A Performance Comparison of Native IP Multicast and IP Multicast Tunneled through a Peer-to-Peer Overlay Network

Marc Brogle, Dragan Milic, Luca Bettosini, Torsten Braun
Institute for Computer Science and Applied Mathematics
University of Bern, Neubrückstrasse 10, CH-3012 Bern, Switzerland
brogle|milic|bettosin|braun@iam.unibe.ch

***Abstract*—We have developed the Multicast Middleware, a bridge between IP Multicast and a self-organizing Overlay Multicast infrastructure, in order to make IP Multicast available to the end user. We compare the performance of native IP Multicast and IP Multicast tunneled through a P2P Overlay Multicast network using the Multicast Middleware. We show that the achievable throughput can be quite high when using Overlay Multicast for data transport together with the IP Multicast API and that the latency introduced by processing (en- and decapsulation) and tunneling the captured packets can be negligible.**

***Index Terms*—Multicasting, Overlay Multicast, IP Multicast, Performance, Overlay Networks**

I. Introduction

IP Multicast [1], [2] is an efficient way to disseminate data from a sender to multiple receivers concurrently. Unfortunately, IP Multicast has not been widely deployed in the Internet today, although several applications exist (such as Video Lan Client [3] for video broadcasting) that can use IP Multicast to transmit and receive data. To overcome this limitation and offer multicast services to end users in the future Internet, Overlay Multicast, also called Application Layer Multicast [4]–[6], which runs on-top of Peer-to-Peer (P2P) [7] Networks, has been introduced. Therefore, Overlay Multicast can be used as an efficient data dissemination scheme/mechanism for live video broadcasting, IPTV, multiplayer games, and other scenarios. Unfortunately, Overlay Multicast is not a standardized protocol, such as IP Multicast. Hence, different Overlay Multicast protocols exist.

In order to offer multicasting to end users, we developed the Multicast Middleware, which combines the IP Multicast API with a self-organizing Overlay Multicast network. We have chosen to use the Scribe [8], [9] Overlay Multicast facility, which runs on top of the the Distributed Hash Table (DHT) P2P Pastry [10], [11]. Pastry is a simple P2P routing substrate using a ring structure (one-dimensional torus), where routing is performed using Plaxton's method [12]. When peers join the network, they chose a random (uniformly distributed) and unique ID. On each hop from source to destination, one or more prefixes of the message's destination address are matched, while trying to minimize intermediate hop-delay. Peers know more peers in their ID neighborhood than peers further away on the ring. Scribe is a core based Overlay Multicast infrastructure, where multicast groups are called topics (topic IDs are from the same space as Pastry IDs) and the root for a topic is the Pastry peer numerically closest to the topic ID. Scribe builds multicast trees using Pastry's routing mechanism to discover a path to the corresponding root peer that is responsible for the topic. The reverse of this path is the dissemination path for the multicast messages.

Scribe/Pastry scales well for a large number of participants and multicast groups. A Java implementation in the form of Freepastry [13] exists, which we used in our Multicast Middleware prototype to build and manage the Overlay Multicast structure. For the actual multicast message dissemination, we used a self-developed and highly optimized protocol and mechanisms to efficiently replicate and forward the data, and therefore supporting high bandwidth scenarios. The Multicast Middleware is not limited to use Scribe/Pastry, but could also be adapted to use other P2P Overlay Multicast systems, such as multicasting extensions for Chord [14], [15], Bayeux [16] running on-top of Tapestry [17], and many others.

In this paper, we describe and discuss measurements to compare the performance of native IP Multicast and IP Multicast tunneled through Overlay Multicast. We used the SmartBits [18] "Portable High-density Network Performance Analysis System" to generate and capture the traffic for comparing latency and packet loss. Our measurements focus on the achievable throughput and the latency on end systems with acceptable packet loss for high-bandwidth scenarios, such as video broadcasting or IPTV, and for real-time and interactive applications, such as VoIP and multiplayer games.

