PERFORMANCE COMPARISON OF NATIVE MULTICAST VERSUS OVERLAY MULTICAST

Informatikprojekt
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Luca Bettosini
2008

Leiter der Arbeit:
Professor Dr. Torsten Braun

Betreuer der Arbeit:
Marc Brogle
Dragan Milic

Institut für Informatik und angewandte Mathematik

Contents

Chapter 1

Introduction

1.1 Multicast

Multicast is a communication paradigm for delivering data to a specific group of recipients simultaneously. The senders (usually one) need only to send one transmission stream to the network, from where the stream is distributed to the interested receivers [1]. This paradigm differs from the unicast and broadcast paradigm, where each transmission stream can only be transmitted to one or all recipients.
In the Internet the multicast paradigm has been implemented in the form of IP Multicast [2]. Although IP Multicast has been suggested and specified almost twenty years ago, it has never been widely deployed and used by the commercial Internet service providers (ISP). Some reasons for this are [1]:

- IP Multicast must be supported and enabled by all routers on the path from source to destination.
- Additional inter-ISP coordination is required (policy issues of inter-domain routing).
- IP Multicast routing can be very resource intensive.

But despite these problems IP Multicast is a powerful implementation of the multicast paradigm.

1.2 IP Multicast

IP Multicast is an extension of the Internet Protocol (IP) [3] to support multicasting. Instead of sending the IP datagram to one receiver using unicast, the datagram can be submitted to a set of zero or more receivers by only using one IP destination address. Such a multicast datagram has the same reliability as a regular unicast datagram, meaning that the arrival of the datagram at the destination is not guaranteed.
Hosts (receivers and senders) can join listening to a specific IP Multicast address. All hosts listening to the same IP Multicast address are called members of this address. This membership is dynamic, meaning that hosts can join and leave an IP Multicast group. Furthermore, they can be member of more than one group at a time and there is no restriction on how many users can join a multicast group nor where they are located in the network.
There are two types of multicast groups. There is the permanent group with a fixed assigned IP address (not a permanent membership). These groups are mostly used for administrative purposes and can have zero members. The other type of groups are transient groups. These groups only exist, when at least one host has joined the group. Both groups can be identified by their IP address. Multicast groups are using class D IP addresses, meaning that the multicast groups have ”1110” as their high-order four bits. In the ”dotted decimal” notation, multicast groups have a range between 224.0.0.0 and 239.255.255.255. Within this range there are several IP addresses, which are reserved for adminstration (routing protocols,

maintenance protocols, ...) or are reserved by the Internet Assigned Numbers Authority (IANA) [4].

For transmitting information within these groups, the forwarding of the IP Multicast datagrams between different networks is handled by specific routers with multicast availability. Within a local network a host transmits IP Multicast datagrams directly to all immediately-neighboring members of the designated multicast group as local multicast. Depending on the time-to-live (TTL) [3] value in the datagram, the multicast router forwards the datagram to all other connected networks, which are interested in this multicast group. Within these interested networks, the multicast router forwards the datagram as local multicast.

With the Internet Group Management Protocol (IGMP) [2] multicast routers are able to find out, which networks attached to them want to receive datagrams of which multicast group. At the moment there are three existing versions of IGMP: IGMPv1 [2], IGMPv2 [5] and IGMPv3 [6].

With IGMPv1 multicast routers send Host Membership Query messages to their attached networks. These queries are sent to the all-host group (address 224.0.0.1) with an IP TTL of 1. All hosts receiving this query respond to it with a Host Membership Report message for each multicast group, from which they want to receive data. To reduce the flood of reports generated by the hosts, two techniques are used:

1. After receiving a query, the host does not reply immediately by sending the reports. Instead the host starts a report delay timer with a different, randomly-chosen value between zero and X seconds, for each multicast group membership. After a timer expires the report for the corresponding group is generated and sent back to the originator of the query. With this technique the flush of reports is spread over an X second interval.

2. After receiving a query, the host does not send back the report to the originator. Instead the report is sent out to the multicast group address, to which the report belongs to, with a TTL of one. By doing this, other members of the group within the same network can overhear the report. After a host hears a report for a group, of which it is member of, the host stops its own timer for that group and does not generate a report for that group. With this technique the flush of reports will be reduced, normally the multicast router will only receive one report for every multicast group, which has members in that specific network.

The multicast router sends the queries periodically out to refresh its knowledge of memberships present within a network. If it does not receive a report for a particular group after some numbers of queries, the router can assume that no host within this network is member of that group and that it does not have to forward datagrams from that group to this network anymore (implicit leave).

With IGMPv2 [5] the problem of having two multicast routers attached to one network was solved. Depending on the IP address of a multicast router it can now act as a querier or non-querier. There exists normally only one querier in a physical network. All multicast routers start up as a querier on each attached network and after receiving query messages from another multicast router, the multicast router with the higher IP starts acting as a non-querier and stops sending query messages to this specific attached network. Besides the ability of a host to send Leave Messages (when a host wants to leave a specific multicast group) back to the querier was implemented in IGMPv2 to reduce network traffic. In addition, multicast routers can send two kinds of membership queries: a general query, to learn which groups have members on the attached networks, and a group-specific query to learn if a particular group has any member in the attached network.

In IGMPv3 [6] the support for ”source filtering” was added to IGMP. ”Source filtering” is the ability of a system to report interest in receiving only packets from a specific source address.

1.3 Overlay Multicast

Overlay multicast, also called end system multicast or application level multicast, was proposed as a new group communication paradigm in place of IP Multicast due to the deployment problem of native multicast. A virtual topology can be built to form an overlay network on top of the physical Internet. Each link in the virtual topology is a unicast tunnel in the physical network. The IP layer provides a best-effort unicast datagram service, while the overlay network implements all the multicast functionalities such as dynamic membership maintenance, packet duplication and multicast routing [7].

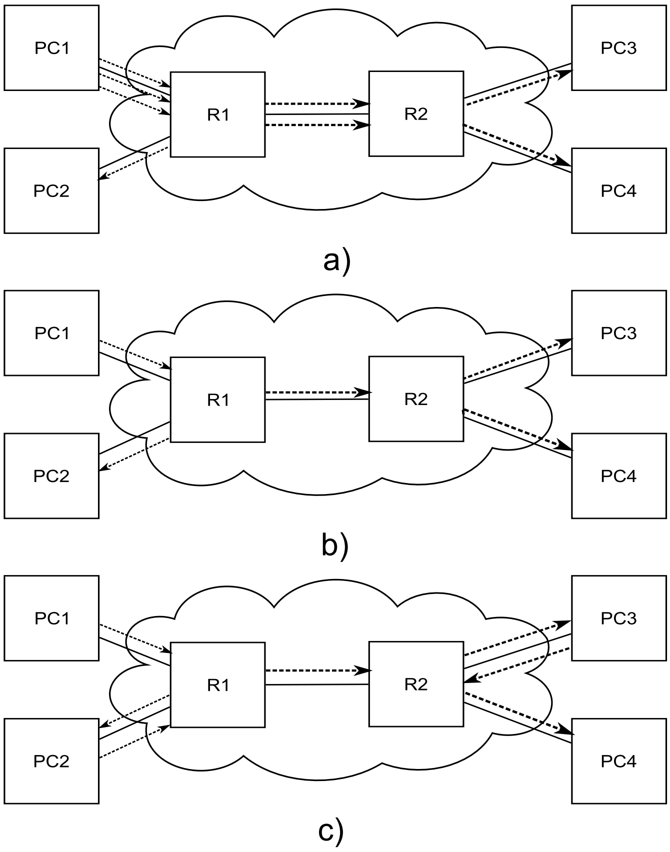

Figure 1.1: a) unicast b) native multicast c) overlay multicast

In Fig. 1.1 the main differences between unicast, native multicast and overlay multicast are shown. In Fig. 1.1 a) the host PC1 is sending packets using unicast to the tree receivers PC1, PC2 and PC3. The routers R1 and R2 are not required to support IP Multicast. However in Fig. 1.1 b) the host PC1 is sending the packets using IP Multicast, here the routers need to support the native multicast paradigm. Furthermore, a reduction of network traffic occurs, the host is sending the packets only once instead of tree times. In Fig 1.1 c) the overlay multicast paradigm is implemented in the receivers PC2, PC3, PC4 and in the host PC1 system. The host is sending the packets using multicast. Using overlay multicast, the packets are now sent over the virtual topology to the receivers using unicast tunnels in the physical networks. Therefore the routers are again not required to support multicast.
In the paper "Supporting IP Multicast Streaming Using Overlay Networks" [1] a performance evaluation

between the IP Multicast and overlay multicast was done. The authors performed real-time measurements with two different topologies, as shown in Fig. 1.2, measuring throughput and packet loss. Both topologies were chain topologies, one consisted of two and the other of five computers.

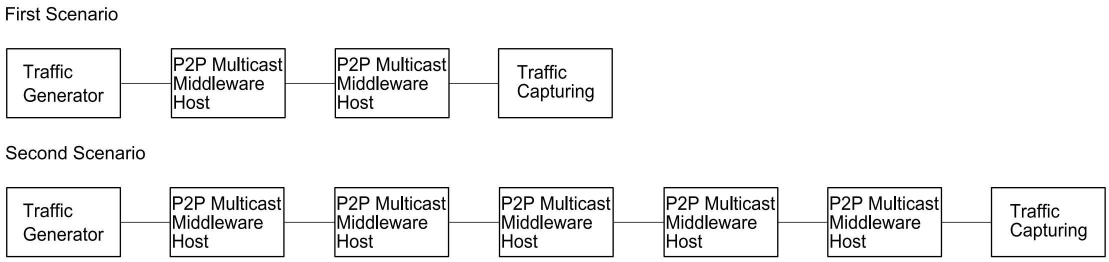

Figure 1.2: Scenarios for the Multicast Middleware performance evaluation

The packets were generated and captured using the MGEN traffic-generating tool. For each scenario a total of 24 packet flows with different sending rates, ranging from 11 to 241 Mbps in steps of 10 Mbps, were generated. Each packet consisted of 1 024 bytes payload and the packet flow was sent for 120 seconds.

The results showed that the packet loss for a bandwidth up to 100 Mbps was negligible for both scenarios. But over a 100 Mbps bandwidth the packet loss increased significantly. Overall the packet loss was less than 4% for a bandwidth up to 155 Mbps. Furthermore no significant difference between the packet loss for both scenarios was recognized and therefore the authors suggested that the impact of network scaling for the Multicast Middleware is minimal. It was also shown that an instance of the Multicast Middleware can deliver a maximum of 210 Mbps bandwidth and that the jitter increased with the number of peers involved in transporting the traffic.

1.4 Delay and Jitter

The common definition of a delay is the amount of time by which an event is deferred. In the network environment latency refers to the delay associated with the delivery of media stream data between two points in the switching fabric. Latency for a given path through a switching fabric is the sum of the following three types of delay that may be present in a given implementation [8]:

- **Transmission Delay**
  The Transmission delay refers to the delay introduced by the encoding, framing or packetizing of the media stream.

- **Propagation Delay**
  The Propagation delay refers to the delay associated with the propagation of the signal over the transmission facility used to implement the media stream channel. In this paper the propagation delay will not be an issue due the small network distances.

- **Processing Delay**
  Processing delay refers to any incremental delay introduced in a switching fabric for processing each packet in a packetized media stream. Sources of processing delay include time required for interpreting and updating headers, time required for determining how to route the packet and, most significantly, any buffering required prior to the forwarding of the packet. As we can see later, this delay plays a major part in our measurements.

- **Jitter**
  The Jitter refers to the variance in the latency described above. The latency associated with a given media stream channel may be constant (in which case there is no jitter) or it may change from moment to moment.

1.5 Purpose and Expectations

The purpose of this thesis is to compare the efficiency of native IP Multicast and Application Layer Overlay Multicast (ALM) ”packet forwarding”. In particular we compare the bandwidth, the delay and the jitter between native IP Multicast and ALM. We expect to show that an ALM is a valid and efficient solution to enable multicast on a given infrastructure.

1.6 Structure of the Thesis

This paper is structured as follows. In the next chapter we describe the setup of the test environment. Furthermore, a short explanation of the code used to control the network performance analysis system Smartbits is given. In chapter 3 the results are presented and commented. In the last chapter we summarize our results and give an overview of future work.

Chapter 2

Experiment Setup

To perform measurements for the comparison of native multicast and overlay multicast, a real time experiment environment was set up. This environment consisted of seven computers and a network performance analysis system called ”Smartbits”. The idea of this experiment was to create a specific multicast data stream with Smartbits, sending it to the network built with the computers and capturing the packets at the end of the network with Smarbits. With this setup we measured the delay and the packet loss for each packet in the data stream. We made measurements with different data streams varying in packet payload and network throughput.

2.1 Smartbits

Smartbits is a high-density network performance analysis test system, which consists of a chassis and multiple modules. For this experiment we used the SmartBits 600B chassis with the module ”SmartMetrics 10/100 Base-T Ethernet LAN-3101B”. This configuration of the Smartbits [9] had six 10/100 Mbit LAN ports on the front side of the device, to send or capture data. It can be controlled over another Ethernet port or a serial console, which are both located on the backside.
There are several ways how to work with a Smartbits device. Spirent Communicaton, the manufacturer of Smartbits, delivers several software packages with the device. For our experiment, the data analysis provided by this software was not sufficient for our needs, it only provides high level summary details. Another way to communicate with Smartbits is over an open-source script language ”tool command language (TCL)” or over the language C.
For our experiment we used TCL. Spirent provides a simple command library, in which every command for all Smartbits modules and chassis are written. Smartbits accepts all commands provided by this library without an error message, but delivers not always the expected result. Only a small set of the commands was working properly with our device.

2.2 Configuration of the Computers

The operating system on all computers was Fedora-Core 5. For a detailed hardware and software specification we refer to Appendix A.

