statistics For     Only For Traffic
128.32.33.103   128.119.41.176   All Multicast Traffic     From 128.32.33.103
     v       __/  rtt  960 ms    Lost/Sent = Pct  Rate       To 224.2.239.142
128.32.33.1
128.32.120.180  f1-0.inr-180-soda.Berkeley.EDU
     v     ^      ttl    0     -458/1173 =-38% 130 pps     0/2    = --%    0 pps
128.32.120.107
128.32.2.1      f1-0-0.inr-107-eva.Berkeley.EDU
     v     ^      ttl    1       40/1633 =  2% 181 pps     0/2    = --%    0 pps
128.32.2.2
128.32.0.83     fast4-0-0.inr-666-eva.Berkeley.EDU
     v     ^      ttl    2      -39/1593 = -1% 177 pps     0/2    = --%    0 pps
128.32.0.82
128.32.0.78     pos6-0-0.inr-002-eva.Berkeley.EDU
     v     ^      ttl    3       10/1627 =  1% 180 pps     0/2    = --%    0 pps
128.32.0.77
128.32.0.89     pos1-0.inr-000-eva.Berkeley.EDU
     v     ^      ttl    4        0/1617 =  0% 179 pps     0/2    = --%    0 pps
128.32.0.90
198.32.251.2    BERK-vBNS.Calren2.net
     v     ^      ttl    5        0/1617 =  0% 179 pps     0/2    = --%    0 pps
198.32.251.1
204.147.130.38  cs-atm0-0-8.hay.vbns.net
     v     ^      ttl    6     2686/4309 = 62% 478 pps     0/2    = --%    0 pps
204.147.130.37
```

```
204.147.130.242 cs-atm0-0-101.psc.vbns.net
     v     ^      ttl    7      219/9824 =  2%1091 pps     0/2    = --%   0 pps
204.147.130.241
204.147.128.189 cs.res.vbns.net
     v     ^      ttl    8      142/7223 =  2% 802 pps     0/2    = --%   0 pps
204.70.176.23   ?
     v     ^      ttl    9        5/7081 =  0% 786 pps     0/2    = --%   0 pps
204.70.74.77
204.70.74.61    dec3800-2-fddi-0.Washington.cw.net
     v     ^      ttl   10        5/8122 =  0% 902 pps     0/2    = --%   0 pps
204.70.2.13     dec3800-1-fddi-0.Washington.cw.net
     v     ^      ttl   11        1/8542 =  0% 949 pps     0/2    = --%   0 pps
204.70.64.45    dec3800-1-fddi-1.WestOrange.cw.net Output pruned
     v     ^      ttl   64      190/1935 = 10% 215 pps     2/2    = --%   0 pps
128.119.2.9     spare-gw.gw.umass.edu Prune sent upstream
     v     ^      ttl   65        0/1318 =  0% 146 pps     0/0    = --%   0 pps
128.119.40.203  erlang         Prune sent upstream
     v      \__   ttl   66          2            0 pps       0            0 pps
128.119.41.176  128.119.41.176
  Receiver      Query Source
```

Another similar sample mtrace output for the experiment from Bagpipe to Eraser (Total Samples = 833) is given below.

```
Mtrace from 204.198.75.113 to 128.119.41.176 via group 224.2.235.94
Querying full reverse path...
  0  eraser (128.119.41.176)
 -1  erlang (128.119.40.203)  DVMRP  thresh^ 1  Prune sent upstream
 -2  spare-gw.gw.umass.edu (128.119.2.9)  Unknown protocol code 5  thresh^ 32
Prune sent upstream
 -3  dec3800-1-fddi-1.WestOrange.cw.net (204.70.64.45)  DVMRP  thresh^ 64
Output pruned
 -4  dec3800-2-fddi-1.WestOrange.cw.net (204.70.64.77)  DVMRP  thresh^ 1
 -5  e1.cambridge1-mbone1.bbnplanet.net (199.94.207.2)  Unknown protocol code 5
thresh^ 32
 -6  l0.paloalto-mbone1.bbnplanet.net (131.119.1.38)  Unknown protocol code 5
thresh^ 32
 -7  f0.atlanta1-mbone1.bbnplanet.net (4.0.35.20)  Unknown protocol code 5
thresh^ 64
 -8  santoor-tbt.dcs.uky.edu (204.198.76.137)  DVMRP  thresh^ 32
 -9  bagpipe.dcs.uky.edu (204.198.75.113)
Round trip time 894 ms

Waiting to accumulate statistics... Results after 10 seconds:

  Source        Response Dest    Packet Statistics For     Only For Traffic
204.198.75.113  128.119.41.176   All Multicast Traffic     From 204.198.75.113
     v       __/  rtt  976 ms    Lost/Sent = Pct  Rate       To 224.2.235.94
204.198.75.137
```