The goal of this work is to answer the following question: is using IP Multicast tunneled through overlay networks a viable alternative to native IP Multicast? We investigate this in terms of additional jitter and delay introduced by using an overlay network. In this comparison, we also consider the impact of the packet size on the results. Compared to our previous work [19], we used Smartbits [18] to generate/capture the multicast traffic. This allows precise delay measurements using variable packet size, which was impossible in [19].

The remainder of the paper is structured as follows. The next Section describes the Multicast Middleware in more detail. Section III gives a short overview of the test scenarios and the different topologies used for the measurements. An evaluation of the measurement results is given in Section IV. Section V finally concludes this paper and gives an outlook to further possible analysis.

II. Multicast Middleware

We developed the Multicast Middleware in the scope of the European Sixth Framework Program Project EuQoS [20], [21]. The Multicast Middleware [19], [22] enables IP Multicast for

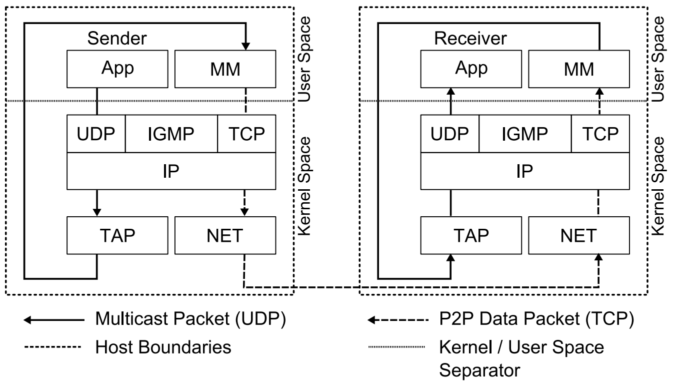


Fig. 1. Packet Flow between Applications and the Multicast Middleware

end users, by offering the IP Multicast API and by tunneling multicast data through an Overlay Multicast facility. The process is completely transparent to applications. Figure 1 shows how applications send and receive IP Multicast traffic. A virtual network interface TAP [23] captures/injects these IP Multicast packets at end systems. It then passes/receives IP Multicast data to/from the Multicast Middleware, which disseminates the packets among the group subscribers using a Peer-to-Peer (P2P) overlay network. Furthermore, users can define Quality of Service (QoS) requirements for the different IP Multicast groups they are interested in. QoS reservations are performed in the underlying QoS-enabled network and the Overlay Multicast distribution tree in the P2P network is set up as described in [19].

We only use Scribe/Pastry to create the overlay structure through which captured IP Multicast data is transmitted among the receivers using our own P2P Overlay Multicast data transport protocol, which is optimized for high-bandwidth and low-latency scenarios. Typically, P2P Overlay Multicast protocol implementations (e.g. Freepastry) are not optimized for this purpose. The P2P Overlay Multicast infrastructure or strategy to build the overlay used by the Multicast Middleware can though be easily exchanged with other implementations and protocols.

III. Experimentation and Measurement Scenarios

A. Topologies used for Measurements

We have conducted performance measurements with several chain and tree topologies. In this paper we will look at a chain topology consisting of four hosts and a tree topology with the maximum tree depth of four. Each host has an out-degree of one in the chain scenario, so no IP Multicast packets need to be duplicated but just need to be forwarded. In the tree scenario, three hosts have to duplicate the data, and therefore having an out-degree of two. The remaining four hosts in the tree scenario have an out-degree of one. The tree scenario represents a small sub-tree of a big overlay scenario, where typically the out-degree of hosts is between 0 to 2. We used traffic characteristics reflecting a constantly sending source at various fixed rates (1.0, 5.0, 10.0, 24.8, 49.6, 75.2 and 84.8 Mbps) with packet payload sizes of 32, 512 and 1024 bytes. The traffic was generated and measured using the SmartBits, which allows more accurate measurements than a software based solution running on a Linux machine as performed in earlier measurements. The Linux routers in-between were running no background processes, their full system performance was available for IP Multicast routing in the native IP Multicast measurements, and for the Multicast Middleware packet capturing/processing and overlay network forwarding in the Overlay Multicast measurements.