2.2.1 Configuration for IP Multicast

To enable IP Multicast on the computers, we used the tool called SMCRoute [10]. SMCRoute is a command line tool to manipulate the multicast routes of the Linux kernel and allows us to maintain static IP Multicast routes. Generally multicast routes exist in the kernel only as long as smcroute is running. For simplification we used static routes.
To establish an IP Multicast network, the Network Interface Cards (NICs) were activated with a fixed IP address according to the network topology setup. Afterward the tool smcroute was started with the command *smcroute -d*. Finally the static IP Multicast routes were established using the command *smcroute -a <InputIntf> <OriginIpAdr> 224.1.2.3 <OutputIntf>*. These operations had to be done on every computer in the network topology.

2.2.2 Configuration for Overlay Multicast

To enable overlay multicast on the computers we used the Multicast Middleware package [11]. The Multicast Middleware enables the transparent use of ALM mechanisms. This is achieved by a virtual network interface (TAP) intercepting and forwarding IP Multicast packets to the Multicast Middleware [1]. The TAP mechanism is used by the Multicast Middleware to emulate an IP Multicast router attached to the Ethernet network. All IP Multicast traffic will be redirected to the Multicast Middleware entity on the end system, where it will be send to other end systems using a P2P ALM mechanism. Furthermore, the Multicast Middleware can send IP Multicast traffic back to the end system through the same virtual network interface. The packet flow is described in Fig. 2.1.

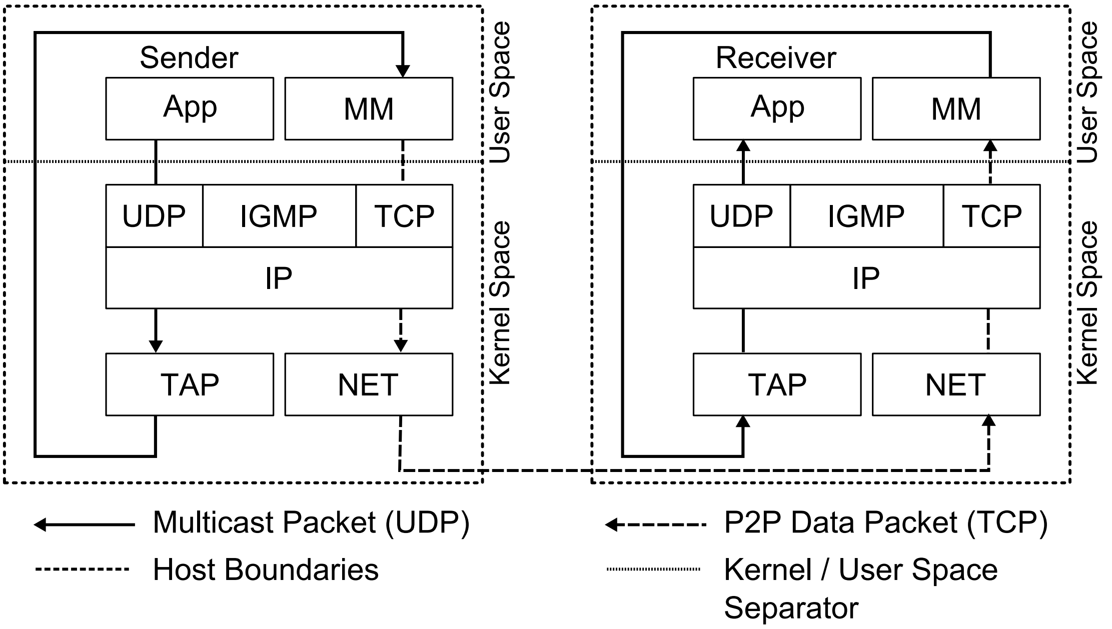

Figure 2.1: Packet flow between Applications and the Multicast Middleware

To establish an overlay multicast network, the Multicast Middleware needs to be installed on every computer in the network. As well as for IP Multicast, the NICs need to be activated with a fixed IP address according to the network topology setup. Afterward the file

start-middleware.sh needs to be reconfigured. We need to set the master node on the line: *export MASTER_NODE=${MASTER_NODE:=<INPUT>/2222}*. For the first computer in the network topology, the *<INPUT>* is its own IP address. For all other computers the *<INPUT>* is the IP address of its parent.
After this reconfiguration, the Multicast Middleware can be started with the command *sh start-middleware.sh*. Furthermore, on the computers directly linked to the Smartbits, the TAP-interface, generated by the Multicast Middleware, needs to be bridged with the NICs, which are directly connected to the Smarbits. To establish bridges between the NICs and TAPs, we used the bridge-util package. A bridge can be created using the brctl command and needs then to be activated using ifconfig. This bridging is necessary to allow the Multicast Middleware to intercept the packets sent from and deliver to the Smartbits.

Furthermore, on the computers directly liked to the Smartbits, we need to create a bridge between the TAP-interface, generated by the Multicast Middleware, and the local NIC. A bridge can be created using the brctl command and needs then to be activated using ifconfig. This bridging is necessary to allow the Multicast Middleware to intercept the packets sent from and deliver to the Smartbits.

2.3 Network Topologies

For our measurements we defined five different network topologies. Fig. 2.2 a)-c) show the different computer chains. We have chosen this kind of setup to see the behavior of the packet delay and the packet loss in a linear scalable network chain. The multicast interface can forward the packets without copying them to the next computer.
Fig. 2.2 d) and e) are both tree topologies. This setup was chosen to examine the behavior in a network, where the multicast interface has to copy the packets to forward them on multiple interfaces.

2.4 Code

As mentioned above, we controlled the Smartbits over a TCL interface. We have created two similar files to generate a multicast datastream for the native multicast and for the overlay multicast network. For the native multicast network we disabled the IGMP messages, due to the fact that we used static multicast routes. And for the overlay multicast network we enabled the IGMP messages.The full code is attached in Appendix B. See also [12, 13].
In both example codes the network ports are initialized first. The port number one (*iHub 0, iSlot 0, iPort 0*) is always used as the outgoing port, the other ports are used to capture the data. Then a package stream is created in which every packet obtains a unique signature for identification. Afterwards the intercepting of packets at the receiving ports is enabled. Finally the data stream is sent to the network through the outgoing port and the receiving ports intercept all packets, which have passed the network successfully and Smartbits provides the packet number, the time when the packet was sent, the time when the packet was intercepted and the delay.
For a proper setup of the Smartbits we have defined several variables, which are listed below.

Variables:

```
set ipAdr 10.0.0.2
```

The *ipAdr* variable is used by the smartlib.tcl and needs to be set to connect to the Smartbits.

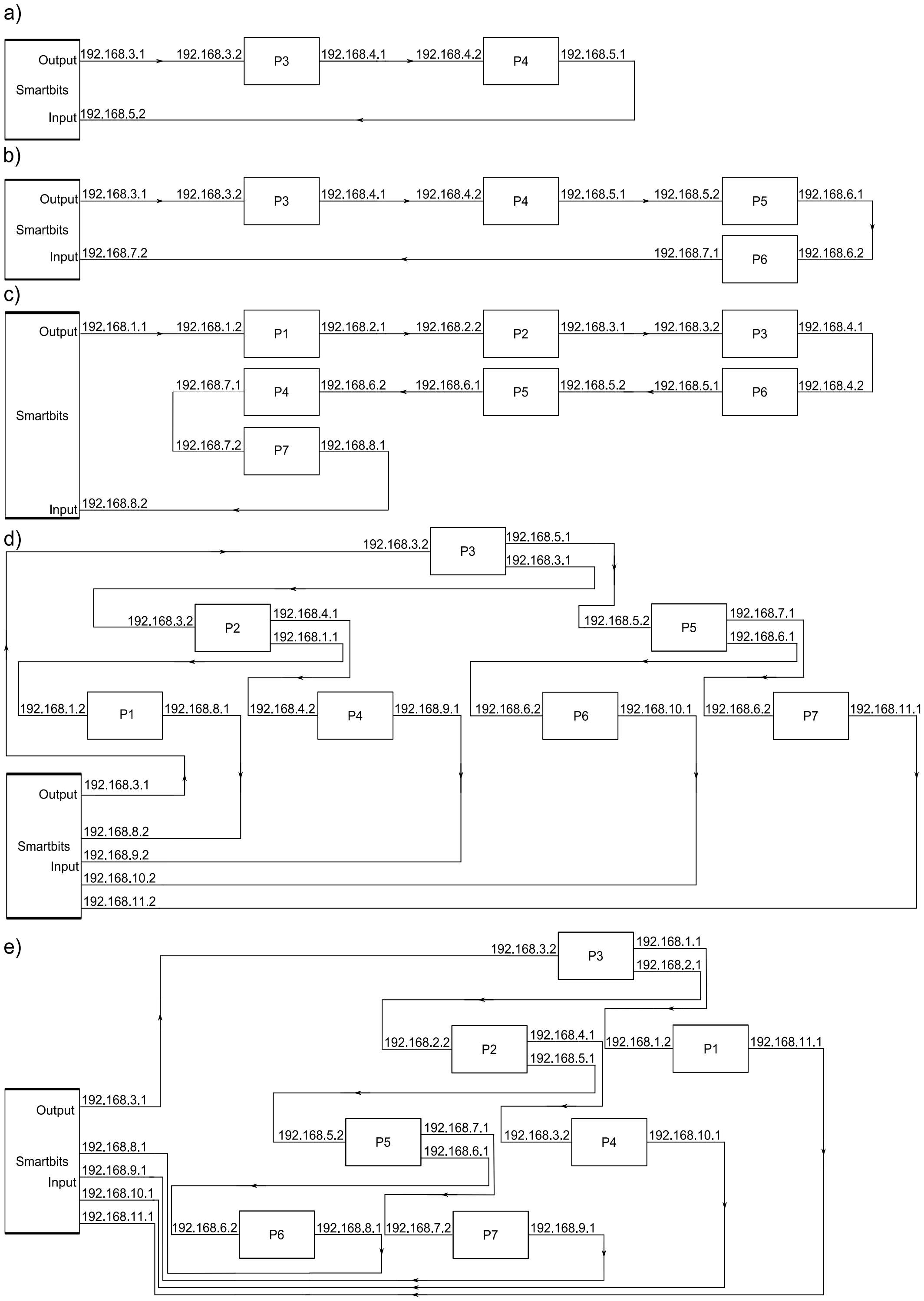

Figure 2.2: The different topologies used for the experiments (P1-P7 are Linux computers used as multicast routers)

```
set iHub 0
        set iSlot 0
        set iPort 0
```

These variables together define an Ethernet port. *iHub* identifies the destination SmartBits chassis, *iSlot* the destination slot and *iPort* the destination port. For our test the *iHub* and *iSlot* were always set to zero, as we only had one chassis and one module in the Smartbits.

```
set advRegisterInput  0x0080
```

This variable specifies the network technology. In our experiment it was set to 100-Base-TX.

```
set Speed2  0x0008
```

Defines the values for MII Register 1: Set variable for the network card to 100MHZ.

```
set ctrlRegisterInput      0xE410
```

Defines the values for control register MII Register 0. In our experiment we used the following settings: (PHY reset, Enable loopback mode, Speed Selection: 100MB, Auto Negotiation Disable, Power Down: Normal Operation, Isolate: Normal Operation, Restart Auto Negotiation: Normal, Duplex mode: Half-Duplex, Collision Test: Disable).

```
set streamNumber 1
```

Defines how many data stream have to be generated.

```
set numFrames 56000
```

Defines how many frames to send per data stream.

```
set gap 4000
```

Defines the inter-packet gap for packets transmitted on a addressed port (1 bit time = 10 nanoseconds).

```
set dataLength 559
```

Defines the frame length of a packet without CRC information. It is the payload of a package with 47 Bytes of header information.

To generate varied data steams, only the variables dataLength, numFrames and gap need to be changed.

2.5 Performed Measurements

To analyze the behavior of native multicast and overlay multicast we have defined three packet payloads and for each packet payload five throughput measurements. Due to the fact that we cannot enter the throughput, only the inter-packet gap in our code, we have made the measurements for every topology with the following settings:

Table 2.1: Traffic charactristics

Payload (Bytes)	**Data length** (Bytes)	**Network load** (mbps)	**Inter-packet gap** (bit time)	**# packets**
32	79	1.0	61 000	8 000
32	79	5.0	11 600	40 000
32	79	10.0	5 500	80 000
32	79	24.8	1 720	130 000
32	79	49.6	485	130 000
512	559	1.0	430 000	1 200
512	559	24.8	12 500	28 000
512	559	49.6	4 000	56 000
512	559	75.2	1 100	85 000
512	559	84.8	450	95 000
1 024	1 071	1.0	830 000	600
1 024	1 071	24.8	24 000	15 000
1 024	1 071	49.6	7 700	29 000
1 024	1 071	75.2	2 250	44 000
1 024	1 071	84.8	960	50 000

Chapter 3

Experiment Results and Evaluation

In our experiment we conducted measurements with five different network topologies. For every topology we performed fifteen measurements, varying network load and payload. Altogether we completed 75 measurements with IP Multicast and equivalent with overlay multicast. During the experiment, we faced several problems. We had some technical problems with the network interface cards (NICs), every once a while they crashed in the middle of an experiment. We discovered that when the Smartbits generated too heavy traffic, meaning that the inter-packet gap was too small, the NIC could not handle the traffic anymore and crashed. Furthermore, we had several problems with faulty memory blocks.

3.1 Native Multicast Results

3.1.1 Chain Topologies

In the measurements with the chain topologies (Fig. 2.2 a-c), we had some packets loss. As shown in Fig. 3.1, Fig. 3.2 and Fig. 3.3 most of the packets got lost with the 32 bytes packet payload configuration. But overall the packet loss was more or less constant in all three topologies. For the configuration with a payload of 1 024 bytes, we measured the least packet loss.