```
204.198.76.137  santoor-tbt.dcs.uky.edu
     v     ^      ttl   32        0/2    = --%   0 pps     0/2    = --%   0 pps
4.0.35.20       f0.atlanta1-mbone1.bbnplanet.net
     v     ^      ttl   64        0/2    = --%   0 pps     0/2    = --%   0 pps
131.119.0.197
131.119.1.38    l0.paloalto-mbone1.bbnplanet.net
     v     ^      ttl   65       40/70   = 57%   7 pps     0/2    = --%   0 pps
4.0.4.68
199.94.207.2    e1.cambridge1-mbone1.bbnplanet.net
     v     ^      ttl   66       11/1397 =  1% 139 pps     0/2    = --%   0 pps
204.70.64.61
204.70.64.77    dec3800-2-fddi-1.WestOrange.cw.net
     v     ^      ttl   67        0/678  =  0%  67 pps     0/2    = --%   0 pps
204.70.64.45    dec3800-1-fddi-1.WestOrange.cw.net Output pruned
     v     ^      ttl   68       -2/731  =  0%  73 pps     2/2    = --%   0 pps
128.119.2.9     spare-gw.gw.umass.edu Prune sent upstream
     v     ^      ttl   69       -1/673  =  0%  67 pps     0/0    = --%   0 pps
128.119.40.203  erlang         Prune sent upstream
     v      \__   ttl   70          6            0 pps       0            0 pps
128.119.41.176  128.119.41.176
  Receiver      Query Source
```

## 9 Conclusions

Though the data needs further analysis, we have drawn the following conclusions from the graphs that we plotted and the values of the mean join latency values that we calculated.

- The mean join latency values are significant and it is expected that services and protocols using multiple multicast groups will benefit from these actual values.
- The data strongly reflects that pruning on the current Mbone does not happen properly. This can be very disastrous as far as the wastage of bandwidth is concerned, since most of the multicast routers are connected through tunnels which consist of many unicast routers.
- The existing IGMPv2 protocol fails to take into account all the interoperability scenarios with IGMPv1 protocol. A solution has been proposed which requires modifications to IGMPv2 on the router's side only.

## 10 Future Work

The following are the items we have identified as future work.

- Collect the join latency data at other receivers and more extensively. In this report, the sender's rate was fixed at 5 sec. We would like to see the effect of sender's rate on the mean join latency values.

- Of all the measured join latency values, samples with value greater than 6.5 sec were ignored and these demand more investigation. The inference of packet losses or network congestion from these values also demands further investigation.
- If a new receiver joins a group, its join latency value would depend upon how many hops it is far away from the branching point of the distribution tree. The join latency value should be directly proportional to the number of hops. The authors initially wanted to compute the join latency for a receiver as a function of the number of hops but could not do so due to bad pruning in the current network. In all the experiments, the number of hops between a receiver's router and the corresponding branching point was 2. More join latency values for more hops between a receiver and a branching point should be collected and analysed.
- The mean join latency values depend upon the multicast routing protocol being used even thought the process remains almost the same. We would like to measure the join latency values due to other multicast routing protocols and their interactions.
- The proposed modification to IGMPv2 should be implemented and deployed.

## References

[1] S. Deering. Host Extensions for IP Multicasting, Aug 1989. RFC 1112.

[2] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, A. Helmy, D. Meyer, and L. Wei. Protocol Independent Multicast Version 2 Dense Mode Specification, Jun 1999. draft-ietf-pim-v2-dm-03.txt.

[3] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C. Liu, and L. Wei. The PIM Architecture for Wide-Area Multicast Routing. *IEEE/ACM Transactions on Networking*, Apr 1996.

[4] D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, P. Sharma, and L. Wei. Protocol Independent Multicast - Sparse Mode (PIM-SM): Protocol Specification, Jun 1998. RFC 2362.

[5] W. Fenner. Internet Group Management Protocol, Version 2, Nov 1997. RFC 2236.

[6] L. Gao and D. Towsley. Supplying Instantaneous Video-on-Demand Services Using Controlled Multicast. *Proceedings of IEEE Multimedia Computing Systems*, Jun 1999.

[7] K.A. Hua, Y. Cai, and S. Sheu. Patching: A Multicast Technique for True for Video-on-Demand Services. *ACM Multimedia*, September 1998.

[8] K.A. Hua and S. Sheu. Skyscraper Broadcasting: A New Broadcasting Scheme for Metropolitan Video-on-Demand Systems. *Proceedings of ACM SIGCOMM*, September 1997.

[9] S. Kasera, J. Kurose, and D. Towsley. Scalable Reliable Multicast Using Multiple Multicast Groups . *Proceedings of ACM Sigmetrics*, June 1997.

[10] S. Mccanne and V. Jacobson. Receiver Driven Layered Multicast. *Proceedings of ACM Sigcomm*, Aug 1996.

[11] T. Pusateri. Distance Vector Multicast Routing Protocol, Feb 1999. draft-ietf-idmr-dvmrp-v3-08 Expires August 31, 1999.

[12] Luigi Rizzo. Fast group management in IGMP. *Proceedings of Hipparc*, 1998.

[13] L. Vicisano, L. Rizzo, and J. Crowcroft. TCP like Congestion Control for Layered Multicast. *Proceedings of IEEE Infocom*, Mar 1998.

[14] George Xylomenos and George C. Polyzos. IP Multicasting for Point-to-Point Local Distribution. *Proceedings of the IEEE Infocom*, 1997.