The different topologies used for the performance measurements are depicted in Figures 2 and 3. The chain topology in Figure 2 consists of the Smartbits (traffic generation and capture) and four PCs acting as Linux multicast routers connected in a chain.

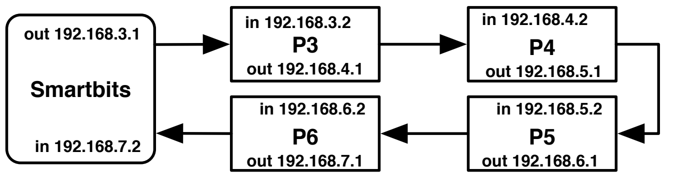


Fig. 2. The Chain Topology used for the Measurements

The tree topology in Figure 3 consists of the Smartbits (traffic generation and capture) and seven PCs acting as Linux multicast routers. They are connected such that the longest branch of the tree consists also of four PCs, but the branching nodes have to additionally duplicate the packets and send them to two receivers.

We wanted to accurately measure and compare the latency introduced by processing and tunneling the IP Multicast data through Overlay Multicast and also wanted to determine the maximum achievable throughput, the packet loss and delays using the Multicast Middleware in different topologies. Unfortunately, there are no such large pure IP Multicast networks available in the Internet to support these high-bandwidth measurements. Additionally, the measurements to determine the introduced delay as described above would not be possible in the Internet. Therefore, we also could not use the PlanetLab environment, which would introduce additional overhead and delays through the virtual machines used on PlanetLab nodes. Hence, we performed the measurements locally in a fully controllable laboratory environment.