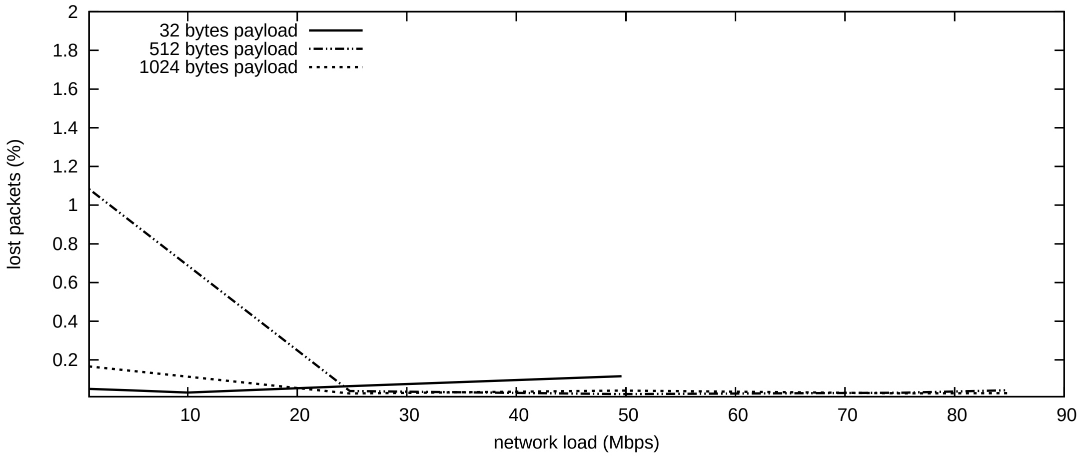

Figure 3.1: Packet loss with IP Multicast in topology a)

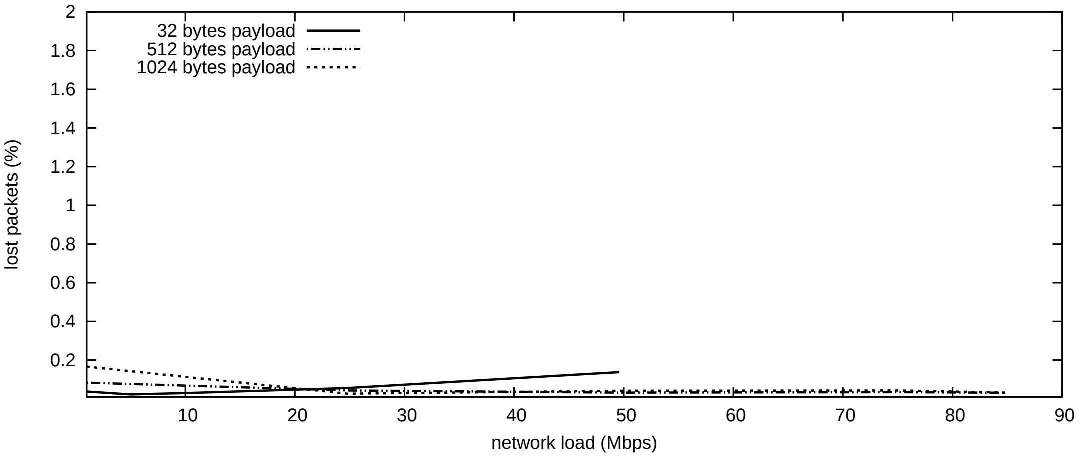

Figure 3.2: Packet loss with IP Multicast in topology b)

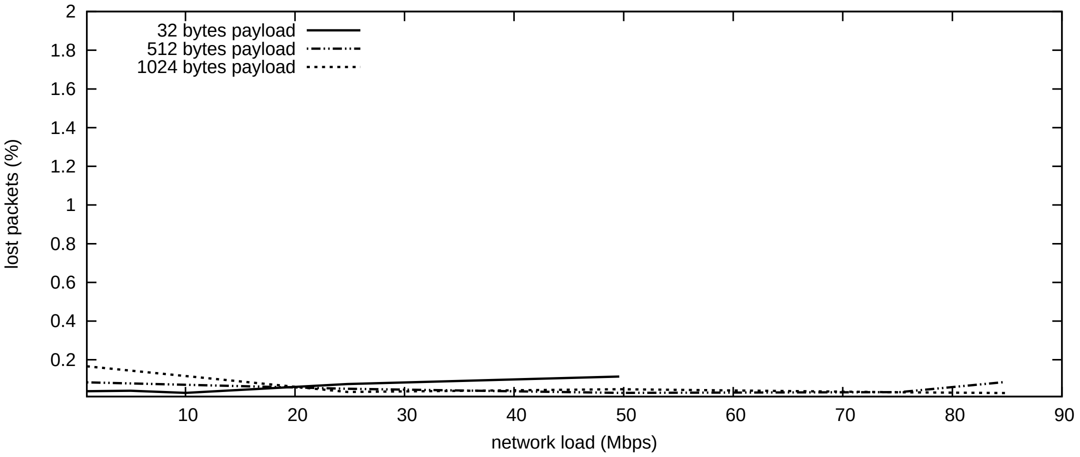

Figure 3.3: Packet loss with IP Multicast in topology c)

The latencies for the chain topologies as shown in Fig. 3.4, Fig. 3.5 and Fig. 3.6 were linear to the number of computers. In addition the network load had nearly no influence on the latency. However for the configurations with a network load of one Mbps, the latency was a little higher comparing to the other configurations. This deviation grew with a higher payload as well as with the number of computers in the topology chain. A reason for this deviation could be the high inter-gap time between the packets, which could cause a short delay at the kernel-forwarding.

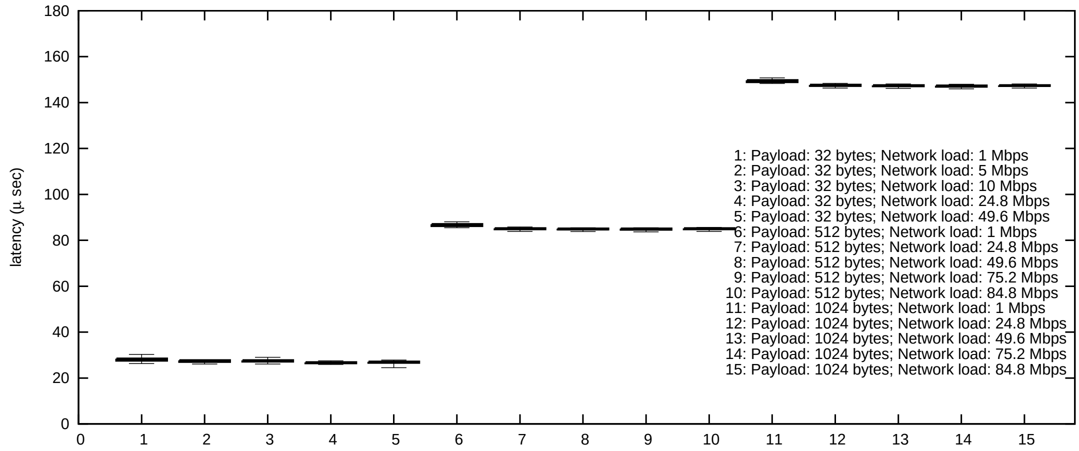

Figure 3.4: Latency w/o 5% outliers and with IP Multicast in topology a)

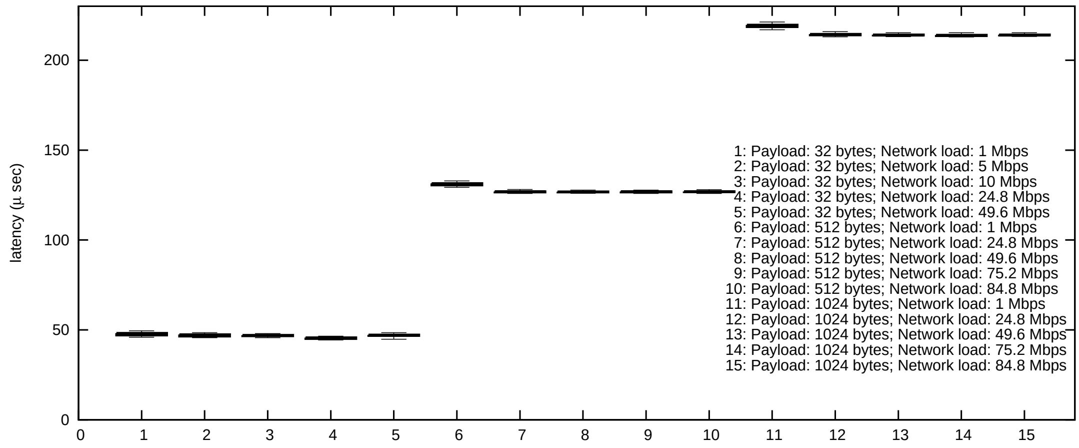

Figure 3.5: Latency w/o 5% outliers and with IP Multicast in topology b)

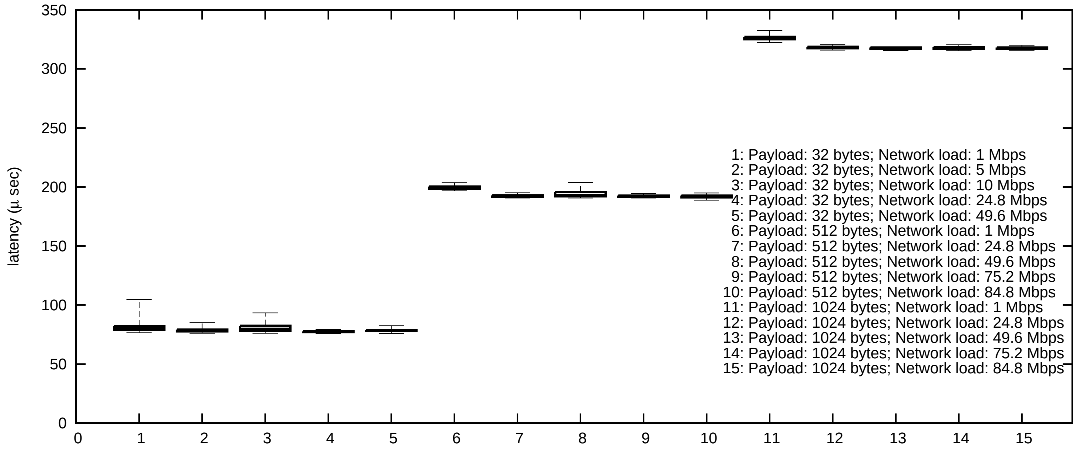

Figure 3.6: Latency w/o 5% outliers and with IP Multicast in topology c)

3.1.2 Tree Topologies

For the tree topologies we measured the packet loss and the latency from the output of every computer located at the end of the tree. As shown in the first tree topology (Fig. 2.2 d) we measured the output from PC1, PC4, PC6 and PC7. The packet loss as well as the latency were more or less identical for these four end points. Therefore we only show the measurement results from PC4.
Compared to the chain topologies, the packet loss in Fig. 3.7 did not change significantly although, for configurations with higher payload, the packet loss increased a little.

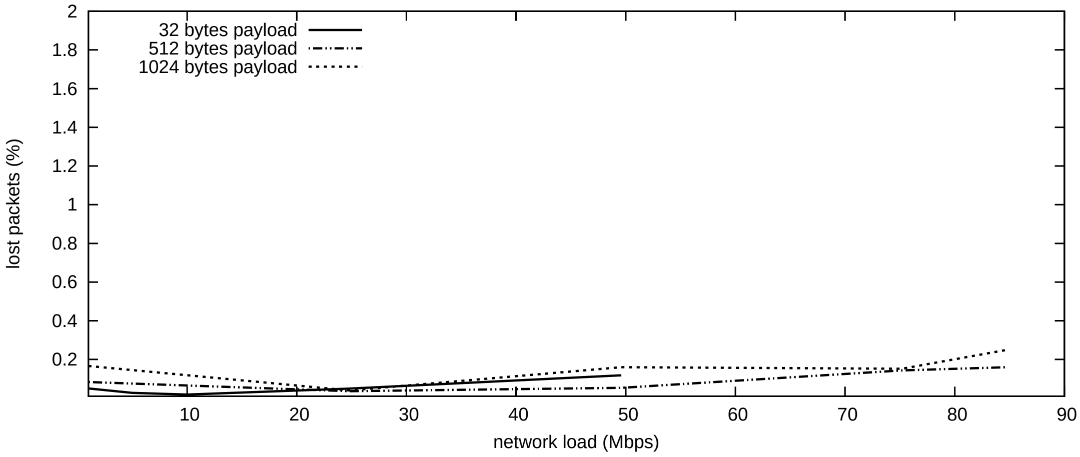

Figure 3.7: Packet loss with IP Multicast in topology d) for PC4

Considering the latency for this topology in Fig. 3.8, it was almost equal for every configuration. Comparing to the chain topology, where the latency was linear to the number of computers, the latency had a different behavior in the tree topology.
Furthermore, we had two configurations with a bigger jitter as on the average. For configurations with a small payload and a high network load the jitter seems to increase. This behavior could be caused by the

small inter-packet gap, and the amount of packets sent to the computers, as with this configuration the kernel cannot transfer the packets quickly enough and they get queued.

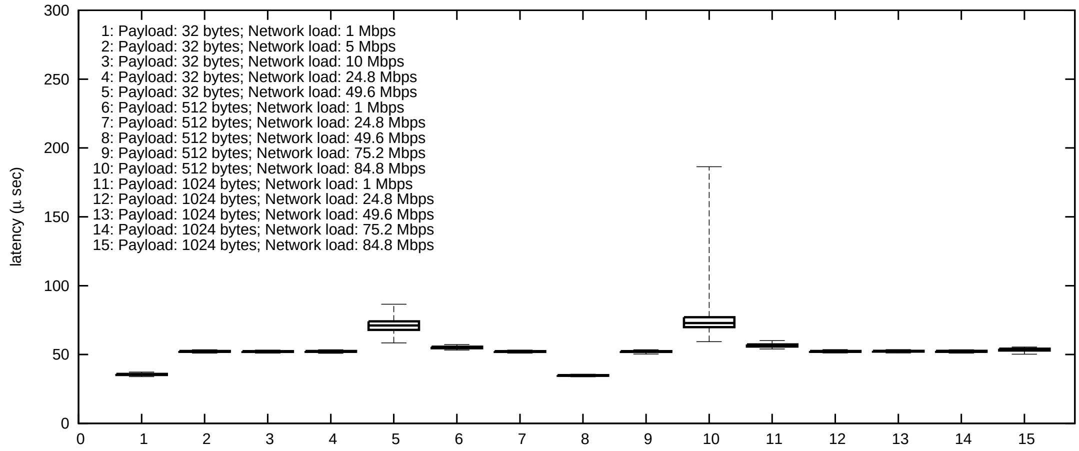

Figure 3.8: Latency w/o 5% outliers and with IP Multicast in topology d) for PC4

For the second tree topology (Fig. 2.2 e) we captured the data on the different depths of the tree. The packet losses for PC1 (tree depth 1), PC4 (tree depth 2), PC6 (tree depth 3) and PC7 (tree depth 3) were equal and therefore we only show the packet loss for PC1 in Fig. 3.9 and for PC6 in Fig. 3.11. It can be compared to the diagrams we have seen in the first tree topology.