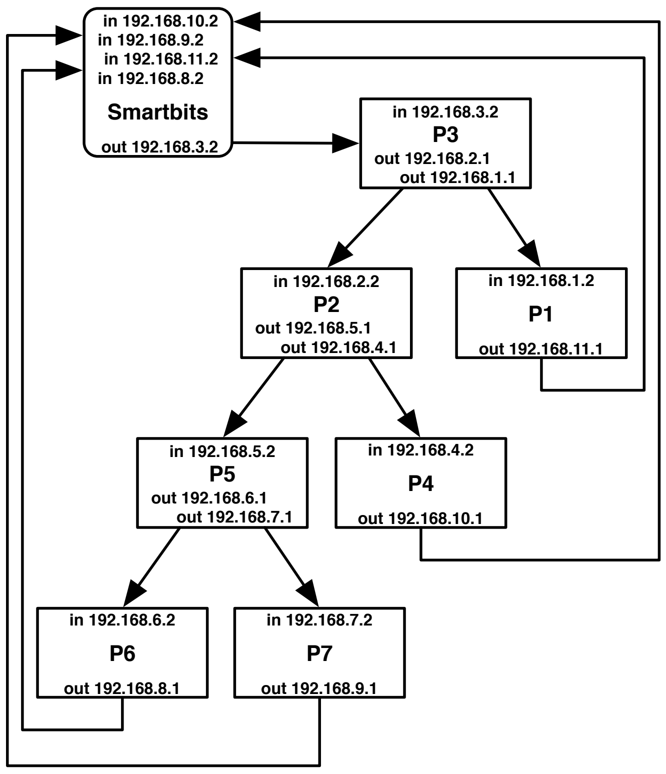


Fig. 3. The Tree Topology used for the Measurements

The computers acting as routers in the test environment had an Intel Pentium IV 3.0 GHz CPU with 1 MB Cache, two times 512 MB Take M5 DDR 400 CL 2.5 RAM, P4S800-MX mainboards with BIOS rev. 0501, one SiS 900/7016 100 Mbps onboard network adapter, two Realtek RTL-8169 Gigabit LAN network cards, and one Hitachi Deskstar 7K80 80GB HDS728080PLAT20 hard drive. The operating system used was Fedora Core 5 with Linux kernel 2.6.20-1.2307. To configure IP Multicast routing on Linux, we used SMCRoute [24] version 0.92. SMCRoute is a combination of a daemon and command line tool, which allows to setup static IP Multicast routes for the different network interfaces of the Linux routers. It is similar to mrouted [25], with the difference that it supports static instead of dynamic multicast routes, which is what we needed for our experiment setup, since we wanted to maintain static multicast routes.

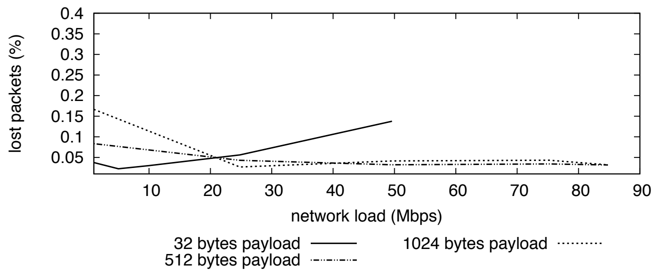
Previous measurements [19] using two and five host chain scenarios showed that the Multicast Middleware running on a similar hardware is able to process roughly 100 Mbps of total network traffic (incoming, duplicating and outgoing) without any packet drops. Those previous measurements were performed using the Multi-Generator (MGEN) [26] software to generate and capture the traffic rather then using the more sophisticated traffic measurement system SmartBits used for the measurements presented in this paper. In those previous measurements, we used a single Dual-Core Pentium D 3.2 GHz CPU system for generating, capturing and forwarding the data. Since two different hosts were used to generate and capture the traffic, delay measurements as presented in this paper were not possible. This is due to the fact that it is very hard to synchronize the clock accurately between two independent systems. Using SmartBits allows us to perform delay specific measurements and also allows us to accurately generate traffic with smaller packet sizes at high bandwidth, which would not be possible with software based systems.

B. Performed Measurements

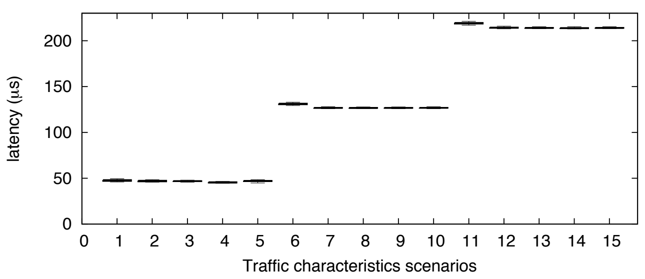
We defined three different packet payloads in order to analyze the differences between native IP Multicast and our Overlay Multicast solution. For each of the different packet payloads, we performed the measurements with five different network load values. The nature of the Smartbits API restricted us to use the inter-packet gap instead of directly the bandwidth as a parameter. The different payload sizes, packet inter-arrival delay, resulting bandwidth and number of packets being sent are shown in Table I. The row labeled *Payload* denotes the actual payload size in a data packet, whereas values in the row *Data* show the overall packet size (including IP header). The *Packet gap* value is the time in microseconds between the moment a packet has been completely sent and when the next packet starts being sent. The number of packets sent in a scenario is shown in the *# packet* row. The total number of packets that the Smartbits can process is limited. Due to this limitation, we did not send more than 130000 packets in any of the scenarios. Therefore, the last two scenarios with a payload of 32 bytes had a runtime of less than half as long as the other scenarios. The combination of *Data* and *Packet gap* defines the desired *bandwidth* for the scenarios. We used different payload sizes of 32, 512 and 1024 bytes with bandwidth values of 1.0, 5.0, 10.0, 24.8, 49.6, 75.2 and 84.8 Mbps. A smaller packet size resulted in more packets being sent and processed. This allows us to determine the limit of successfully processable packets for the different topologies. For the payload size of 32 bytes, we limited the bandwidth to 49.6 Mbps.