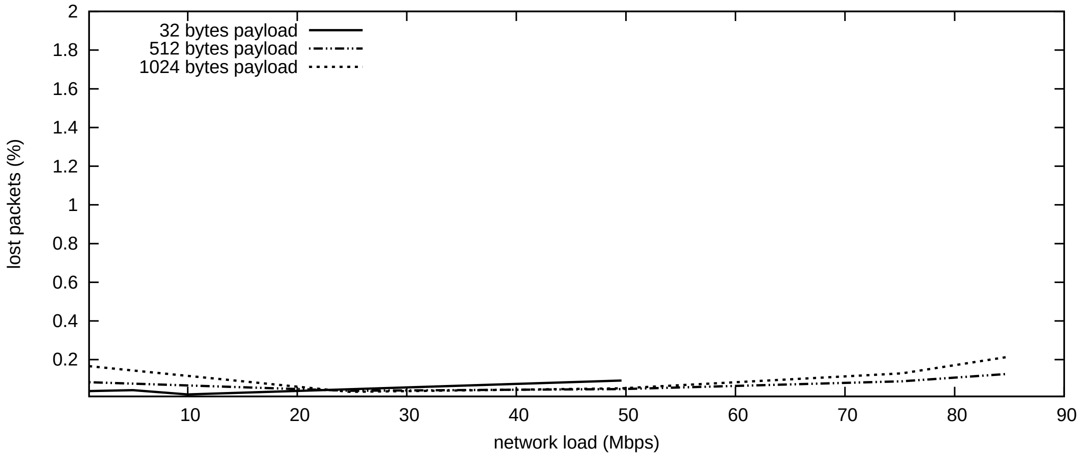

Figure 3.9: Packet loss with IP Multicast in topology e) for PC1

Considering the latency for this tree topology, we can see almost the same behavior. The latencies of PC1 and PC4 (Fig. 3.10) are almost equal and we have the same behavior as for topology in Fig. 2.2 d). But if we look at the latency of PC6 and PC7 (Fig. 3.12), a deviation from the average is shown for configurations with small payload and high network load. Furthermore, the jitter is increased at these configurations.

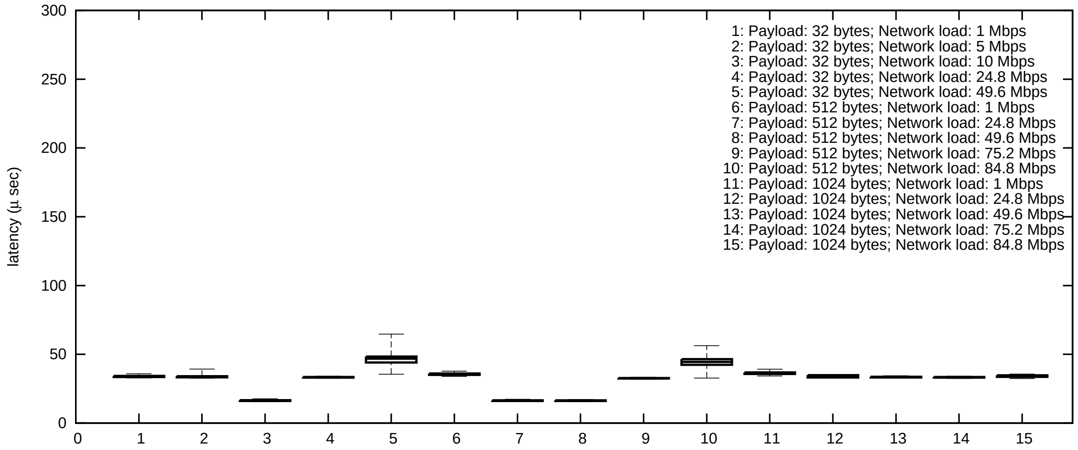

Figure 3.10: Latency w/o 5% outliers and with IP Multicast in topology e) for PC1

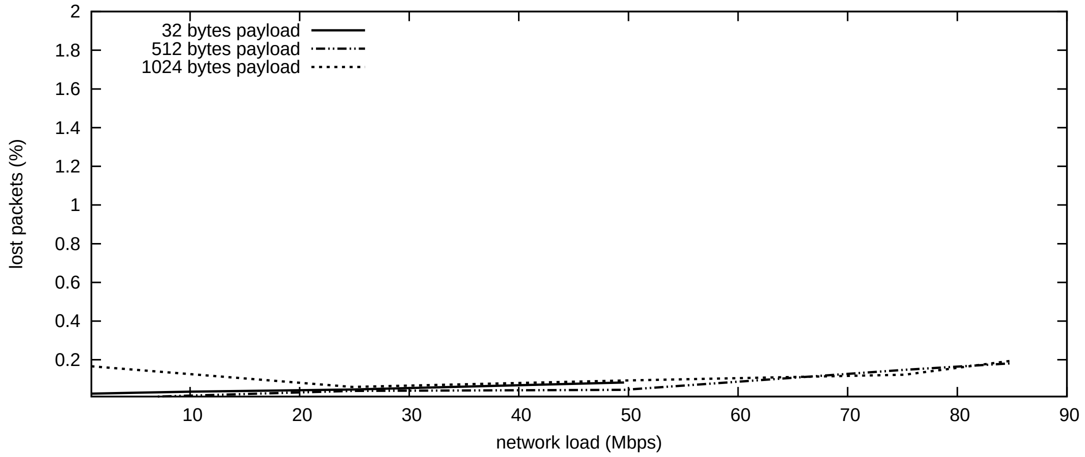

Figure 3.11: Packet loss with IP Multicast in topology e) for PC6

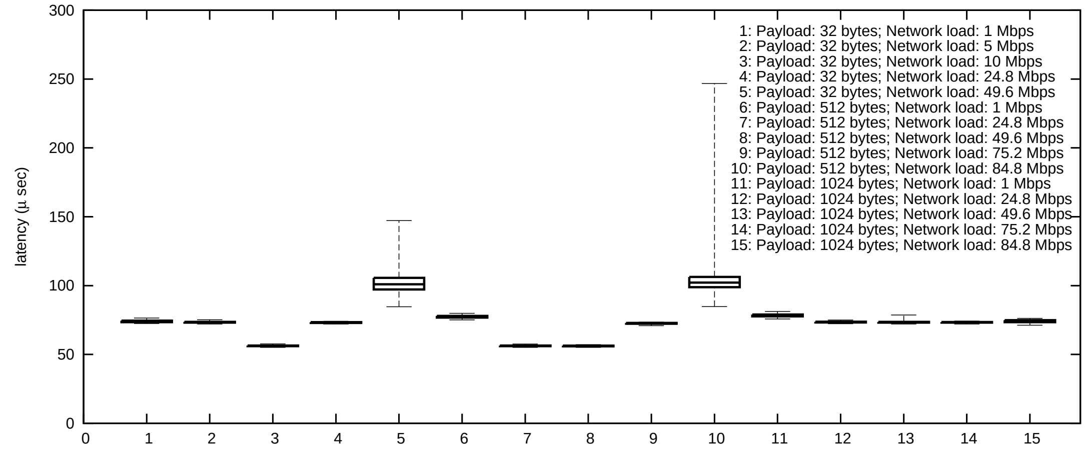

Figure 3.12: Latency w/o 5% outliers and with IP Multicast in topology e) for PC6

3.2 Overlay Multicast Results

3.2.1 Chain Topologies

We conducted the same measurements from Section 3.1 with overlay multicast enabled on the computers. For the chain topologies (Fig. 2.2 a-c) the packet losses are shown (Fig. 3.13, Fig. 3.14 and Fig. 3.15). Comparing to the native multicast measurements for the same topologies the packet losses have increased heavily. For the configurations with 32 bytes payload in all three topologies, the packet loss grew rapidly up to 90 percent.

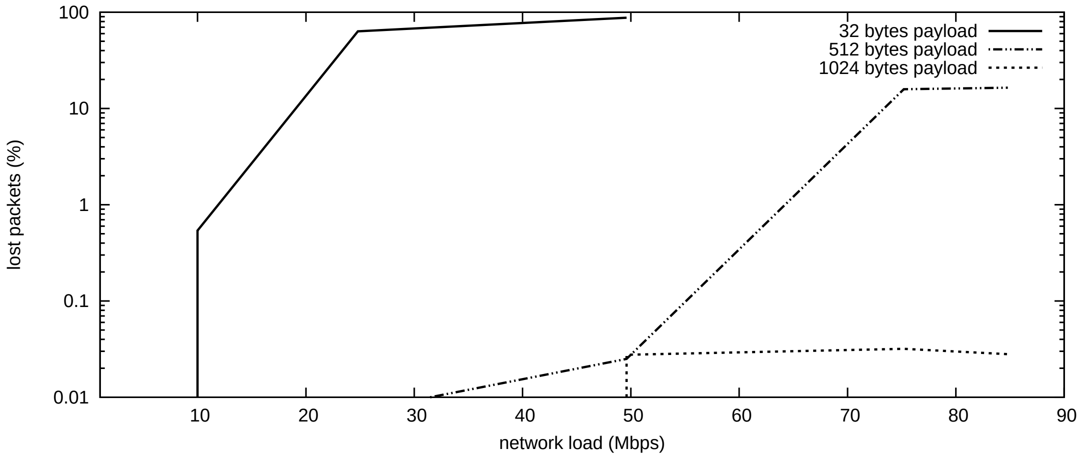

Figure 3.13: Packet loss with overlay multicast in topology a)

For small network load (1mbps) the packet loss had the same behavior as with native multicast (Fig. 3.4, Fig. 3.5 and Fig. 3.6). And for the configurations with a payload of 1 024 bytes the packet loss was also comparable with native multicast. The slight decrease of the packet loss for the configurations with 1 024 bytes payload can be explained with the conversion of the packet loss into percentage.

Comparing the measurements with 512 bytes payload to native multicast (Fig. 3.4), the packet loss increased up to 16 percent.

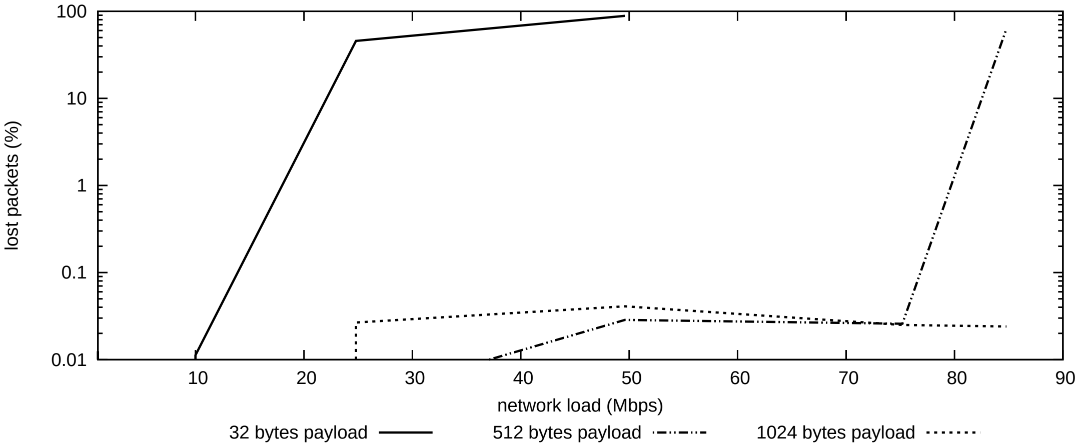

Figure 3.14: Packet loss with overlay multicast in topology b)

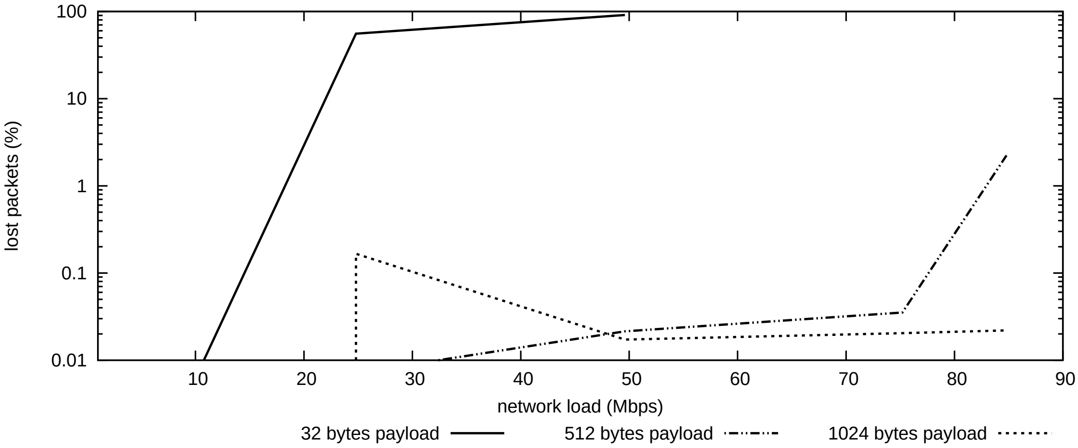

Figure 3.15: Packet loss with overlay multicast in topology c)

For the latency measured upon the chain topologies (Fig. 3.16, Fig. 3.17 and Fig. 3.18), we saw a different behavior as with native multicast. For configurations with a payload of only 32 bytes, the measurements with a network load up to 10 Mbps provided acceptable results. For network load greater than 20 Mbps the delay was too big. The results for the configurations with a payload of 512 bytes and 1 024 bytes were also acceptable. However with a network load of 1 Mbps we had a large jitter. The best results were provided with a 1 024 bytes payload and a network load between 24.8 and 84.8 Mbps, but comparing with native multicast the average delay with overlay multicast was much higher. The different behavior of the measurements (in Fig. 3.16, Fig. 3.17 and Fig. 3.18) can be explained with locking issues in the Multicast Middleware. With more peers in the chain topology (Fig. 3.17 and Fig. 3.18), the behavior observed got worse.