TABLE I
TRAFFIC CHARACTERISTICS SCENARIOS

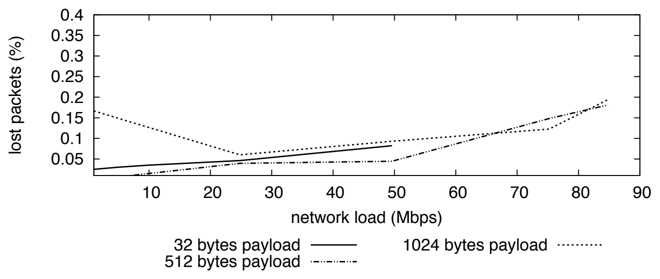
No.	Payload (bytes)	Data (bytes)	Bandw. (Mbps)	Packet gap (bit time)	# packets
1	32	79	1.0	61000	8000
2	32	79	5.0	11600	40000
3	32	79	10.0	5500	80000
4	32	79	24.8	1720	130000
5	32	79	49.6	485	130000
6	512	559	1.0	430000	1200
7	512	559	24.8	12500	28000
8	512	559	49.6	4000	56000
9	512	559	75.2	1100	85000
10	512	559	84.8	450	95000
11	1024	1071	1.0	830000	600
12	1024	1071	24.8	24000	15000
13	1024	1071	49.6	7700	29000
14	1024	1071	75.2	2250	44000
15	1024	1071	84.8	960	50000



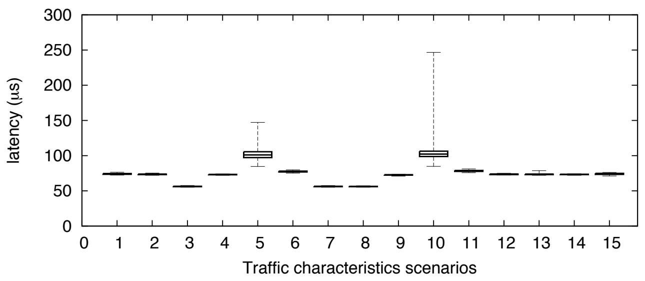
(a) Packet Loss for IP Multicast using Chain Topology



(b) Latency w/o 5% Outliers for IP Multicast using Chain Topology



(c) Packet Loss (P6) for IP Multicast using Tree Topology



(d) Latency w/o 5% Outliers (P6) for IP Multicast using Tree Topology

Fig. 4. Measurement Results for Native IP Multicast using Chain and Tree Topologies

IV. Evaluation

A. Native IP Multicast

The measurements with the chain topology in Figure 4(a) show a very small packet loss below 0.2%. Most of the packets are lost in case of a packet payload of 32 bytes and higher bandwidth values. There is a high loss for high packet rates (small packet payload and high bandwidth). For a payload of 1024 bytes, the packet loss rate starts just below 0.2% and then falls down towards 0% for higher bandwidth values. This is due to the fact, that for low bandwidth values, only a few hundred packets are being sent and a single dropped packet has a significant impact on the percentage value, whereas for higher bandwidth values, many thousand packets are being sent, therefore reducing the impact of a single dropped packet on the percentage value.

The latencies for the chain topology are shown in Fig. 4(b). The network load has almost no influence on the latency, with negligible differences. It seems that SMCRoute is buffering packets before forwarding packets to a host. Therefore, the delay increases as the payload size increases.