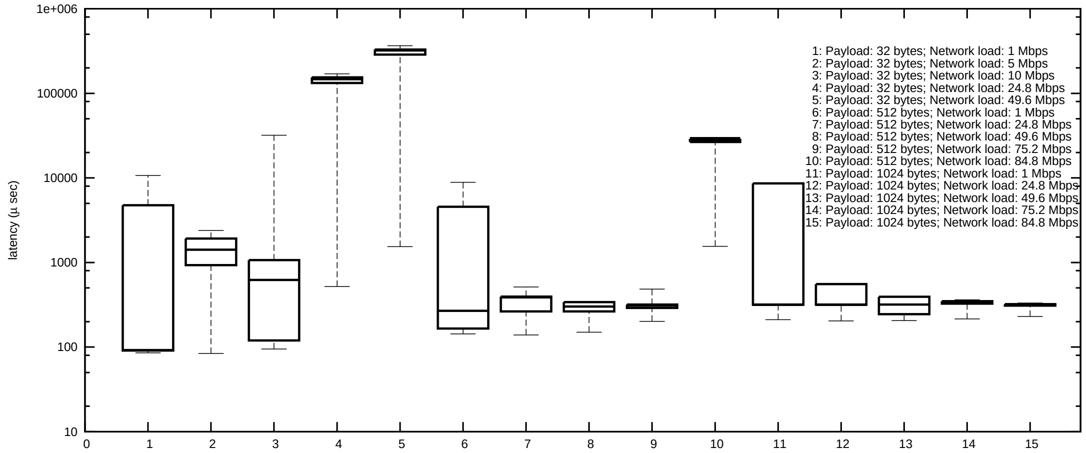

Figure 3.16: Latency w/o 5% outliers and with overlay multicast in topology a)

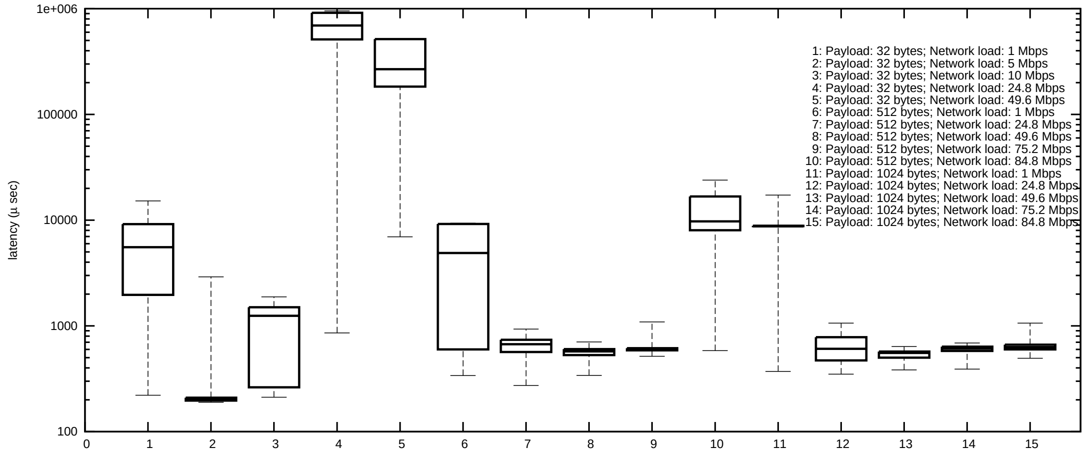

Figure 3.17: Latency w/o 5% outliers and with overlay multicast in topology b)

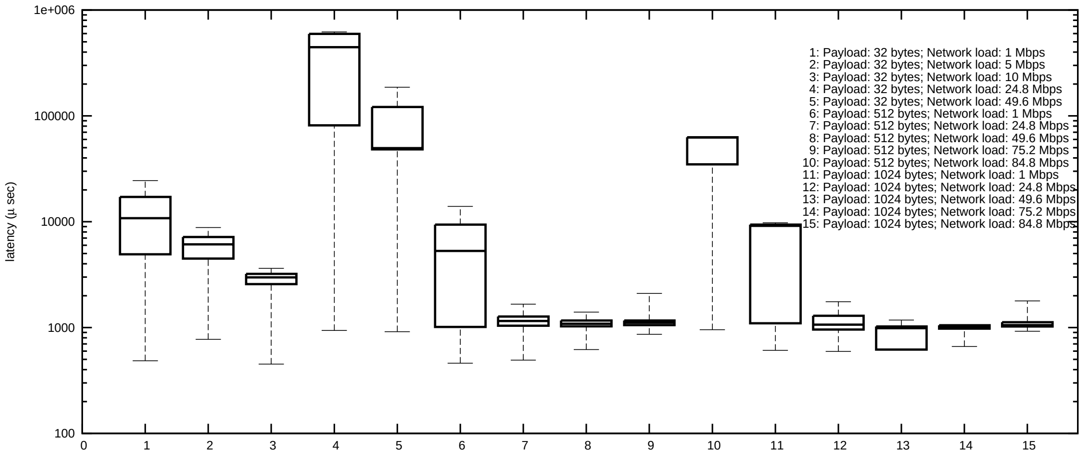

Figure 3.18: Latency w/o 5% outliers and with overlay multicast in topology c)

3.2.2 Tree Topologies

Below we present the results for the measurements performed with overlay multicast in the first tree topologies (Fig. 2.2 d). Equal to the results we measured with native multicast in section 3.1.2, the packet loss on the different paths were similar. Therefore we only present the packet loss measured from the output of PC4 in Fig. 3.19. We had a very high average packet loss in this topology compared to the results from the chain topologies with overlay multicast. Again this behavior can be explained with locking issues from the Multicast Middleware.

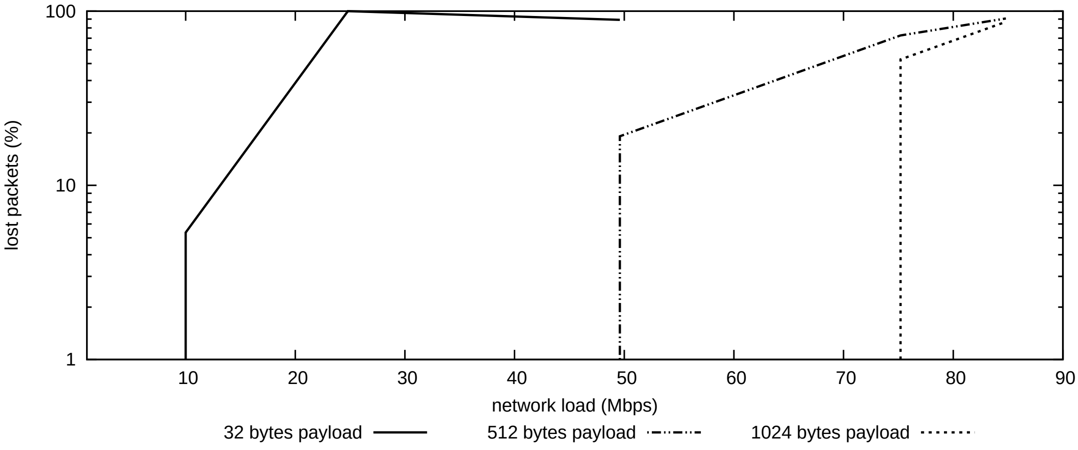

Figure 3.19: Packet loss with overlay multicast in topology d) for PC4

Also for the latency we present only one measurement from PC4, as the measurements of the other paths were almost equal. Considering the results in Fig. 3.20, we have huge differences between the different results. Only for configurations with a small network load the latency is in an acceptable range. For configurations with a higher network load the latency and the jitter are growing heavily.

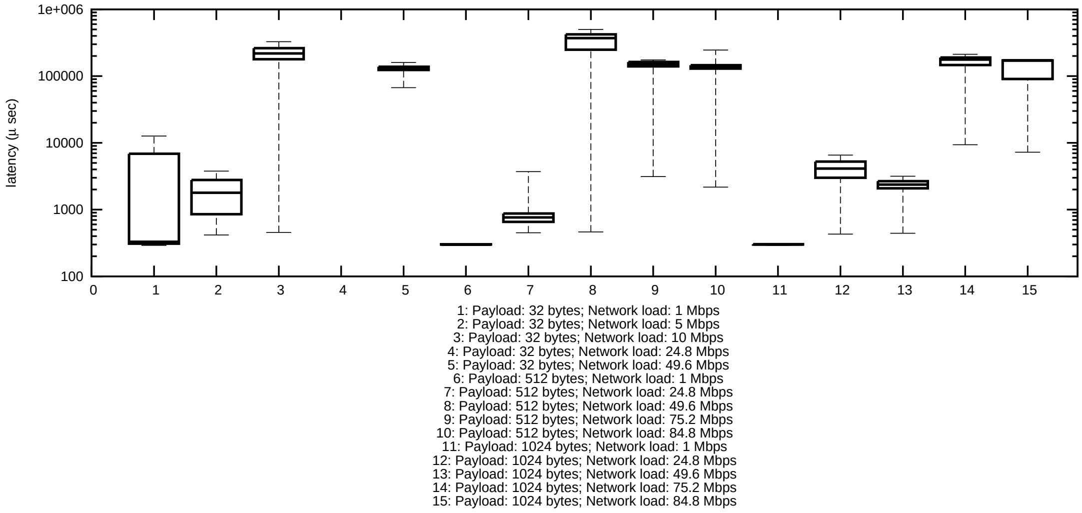

Figure 3.20: Latency w/o 5% outliers and with overlay multicast in topology d) for PC4

For the second tree topology (Fig. 2.2 e) we measured the data on the different depths of the tree. The measurements for the packet loss showed an equal result on all paths for the different depths in the tree, therefore we only present the packet loss for PC4 in Fig. 3.21. Similar to the first tree topology we have a high packet loss for all configurations compared to the results from the chain topologies with overlay multicast.

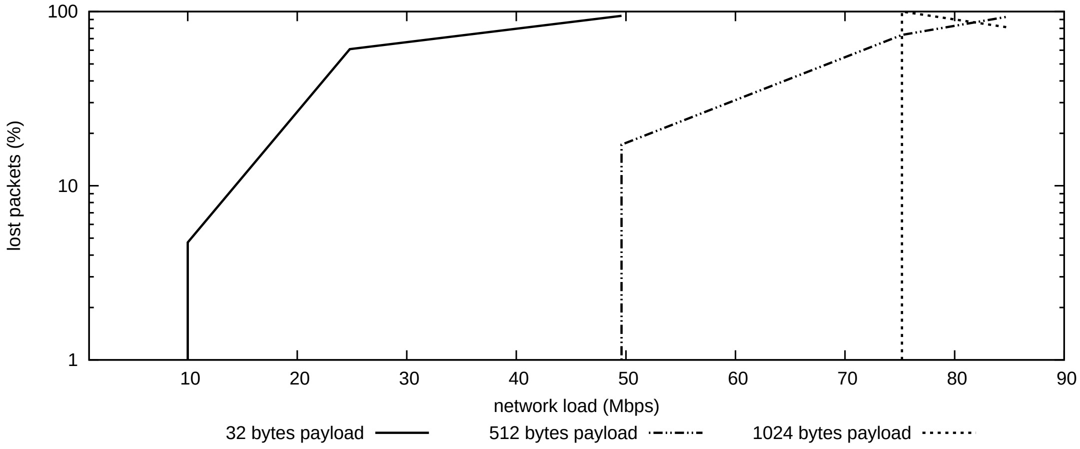

Figure 3.21: Packet loss with overlay multicast in topology e) for PC4

For the latency in this topology we had almost an equal latency on every depth of the tree we measured, therefore we only show the result of PC4 in Fig. 3.22. Similar to the results of the first tree topology the average latency is extremely high. Only for a few configurations with high payload and low network load the latency is in an acceptable range.

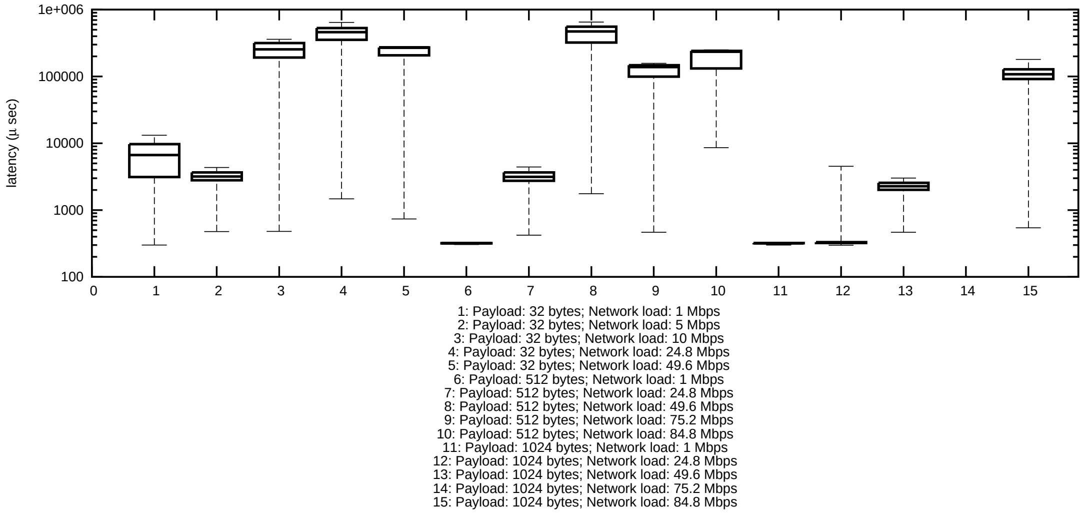

Figure 3.22: Latency w/o 5% outliers and with overlay multicast in topology e) for PC4

3.3 Evaluation

Our results have shown that the overlay multicast has a higher average latency and a higher packet loss as IP Multicast. That the overlay multicast has a higher latency was expected, due to the fact that routing is not performed in the kernel itself. The higher packet loss though was not expected. We explain this behavior with the queuing and dropping of packets.

The measurements with IP Multicast have shown that the packet loss was below 0.2 percent for all configurations with chain topologies (Fig. 3.1, Fig. 3.2 and Fig. 3.3) as well as with tree topologies (Fig. 3.7, Fig. 3.9 and Fig. 3.11). However, for the measurements with configurations of 32 bytes payload we have seen an increase of the packet loss in all topologies almost linear to the network load. For measurements with a network load below 10 Mbps, the packet loss was higher as with a network load greater than 10 Mbps. The latency in the measurements with IP Multicast behaved as expected.

The measurements with overlay multicast have shown that the packet loss has increased comparing to IP Multicast. In the chain topologies (Fig. 3.13, Fig. 3.14 and Fig. 3.15) as well as in the tree topologies (Fig. 3.19 and Fig. 3.21) the packet loss increased up to 90 percent for the configurations with 32 bytes payload and a network load of 24.8 Mbps. For the configuration with 512 bytes payload the packet loss increased significantly from a network load of 49.6 Mbps. For the configurations with 1 024 bytes payload the packet loss remained below one percent for the chain topologies and increased up to 90 percent for the tree topologies with a network load of 75.2 Mbps. The latencies for the measurements with overlay multicast has increased significantly comparing to IP Multicast. An explanation for this result could be that the Multicast Middleware is not suitable for small hops (hops with short delays). The distance and therefore the delay between the PCs was extremely short and had an influence on the queuing of the packets.

Comparing the two multicast implementations, we have shown that IP Multicast delivers better results with a bigger variety in relation to the payload and network load. Overlay multicast has shown acceptable results with a payload of 1 024 bytes and a network load between 49.6 Mbps and 75.2 Mbps in a chain topology network. Finally, for tree topology with overlay multicast, the results were acceptable for small payload and small network load.

Chapter 4

Conclusion and Outlook

In this paper we have compared the performance of native IP Multicast with overlay multicast in a real time experiment. We have described how we have set up the experiment and how we have performed the measurements using IP Multicast for native multicast and the Multicast Middleware for overlay multicast. To conduct the measurements we have used a network performance analysis system called ”Smartbits”. We have performed the measurements on different topologies varying network load and payload.

The results of our experiment shows that the Multicast Middleware performs fairly with a payload of 1 024 bytes and a network load between 49.6 Mbps and 75.2 Mbps in a chain topology network. Furthermore, we have conducted measurements in tree topology networks. The Multicast Middleware only delivered acceptable results for measurements with small payload and small network load. The reason for this result was that we used only a local network topology with small delays.

To improve our results, we could conduct further measurements using a larger topology of computers or using the open platform PlanetLab. Also measurements could be performed with background traffic in the network.

Bibliography

[1] M. Brogle, D. Milic, and T. Braun, “Supporting IP Multicast Streaming Using Overlay Networks.” QShine: International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness, Vancouver, British Columbia, Canada,August 14 - 17 2007.

[2] S. Deering, “Host extensions for IP multicasting,” RFC 1112 (Standard), Aug. 1989, updated by RFC 2236. [Online]. Available: http://www.ietf.org/rfc/rfc1112.txt

[3] J. Postel, “Internet Protocol,” RFC 791 (Standard), Sept. 1981, updated by RFC 1349. [Online]. Available: http://www.ietf.org/rfc/rfc791.txt

[4] Z. Albanna, K. Almeroth, D. Meyer, and M. Schipper, “IANA Guidelines for IPv4 Multicast Address Assignments,” RFC 3171 (Best Current Practice), Aug. 2001. [Online]. Available: http://www.ietf.org/rfc/rfc3171.txt

[5] W. Fenner, “Internet Group Management Protocol, Version 2,” RFC 2236 (Proposed Standard), Nov. 1997, obsoleted by RFC 3376. [Online]. Available: http://www.ietf.org/rfc/rfc2236.txt

[6] B. Cain, S. Deering, I. Kouvelas, B. Fenner, and A. Thyagarajan, “Internet Group Management Protocol, Version 3,” RFC 3376 (Proposed Standard), Oct. 2002, updated by RFC 4604. [Online]. Available: http://www.ietf.org/rfc/rfc3376.txt

[7] S. Fahmy and M. Kwon, “Characterizing overlay multicast networks,” in *Network Protocols, 2003. Proceedings. 11th IEEE International Conference on*, 2003, pp. 61–70. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1249757

[8] M. Bayer, *Computer Telephony Demystified: Putting CTI, Media Services, and IP Telephony to Work with CDROM*. Mcgraw Hill Book Co, 2000.

[9] *Installation Guide SmartBits 600x/6000x*, Spirent Communication Inc.

[10] C. Schill, “SMC route.” [Online]. Available: http://www.cschill.de/smcroute/

[11] D. Milic and M. Brogle, “Euqos multicast middleware.” [Online]. Available: http://www.iam.unibe.ch/~rvs/research/euqos/mcast-4.0.4_r701.zip

[12] *SmartLib Command Reference Volume 2*, Spirent Communication Inc., 2003.

[13] *SmartLib Command Reference Volume 1*, Spirent Communication Inc., 2003.

Appendices

Appendix A

Computer Hardware and Software Specifications

Table A.1: Hardware of the computers used for the experiments

Processor	Intel Pentium IV 3.00 GHz
Memory	2x 512GB Take M5 DDR 400 CL 2.5
Motherboard	ASUS P4S800-MX
Bios version	Rev. 0501
Additional Network Card Interface (NIC):	1x SiS 900/7016 100 Mbps Onboard 2x Realtek, RTL-8169 1 000 Mbps
Hard Disk	Hitachi Deskstar 7K80 80GB HDS728080PLAT20

Table A.2: Software on the computers used for the experiments

Operating System	Fedora Core 5 (2.6.20-1.2307)
Kernel	2.6.20-1.2307
Services running:	acpid gpm haldaemon irqbalance kudzu network sshd syslog

Additional Software:

To create a bridge between two Ifaces:

sysfsutil	V. 1.3.0-1.2.1
bridge-util	V. 1.0.6-1.2

Native multicast support:

SMC Route	V. 0.92

Overlay multicast support:

Multicast Middleware	V. 30 Jan 2006

Appendix B

Smartbits Code Example 1

This is the code to generate one multicast stream (payload: 512 Bytes, throughput: 49.6 Mbps) without generating IGMP messages.

```
##############################################################################
### Load smartlib.tcl ########################################################
        set libPath ”/usr/local/smartbits/smartlib/include/smartlib.tcl”
##############################################################################
### Smartbits Configuration Port Address######################################
        set ipAddr 10.0.0.2
##############################################################################
### Port Configuration #######################################################
        set iHub 0
        set iSlot 0
        set iPort 0

        set iHub2 0
        set iSlot2 0
        set iPort2 1
###############################################################################
### set auto negoatiation advertisement register ##############################
        set advRegisterInput    0x0080
        ### Technology ability : 100 Base−TX
###############################################################################
        set Speed2  0x0008
### 100MegFull = 0x0101 ; 10MegFull 0x0041
### Speed2 100MHz: 0x0008 ; 10 MHz: 0x0004
###############################################################################
####### set control register input ############################################
        set ctrlRegisterInput      0xE410
### 0x1204 = on
###############################################################################
### Numbers of Stream to create################################################
        set streamNumber 1
###############################################################################
### Numbers of Frames to send #################################################
        set numFrames 56000

        set gap 4000
###############################################################################
### Frames per Second #########################################################
        set framesPerSecond 2000
### or ########################################################################
### Load Percent (0 − 100) ####################################################
        set loadPerCent 60
###############################################################################
### Set registeraddress #######################################################
        set controlReg      0
        ### control register address
        set advertiseReg    4
        ### auto negotiation advertisement register address
###############################################################################
### Frame Length (no CRC) 0 − 2148 for Ethernet ###############################
        set dataLength 559
##############################################################################
### Load Smartlib#############################################################
## if ”smartlib.tcl” is not loaded, try to source it from the default path
        if { ! [info exists __SMARTLIB_TCL__] } {
                if {[file exists $libPath]} {
                        source $libPath
                } else {
                        #Enter the location of the ”smartlib.tcl” file or enter ”Q” or ”q” to quit
                        while {1} {
                                puts ”Could␣not␣find␣the␣file␣$libPath.”
                                puts ”Enter␣the␣path␣of␣smartlib.tcl,␣or␣q␣to␣exit.”
                                gets stdin libPath
                                if {$libPath == ”q” || $libPath == ”Q”} {
                                        exit
                                }
                                if {[file exists $libPath]} {
```

```
                                        source $libPath
                                        break
                                }

                        }
                }
        }
################################################################################
### Link to Smartbits #############################################################
        proc linkSMB {ip} {
                if {[ETGetLinkStatus] < 1} {
                        puts ”SmartBits␣chassis␣IP␣address:␣$ip”
                        NSSocketLink $ip 16385 $::RESERVE_ALL
                }
        }
################################################################################
###sets mii register to value of word######################################
        proc writeMII {H S P word register } {
                set address [getMIIAddress $H $S $P]
                LIBCMD HTWriteMII $address $register $word $H $S $P
        }
################################################################################
################################################################################
        proc getMIIAddress {H S P} {
                set address ””
                set con_reg ””
                LIBCMD HTFindMIIAddress address con_reg $H $S $P
                return $address
        }
################################################################################
### simeple wait for user input routine#########################################
        proc press2Continue {} {
                puts ”␣Press␣ENTER␣to␣continue”
                gets stdin response
        }
################################################################################
### create new Streams ##########################################################
        proc createStreamArray {H S P numStreams} {
                struct_new ip StreamIP*$numStreams
                for {set i 0} {$i < $numStreams} {incr i} {
                        set ip($i.ucActive)                             1
                                ### ucActive: 1 = enable Stream ; 0 disable Stream
                        set ip($i.ucProtocolType)                       $::STREAM_PROTOCOL_TCP
                        set ip($i.uiFrameLength)                        $::dataLength
                                ### Frame Length not counting CRC 0 -2148 for Ethernet
                        set ip($i.ucTagField)                           1
                                ### 0 = off ; 1 = insert signature into each frame
                        set ip($i.Protocol)                             4
                                ### 4 = ip on the IP assigned list
                        set ip($i.TimeToLive)                           0xFF
                        set ip($i.DestinationMAC.0)                     0x01