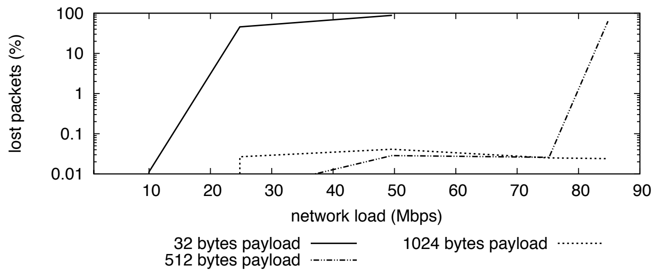
For the tree topology, we captured the data on the different lengths of the tree. The packet losses for P1, P4, P6 and P7 (leaf hosts) were equal and therefore we only show the packet loss for P6 in Fig. 4(c). Compared to the chain topology, packet loss does not change significantly, although for configurations with higher payload, the packet loss increases a little. The drop rate behavior for low bandwidth values and for a payload of 1024 bytes is as described in the chain topology.

The latency has a different behavior in the tree topology as shown in Fig. 4(d). It seems that SMCRoute does not buffer packets when it has to duplicate and forward them to multiple hosts. Hence, the average delay is now similar for different payload values. In the chain topology, the delay varied depending on the payload size due to the impact of buffering performed by SMCRoute. We also have two configurations (traffic characteristics 5 and 10) with larger jitter than the average. It is clearly visible that the delay values measured have a much higher variance for those two configurations. For such configurations with a small payload and a high network load, the jitter seems to increase. This behavior is 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, so they get queued.

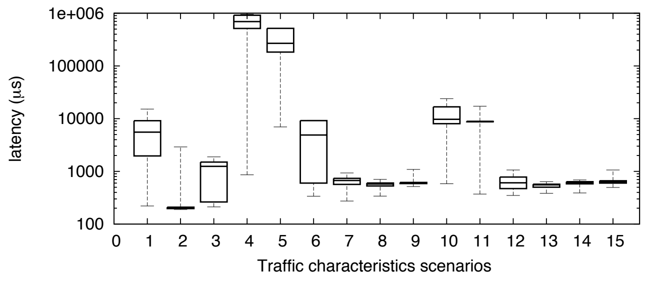
B. IP Multicast Tunneled through an Overlay Network

Our measurements show that the Multicast Middleware generally handles packet sizes of 512 and 1024 bytes very well, while the smaller packet size of 32 bytes together with high bandwidth lead to a high packet drop rate. Figure 5(a) shows that packet loss in the chain topology for 1024 bytes packet payload is between 0% and 0.04%. Packet drops occur when the Multicast Middleware has to process many packets in a short time, as it is the case for 32 bytes packet payload with more than 10 Mbps and 512 bytes packet payload with 75.2 Mbps. Therefore, the Multicast Middleware seems to be able to process around 18000 incoming packets per second in the chain scenario before it starts to drop packets. The Multicast Middleware has to process two packet streams in the chain topology, the incoming and the outgoing stream.

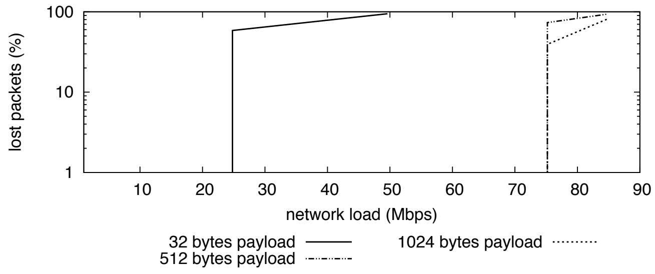
The latency measurements for the chain topology using Overlay Multicast are presented in Figure 5(b). The delays are acceptable (meaning below 10 ms) for bandwidth values of 10 Mbps and below and a payload of 32 bytes. For 512 and 1024 bytes payloads, the latencies are acceptable for all