                                #PC1 0x00  Router1  0x00  Router2 0x00
                        set ip($i.DestinationMAC.1)                     0x00
                                #PC1 0x11 Router1 0x15 Router2 0x15
                        set ip($i.DestinationMAC.2)                     0x5e
                                #PC1 0x6B Router1 0xf9 Router2 0xf9
                        set ip($i.DestinationMAC.3)                     0x01
                                #PC1 0x34 Router1 0x56 Router2 0x56
                        set ip($i.DestinationMAC.4)                     0x02
                                #PC1 0x9A Router1 0xbe Router2 0xba
                        set ip($i.DestinationMAC.5)                     0x03
                                #PC1 0x23 Router1 0xe0 Router2 0xd8
                        set ip($i.SourceMAC.0)                          0x00
                        set ip($i.SourceMAC.1)                          0x00
                        set ip($i.SourceMAC.2)                          0x01
                        set ip($i.SourceMAC.3)                          0x00
                        set ip($i.SourceMAC.4)                          0x00
                        set ip($i.SourceMAC.5)                          0x01
                        set ip($i.DestinationIP.0)                      224
                        set ip($i.DestinationIP.1)                      1
                        set ip($i.DestinationIP.2)                      2
                        set ip($i.DestinationIP.3)                      3
                        set ip($i.SourceIP.0)                           192
                        set ip($i.SourceIP.1)                           168
                        set ip($i.SourceIP.2)                           3
                        set ip($i.SourceIP.3)                           1
                        set ip($i.Netmask.0)                            255
                        set ip($i.Netmask.1)                            255
                        set ip($i.Netmask.2)                            255
                        set ip($i.Netmask.3)                            0
                        set ip($i.Gateway.0)                            192
                        set ip($i.Gateway.1)                            168
                        set ip($i.Gateway.2)                            3
                        set ip($i.Gateway.3)                            2
                        set ip($i.ulARPGap)                                     1000
                                ### the time between ARPs in 100ns
                        }
                LIBCMD HTSetStructure $::L3_DEFINE_IP_STREAM 0 0 0 ip 0 $H $S $P
        }
################################################################################
### create Stream Extensions####################################################
        proc create_extension {H S P frameRate } {
                set numStreams [get_streamcount $H $S $P]
        struct_new L3X L3StreamExtension
                for {set index 1} {$index < $numStreams} {incr index} {
                set L3X(ulFrameRate)                                    $frameRate
                        ### frame transmit rate, in packets per second
```

```
        set L3X(ucIPHeaderChecksumError)        1
            ### 1 = enable ; 0 = disable
        set L3X(ucIPTotalLengthError)           1
            ### 1 = enable ; 0 = disable
        set L3X(ucCRCErrorEnable)                       1
            ### 1 = enable ; 0 = disable
        set L3X(ucDataCheckEnable)                      1
            ### 1 = enable pazload check error ; 0 = disable
        set L3X(ucDataIntegrityErrorEnable) 1
            ### 1 = enable ; 0 = disable
        set L3X(ulBGPatternIndex)                       0
            ### index of the stream background fill pattern
            ### that defined by L3StreamBGConfig
        set L3X(ucRandomBGEnable)                       1
            ###  1 = enable ; 0 = disable
        puts ”creating␣extension␣$index”
        LIBCMD HTSetStructure $::L3_MOD_STREAM_EXTENSION $index 0 0 L3X 0 $H $S $P
        }
    }
################################################################################
###Restarts capture on target card to capture all received frames##############
    proc resetCapture {H S P} {
        struct_new CapSetup NSCaptureSetup
        set CapSetup(ulCaptureMode) $::CAPTURE_MODE_FILTER_ON_EVENTS
            ### Choose between:
            ### − CAPTURE_MODE_FILTER_ON_EVENTS
            ### − CAPTURE_MODE_START_ON_EVENTS
            ### − CAPTURE_MODE_STOP_ON_EVENTS
        set CapSetup(ulCaptureEvents) $::CAPTURE_EVENTS_ALL_FRAMES
            ### Choose between: see list
        LIBCMD HTSetStructure $::NS_CAPTURE_SETUP 0 0 0 CapSetup 0 $H $S $P
            ### Set structure
    }
################################################################################
### reset Rawtags###############################################################
    proc resetRawtags { H S P} {
        LIBCMD HTSetCommand $::L3_HIST_RAW_TAGS 0 0 0 0 $H $S $P
    }
################################################################################
### display the results########################################################
    proc displayRawtags {H S P numFrames} {
        set actLatency 0
        set avgLatency 0
        set minLatency 0
        set maxLatency 0
        struct_new rt Layer3HistTagInfo
            ### rt contains the singature field information for the Raw Tag
            ###histogramm. A Separate record is generated for each SmartBit
            ###test frame (containing a signature) received at this port.
    set recordsOnCard [getRecord $H $S $P]
        ### get the number of histogram records captured on the target
        if {$recordsOnCard < 1} {
            puts ”No␣records␣captured␣on␣Hub␣$H␣Slot␣$S␣Port␣$P”
        } else {
            puts ”——————————————————————————————————————————————————”
            puts ”␣Data:”
            puts ”␣Pack.Nr.:␣␣␣␣␣␣␣␣TX␣Time:␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣RX␣Time:␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣RX−TX:␣”
            for {set i 0} {$i < $recordsOnCard} {incr i} {
                LIBCMD HTGetStructure $::L3_HIST_RAW_TAGS_INFO $i 0 0 rt 0 $H $S $P
                set actLatency [expr ($rt(ulReceiveTime) − $rt(ulTransmitTime)) /10.0]
                    ###get the information for each frame
                puts −nonewline ”␣␣[expr␣$i+1]␣␣␣␣␣␣␣␣␣␣0x[format␣%X␣$rt(ulTransmitTime)]”
                    ### Print Number of the frame and timestamp when this frame left Smartbit
                puts −nonewline  ”␣␣␣␣␣␣0x[format␣%X␣␣$rt(ulReceiveTime)]␣␣␣␣␣␣␣”
                    ### Print timestamp when this frame was received by Smartbits
                puts ”␣␣$actLatency␣uS”
                    ### Print latency
                set avgLatency   [expr $avgLatency + $actLatency]
                if { $i == 0 || $minLatency > $actLatency } {
                    set minLatency $actLatency
                }
                if { $i == 0 || $maxLatency < $actLatency } {
                    set maxLatency $actLatency
                }
            }
            puts ”——————————————————————————————————————————————————”
            puts ”Average␣latency␣=␣[expr␣$avgLatency/␣(␣$recordsOnCard)]”
            puts ”Min␣latency␣=␣$minLatency␣”
            puts ”Max␣latency␣=␣$maxLatency␣”
        }
        puts ”Number␣of␣packets␣send:␣$numFrames”
        puts ”Number␣of␣packets␣received:␣$recordsOnCard”
        puts ”Lost␣packets:␣[expr␣$numFrames␣−␣$recordsOnCard]”
    }
################################################################################
### Returns numbers of records on card #########################################
    proc getRecord { H S P } {
        struct_new ActiveTestInfo Layer3HistActiveTest
        LIBCMD HTGetStructure $::L3_HIST_ACTIVE_TEST_INFO 0 0 0 ActiveTestInfo 0 $H $S $P
        set records_on_card $ActiveTestInfo(ulRecords)
        return $records_on_card
    }
################################################################################
## Returns number of streams on card ###########################################
    proc get_streamcount {H S P} {
        struct_new DefStreams  ULong
        LIBCMD HTGetStructure $::L3_DEFINED_STREAM_COUNT_INFO 0 0 0 DefStreams 0 $H $S $P
```

```
                return $DefStreams(ul)
        }
################################################################################
### Uses NSCalculateGap to set TX as Packets Per Second ########################
        proc setFramesPerSecond {H S P speed dataLength framesPerSecond} {
                set gap ””
                NSCalculateGap $::PPS_TO_GAP_BITS $speed $dataLength $framesPerSecond gap $H $S $P
                puts  ”_Gap_ist_$gap”
                return $gap
        }
################################################################################
### Uses NSCalculateGap to set TX as load percent ##############################
## Returns the GAP value needed to set the requested rate
        proc setLoad {H S P speed dataLength loadPercent} {
                set gap ””
                NSCalculateGap $::PERCENT_LOAD_TO_GAP_BITS $speed $dataLength $loadPercent gap $H $S $P
                return $gap
        }
################################################################################
################################################################################
################################################################################
##########################      MAIN PROGRAM     ################################
################################################################################
################################################################################
        linkSMB $ipAddr
                ### connect to SmartBits
        HGSetGroup ””
                ### clear groups
        HGAddtoGroup $iHub $iSlot $iPort
                ### add port 1 to the group
        HGAddtoGroup $iHub2 $iSlot2 $iPort2
                ### add port 2 to the group
        writeMII $iHub $iSlot $iPort $advRegisterInput $advertiseReg
                ### set auto negotiation Register
        writeMII $iHub2 $iSlot2 $iPort2 $advRegisterInput $advertiseReg
                ### set auto negotiation Register
        writeMII $iHub $iSlot $iPort $ctrlRegisterInput $controlReg
                ### set control register
        writeMII $iHub2 $iSlot2 $iPort2 $ctrlRegisterInput $controlReg
                ### set control register
        after 1000
                ### Wait 1000
        ## Specifies the interpacket gap that is to be transmitted on the addressed port
        LIBCMD HTGap $gap $iHub $iSlot $iPort
                ### or
        ###LIBCMD HTGap [setFramesPerSecond $iHub $iSlot $iPort $Speed2 $dataLength $framesPerSecond] $iHub $iSlot $iPort
                ### or
        ###LIBCMD HTGap [setLoad $iHub $iSlot $iPort $Speed2 $dataLength $loadPerCent] $iHub $iSlot $iPort
        puts ”Creating_a_stream”
        createStreamArray $iHub $iSlot $iPort $streamNumber
                ### create a stream
        create_extension $iHub $iSlot $iPort $framesPerSecond
                ### create stream extension
        puts ”Transmit_and_capture_the_stream!!”
        press2Continue
        resetCapture $iHub2 $iSlot2 $iPort2
                ### Restarts capture on target card to capture all received frames
        LIBCMD HTTransmitMode $SINGLE_BURST_MODE $iHub $iSlot $iPort
                ### Sets port to transmit a single burst of packets, then stop
        LIBCMD HTBurstCount $numFrames $iHub $iSlot $iPort
                ### Sets the number of packets to transmit in a single burst from a SmartCard
        resetRawtags $iHub2 $iSlot2 $iPort2
        LIBCMD HGStop
                ### Simultaneously halts the transmission of packets from all ports associated
                ### with the PortIDGroup defined by the previous HGSetGroup command
        LIBCMD HTSetCommand $::NS_CAPTURE_START 0 0 0 0 $iHub $iSlot $iPort
                ### Start capture
        LIBCMD HGRun $HTRUN
                ###Sets up the run state for all ports associated with the PortIDGroup
        after 100000
                ### WAIT
        LIBCMD HGRun $HTSTOP
                ###Sets up the run state for all ports associated with the PortIDGroup
        LIBCMD HTSetCommand $::NS_CAPTURE_STOP 0 0 0 0 $iHub $iSlot $iPort
                ### Stop capture
        displayRawtags $iHub2 $iSlot2 $iPort2 $numFrames
                ### display results
        ETUnLink
                ### disconnect from SmartBits
################################################################################
```

Appendix C

Smartbits Code Example 2

This is the code to generate one multicast stream (payload: 512 Bytes, throughput: 49.6 Mbps) with generating IGMP messages.

```
  ################################################################################
  ### Load smartlib.tcl ############################################################
        set libPath ”/usr/local/smartbits/smartlib/include/smartlib.tcl”
  ################################################################################
  ### Smartbits Configuration Port Address#######################################
        set ipAddr 10.0.0.2
  ################################################################################
  ### Port Configuration ##########################################################
        set iHub 0
        set iSlot 0
        set iPort 0

        set iHub2 0
        set iSlot2 0
        set iPort2 1
  #################################################################################
  ### set auto negoatiation advertisement register ################################
        set advRegisterInput    0x0080
        ### Technology ability : 100 Base−TX
  #################################################################################
        set Speed2  0x0008
  ### 100MegFull = 0x0101 ; 10MegFull 0x0041
  ### Speed2 100MHz: 0x0008 ; 10 MHz: 0x0004
  #################################################################################
  ####### set control register input ##############################################
        set ctrlRegisterInput      0xE410
  ### 0x1204 = on
  #################################################################################
  ### Numbers of Stream to create##################################################
        set streamNumber 1
  #################################################################################
  ### Numbers of Frames to send ###################################################
        set numFrames 56000

        set gap 4000
  #################################################################################
  ### Frames per Second ###########################################################
        set framesPerSecond 2000
  ### or ##########################################################################
  ### Load Percent (0 − 100) ######################################################
        set loadPerCent 60
  #################################################################################
  ### Set registeraddress #########################################################
        set controlReg       0
        ### control register address
        set advertiseReg     4
        ### auto negotiation advertisement register address
  #################################################################################
  ### Frame Length (no CRC) 0 − 2148 for Ethernet #################################
        set dataLength 559
  ################################################################################
### Load Smartlib ##############################################################
  ## if ”smartlib.tcl” is not loaded, try to source it from the default path
        if { ! [info exists __SMARTLIB_TCL__] } {
                if {[file exists $libPath]} {
                        source $libPath
                } else {
                        #Enter the location of the ”smartlib.tcl” file or enter ”Q” or ”q” to quit
                        while {1} {
                                puts ”Could␣not␣find␣the␣file␣$libPath.”
                                puts ”Enter␣the␣path␣of␣smartlib.tcl,␣or␣q␣to␣exit.”
                                gets stdin libPath
                                if {$libPath == ”q” || $libPath == ”Q”} {
                                        exit
                                }
                                if {[file exists $libPath]} {
```

```
                    source $libPath
                    break
                }

            }
        }
    }
############################################################################
### Link to Smartbits#######################################################
    proc linkSMB {ip} {
        if {[ETGetLinkStatus] < 1} {
            puts "SmartBits␣chassis␣IP␣address:␣$ip"
            NSSocketLink $ip 16385 $::RESERVE_ALL
        }
    }
############################################################################
###sets mii register to value of word#######################################
    proc writeMII {H S P word register } {
        set address [getMIIAddress $H $S $P]
        LIBCMD HTWriteMII $address $register $word $H $S $P
    }
############################################################################
############################################################################
    proc getMIIAddress {H S P} {
        set address ""
        set con_reg ""
        LIBCMD HTFindMIIAddress address con_reg $H $S $P
        return $address
    }
############################################################################
### simeple wait for user input routine#####################################
    proc press2Continue {} {
        puts "␣Press␣ENTER␣to␣continue"
        gets stdin response
    }
############################################################################
### create new Streams #####################################################
    proc createStreamArray {H S P numStreams} {
        struct_new ip StreamIP*$numStreams
        for {set i 0} {$i < $numStreams} {incr i} {
            set ip($i.ucActive)                         1
                ### ucActive: 1 = enable Stream ; 0 disable Stream
            set ip($i.ucProtocolType)               $::STREAM_PROTOCOL_TCP
            set ip($i.uiFrameLength)                $::dataLength
                ### Frame Length not counting CRC 0 -2148 for Ethernet
            set ip($i.ucTagField)                   1
                ### 0 = off ; 1 = insert signature into each frame
            set ip($i.Protocol)                         4
                ### 4 = ip on the IP assigned list
            set ip($i.TimeToLive)                   0xFF
            set ip($i.DestinationMAC.0)             0x01
                #PC1 0x00  Router1  0x00  Router2 0x00
            set ip($i.DestinationMAC.1)             0x00
                #PC1 0x11 Router1 0x15 Router2 0x15
            set ip($i.DestinationMAC.2)             0x5e
                #PC1 0x6B Router1 0xf9 Router2 0xf9
            set ip($i.DestinationMAC.3)             0x01
                #PC1 0x34 Router1 0x56 Router2 0x56
            set ip($i.DestinationMAC.4)             0x02
                #PC1 0x9A Router1 0xbe Router2 0xba
            set ip($i.DestinationMAC.5)             0x03
                #PC1 0x23 Router1 0xe0 Router2 0xd8
            set ip($i.SourceMAC.0)                  0x00
            set ip($i.SourceMAC.1)                  0x00
            set ip($i.SourceMAC.2)                  0x01
            set ip($i.SourceMAC.3)                  0x00
            set ip($i.SourceMAC.4)                  0x00
            set ip($i.SourceMAC.5)                  0x01
            set ip($i.DestinationIP.0)              224
            set ip($i.DestinationIP.1)              1
            set ip($i.DestinationIP.2)              2
            set ip($i.DestinationIP.3)              3
            set ip($i.SourceIP.0)                   192
            set ip($i.SourceIP.1)                   168
            set ip($i.SourceIP.2)                   3
            set ip($i.SourceIP.3)                   1
            set ip($i.Netmask.0)                    255
            set ip($i.Netmask.1)                    255
            set ip($i.Netmask.2)                    255
            set ip($i.Netmask.3)                    0
            set ip($i.Gateway.0)                    192
            set ip($i.Gateway.1)                    168
            set ip($i.Gateway.2)                    3
            set ip($i.Gateway.3)                    2
            set ip($i.ulARPGap)                         1000
                ### the time between ARPs in 100ns
            }
        LIBCMD HTSetStructure $::L3_DEFINE_IP_STREAM 0 0 0 ip 0 $H $S $P
    }
############################################################################
### create new Streams for receiver port2 ###################################
    proc createStream2Array {H S P numStreams} {
        struct_new ip StreamIP*$numStreams
        for {set i 0} {$i < $numStreams} {incr i} {
            set ip($i.ucActive)                         1
                ### ucActive: 1 = enable Stream ; 0 disable Stream
            set ip($i.ucProtocolType)               $::STREAM_PROTOCOL_TCP
```

```
            set ip($i.uiFrameLength)            $::dataLength
                ### Frame Length not counting CRC 0 −2148 for Ethernet
            set ip($i.ucTagField)               1
                ### 0 = off ; 1 = insert signature into each frame
            set ip($i.Protocol)                         4
                ### 4 = ip on the IP assigned list
            set ip($i.TimeToLive)               0xFF
            set ip($i.DestinationMAC.0)         0x01
                #PC1 0x00  Router1  0x00  Router2 0x00
            set ip($i.DestinationMAC.1)         0x00
                #PC1 0x11 Router1 0x15 Router2 0x15
            set ip($i.DestinationMAC.2)         0x5e
                #PC1 0x6B Router1 0xf9 Router2 0xf9
            set ip($i.DestinationMAC.3)         0x01
                #PC1 0x34 Router1 0x56 Router2 0x56
            set ip($i.DestinationMAC.4)         0x02
                #PC1 0x9A Router1 0xbe Router2 0xba
            set ip($i.DestinationMAC.5)         0x03
                #PC1 0x23 Router1 0xe0 Router2 0xd8
            set ip($i.SourceMAC.0)              0x00
            set ip($i.SourceMAC.1)              0x00
            set ip($i.SourceMAC.2)              0x04
            set ip($i.SourceMAC.3)              0x00
            set ip($i.SourceMAC.4)              0x00
            set ip($i.SourceMAC.5)              0x06
            set ip($i.DestinationIP.0)          224
            set ip($i.DestinationIP.1)          1
            set ip($i.DestinationIP.2)          2
            set ip($i.DestinationIP.3)          3
            set ip($i.SourceIP.0)               192
            set ip($i.SourceIP.1)               168
            set ip($i.SourceIP.2)               5
            set ip($i.SourceIP.3)               2
            set ip($i.Netmask.0)                255
            set ip($i.Netmask.1)                255
            set ip($i.Netmask.2)                255
            set ip($i.Netmask.3)                0
            set ip($i.Gateway.0)                192
            set ip($i.Gateway.1)                168
            set ip($i.Gateway.2)                5
            set ip($i.Gateway.3)                1
            set ip($i.ulARPGap)                         1000
                ### the time between ARPs in 100ns
            }
        LIBCMD HTSetStructure $::L3_DEFINE_IP_STREAM 0 0 0 ip 0 $H $S $P
    }
###############################################################################
### create Stream Extensions##################################################
    proc create_extension {H S P frameRate } {
        set numStreams [get_streamcount $H $S $P]
    struct_new L3X L3StreamExtension
        for {set index 1} {$index < $numStreams} {incr index} {
        set L3X(ulFrameRate)                        $frameRate
            ### frame transmit rate, in packets per second
        set L3X(ucIPHeaderChecksumError)        1
            ### 1 = enable ; 0 = disable
        set L3X(ucIPTotalLengthError)           1
            ### 1 = enable ; 0 = disable
        set L3X(ucCRCErrorEnable)                   1
            ### 1 = enable ; 0 = disable
        set L3X(ucDataCheckEnable)                  1
            ### 1 = enable pazload check error ; 0 = disable
        set L3X(ucDataIntegrityErrorEnable) 1
            ### 1 = enable ; 0 = disable
        set L3X(ulBGPatternIndex)                   0
            ### index of the stream background fill pattern
            ### that defined by L3StreamBGConfig
        set L3X(ucRandomBGEnable)                   1
            ###  1 = enable ; 0 = disable
        puts ”creating␣extension␣$index”
        LIBCMD HTSetStructure $::L3_MOD_STREAM_EXTENSION $index 0 0 L3X 0 $H $S $P
        }
    }
###############################################################################
###Restarts capture on target card to capture all received frames##############
    proc resetCapture {H S P} {
        struct_new CapSetup NSCaptureSetup
        set CapSetup(ulCaptureMode) $::CAPTURE_MODE_FILTER_ON_EVENTS
            ### Choose between:
            ### − CAPTURE_MODE_FILTER_ON_EVENTS
            ### − CAPTURE_MODE_START_ON_EVENTS
            ### − CAPTURE_MODE_STOP_ON_EVENTS
        set CapSetup(ulCaptureEvents) $::CAPTURE_EVENTS_ALL_FRAMES
            ### Choose between: see list
        LIBCMD HTSetStructure $::NS_CAPTURE_SETUP 0 0 0 CapSetup 0 $H $S $P
            ### Set structure
    }
###############################################################################
### reset Rawtags##############################################################
    proc resetRawtags { H S P} {
        LIBCMD HTSetCommand $::L3_HIST_RAW_TAGS 0 0 0 0 $H $S $P
    }
###############################################################################
### display the results#######################################################
    proc displayRawtags {H S P numFrames} {
        set actLatency 0
        set avgLatency 0
```

```
        set minLatency 0
        set maxLatency 0
        struct_new rt Layer3HistTagInfo
            ### rt contains the singature field information for the Raw Tag
            ###histogramm. A Separate record is generated for each SmartBit
            ###test frame (containing a signature) received at this port.
    set recordsOnCard [getRecord $H $S $P]
        ### get the number of histogram records captured on the target
        if {$recordsOnCard < 1} {
            puts "No␣records␣captured␣on␣Hub␣$H␣Slot␣$S␣Port␣$P"
        } else {
            puts "------------------------------------------------------------------------"
            puts "␣Data:"
            puts "␣Pack.Nr.:␣␣␣␣␣␣␣␣TX␣Time:␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣RX␣Time:␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣RX-TX:␣"
            for {set i 0} {$i < $recordsOnCard} {incr i} {
                LIBCMD HTGetStructure $::L3_HIST_RAW_TAGS_INFO $i 0 0 rt 0 $H $S $P
                set actLatency [expr ($rt(ulReceiveTime) - $rt(ulTransmitTime)) /10.0]
                    ###get the information for each frame
                puts -nonewline "␣␣[expr␣$i+1]␣␣␣␣␣␣␣␣␣␣0x[format␣%X␣$rt(ulTransmitTime)]"
                    ### Print Number of the frame and timestamp when this frame left Smartbit
                puts -nonewline  "␣␣␣␣␣␣0x[format␣%X␣␣$rt(ulReceiveTime)]␣␣␣␣␣␣␣"
                    ### Print timestamp when this frame was received by Smartbits
                puts "␣␣$actLatency␣uS"
                    ### Print latency
                set avgLatency  [expr $avgLatency + $actLatency]
                if { $i == 0 || $minLatency > $actLatency } {
                    set minLatency $actLatency
                }
                if { $i == 0 || $maxLatency < $actLatency } {
                    set maxLatency $actLatency
                }
            }
            puts "------------------------------------------------------------------------"
            puts "Average␣latency␣=␣[expr␣$avgLatency/␣(␣$recordsOnCard)]"
            puts "Min␣latency␣=␣$minLatency␣"
            puts "Max␣latency␣=␣$maxLatency␣"
        }
        puts "Number␣of␣packets␣send:␣$numFrames"
        puts "Number␣of␣packets␣received:␣$recordsOnCard"
        puts "Lost␣packets:␣[expr␣$numFrames␣-␣$recordsOnCard]"
    }
############################################################################
### Returns numbers of records on card ######################################
    proc getRecord { H S P } {
        struct_new ActiveTestInfo Layer3HistActiveTest
        LIBCMD HTGetStructure $::L3_HIST_ACTIVE_TEST_INFO 0 0 0 ActiveTestInfo 0 $H $S $P
        set records_on_card $ActiveTestInfo(ulRecords)
        return $records_on_card
    }
############################################################################
## Returns number of streams on card ########################################
    proc get_streamcount {H S P} {
        struct_new DefStreams ULong
        LIBCMD HTGetStructure $::L3_DEFINED_STREAM_COUNT_INFO 0 0 0 DefStreams 0 $H $S $P
        return $DefStreams(ul)
    }
############################################################################
### Uses NSCalculateGap to set TX as Packets Per Second #####################
    proc setFramesPerSecond {H S P speed dataLength framesPerSecond} {
        set gap ""
        NSCalculateGap $::PPS_TO_GAP_BITS $speed $dataLength $framesPerSecond gap $H $S $P
        puts  "␣Gap␣ist␣$gap"
        return $gap
    }
############################################################################
### Uses NSCalculateGap to set TX as load percent ###########################
## Returns the GAP value needed to set the requested rate
    proc setLoad {H S P speed dataLength loadPercent} {
        set gap ""
        NSCalculateGap $::PERCENT_LOAD_TO_GAP_BITS $speed $dataLength $loadPercent gap $H $S $P
        return $gap
    }
############################################################################
############################################################################
############################################################################
#########################      MAIN  PROGRAM      ##########################
############################################################################
############################################################################
    linkSMB $ipAddr
        ### connect to SmartBits
    HGSetGroup ""
        ### clear groups
    HGAddtoGroup $iHub $iSlot $iPort
        ### add port 1 to the group
    HGAddtoGroup $iHub2 $iSlot2 $iPort2
        ### add port 2 to the group
    writeMII $iHub $iSlot $iPort $advRegisterInput $advertiseReg
        ### set auto negotiation Register
    writeMII $iHub2 $iSlot2 $iPort2 $advRegisterInput $advertiseReg
        ### set auto negotiation Register
    writeMII $iHub $iSlot $iPort $ctrlRegisterInput $controlReg
        ### set control register
    writeMII $iHub2 $iSlot2 $iPort2 $ctrlRegisterInput $controlReg
        ### set control register
    after 1000
        ### Wait 1000
    ## Specifies the interpacket gap that is to be transmitted on the addressed port
```

```
      LIBCMD HTGap $gap $iHub $iSlot $iPort
      LIBCMD HTGap $gap $iHub2 $iSlot2 $iPort2

              ### or
      ###LIBCMD HTGap [setFramesPerSecond $iHub $iSlot $iPort $Speed2 $dataLength $framesPerSecond] $iHub $iSlot $iPort
              ### or
      ###LIBCMD HTGap [setLoad $iHub $iSlot $iPort $Speed2 $dataLength $loadPerCent] $iHub $iSlot $iPort
      puts "Creating␣a␣stream"
      createStreamArray $iHub $iSlot $iPort $streamNumber
      createStream2Array $iHub2 $iSlot2 $iPort2 $streamNumber

              ### create a stream
      create_extension $iHub $iSlot $iPort $framesPerSecond
      create_extension $iHub2 $iSlot2 $iPort2 $framesPerSecond

              ### create stream extension
      puts "Transmit␣and␣capture␣the␣stream!!"
      ##############NEW CODE ############################

      # Set L3 address on cards
      #############################
      puts "Setting␣L3␣Address"
      catch {unset L3Addr}
      struct_new L3Addr        Layer3Address
      set L3Addr(szMACAddress) {00 00 01 00 00 01}
      set L3Addr(IP) {192 168 3 1}
      set L3Addr(Gateway) {192 168 3 2}
      set L3Addr(PingTargetAddress) {192 168 3 2}
      set L3Addr(Netmask) {255 255 255 0}
      set L3Addr(iControl)  0
      set L3Addr(iPingTime) 0
      set L3Addr(iSNMPTime) 0
      set L3Addr(iRIPTime)  0
      LIBCMD HTLayer3SetAddress L3Addr $iHub $iSlot $iPort
      # set for second card
      set L3Addr(szMACAddress) {00 00 04 00 00 06}
      set L3Addr(IP) {192 168 5 2}
      set L3Addr(Gateway) {192 168 5 1}
      set L3Addr(PingTargetAddress) {192 168 5 1}


      LIBCMD HTLayer3SetAddress L3Addr $iHub2 $iSlot2 $iPort2
      unset L3Addr


      struct_new Init NSIGMPConfig
              set Init(ucVersion)  1
              set Init(ucOptions) $IGMP_INIT_ALWAYS_SEND_V2_LEAVE_REQUEST
              ##set Init(ucOptions) 0

      ##MD HTSetCommand $NS_IGMP_CONFIG 0 0 0 Init $iHub $iSlot $iPort
      LIBCMD HTSetCommand $NS_IGMP_CONFIG 0 0 0 Init $iHub2 $iSlot2 $iPort2

      catch {unset addr2}
      struct_new addr2 NSIGMPAddress
              set addr2(ucIPAddress) {224 1 2 3}
      ##BCMD HTSetCommand $NS_IGMP_JOIN 0 0 0 addr2 $iHub $iSlot $iPort
      LIBCMD HTSetCommand $NS_IGMP_JOIN 0 0 0 addr2 $iHub2 $iSlot2 $iPort2
      after 1000




     ##########################################################################
      press2Continue
      resetCapture $iHub2 $iSlot2 $iPort2
              ### Restarts capture on target card to capture all received frames
      LIBCMD HTTransmitMode $SINGLE_BURST_MODE $iHub $iSlot $iPort
              ### Sets port to transmit a single burst of packets, then stop
      LIBCMD HTBurstCount $numFrames $iHub $iSlot $iPort
              ### Sets the number of packets to transmit in a single burst from a SmartCard
      resetRawtags $iHub2 $iSlot2 $iPort2
      LIBCMD HGStop
              ### Simultaneously halts the transmission of packets from all ports associated
              ### with the PortIDGroup defined by the previous HGSetGroup command
      LIBCMD HTSetCommand $::NS_CAPTURE_START 0 0 0 0 $iHub $iSlot $iPort
              ### Start capture
      LIBCMD HGRun $HTRUN
              ###Sets up the run state for all ports associated with the PortIDGroup
      after 20000
              ### WAIT
      LIBCMD HGRun $HTSTOP
              ###Sets up the run state for all ports associated with the PortIDGroup
      LIBCMD HTSetCommand $::NS_CAPTURE_STOP 0 0 0 0 $iHub $iSlot $iPort
              ### Stop capture
      displayRawtags $iHub2 $iSlot2 $iPort2 $numFrames
              ### display results
      ETUnLink
              ### disconnect from SmartBits
#########################################################################################
```