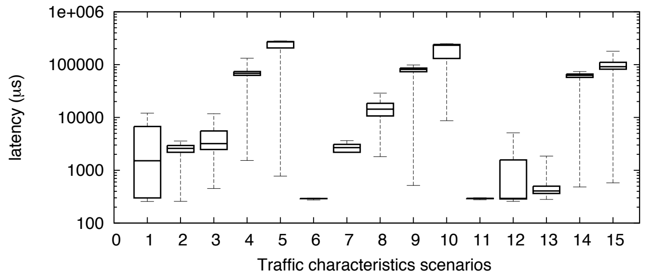
(a) Packet Loss for Overlay Multicast using Chain Topology



(b) Latency w/o 5% Outliers for Overlay Multicast using Chain Topology



(c) Packet Loss (P6) for Overlay Multicast using Tree Topology



(d) Latency w/o 5% Outliers (P6) for Overlay Multicast in Tree Topology

Fig. 5. Measurement Results for IP Multicast Tunneled through Overlay Multicast using Chain and Tree Topologies

bandwidth values used for the measurements. The latencies between 1 to 10 Mbps are decreasing for increasing bandwidth. This is due to the multi-threaded design of the Multicast Middleware. Each overlay connection in the Multicast Middleware is handled by a separate thread (light-weight process). This thread reads data from a ring buffer and waits on a conditional variable if the buffer is empty. At high packet rates, this thread never waits. At low packet rates though, the thread waits after transmitting each packet. Waking up this thread produces additional latency in packet transmission. Therefore, scenarios 1 to 3 result in higher latency for lower packet rates. The total number of waiting states per second is lower when the packet size is being increased. Therefore, the impact of multithreading issues on the overall latency is reduced. This explains the smaller additional latency values in scenarios 6 to 8 and 11 to 13.

The results obtained from the presented measurements in the chain topology using a packet payload of 1024 bytes correspond to the previously determined bandwidth limit of 100 Mbps in [19] with acceptable packet loss.

The tree topology packet loss results presented in Fig. 5(c) show that packet loss is at an unacceptable level for 10 Mbps and more with 32 bytes packet payload, above 24.6 Mbps for 512 bytes packet payload and more than 49.6 Mbps for 1024 bytes packet payload. In the tree topology, the Multicast Middleware seems to be able to process at least 9000 incoming packets per second, before it starts to drop packets. In the tree topology, the Multicast Middleware has to process three packet streams, one incoming, and two outgoing.

The latency measurements presented in Fig. 5(d) lead us to the same conclusion. The tree topology measurements show acceptable delays (below 10 ms) for bandwidth values below 10 Mbps for 32 bytes payload packets, below 24.8 Mbps for 512 bytes packet payload and below 75.2 Mbps for a payload of 1024 bytes. Our measurements show that the Multicast Middleware can process incoming packet flows in the chain topology up to 75.2 or 84.8 Mbps with a payload of 512 or 1024 bytes, respectively.

The limit is above 10 Mbps in the chain topology for packet payloads of 32 bytes. For the tree topology, incoming flows up to 10 Mbps for 32 bytes payload and up to 24.8 Mbps for a payload of 512 bytes can be processed, while for a packet payload of 1024 bytes, incoming flows up to 49.6 Mbps can be handled. These results show that the Multicast Middleware can be used for high-bandwidth scenarios and that processing the packets introduces acceptable additional delays. The Overlay Multicast latency measurements that were still acceptable are an order of a magnitude higher than with native IP Multicast. But average delays around 0.7 ms are still acceptable for the four hop chain scenario. In the tree topology, the average still usable delays are around 3 ms, which is still acceptable.

C. Comparison between Native IP Multicast and IP Multicast Tunneled through an Overlay Network

The loss rate in the chain topology does not behave very differently between native IP Multicast and IP Multicast tunneled through an overlay network for a packet payload size of 1024 bytes. Also, for a packet payload size of 512 bytes, the values are in the same area for the different bandwidth values up to 75.2 Mbps. Above this bandwidth, the Overlay Multicast scheme starts to have high losses. For the small packet payload of 32 bytes, both scenarios behave quite similar up to a bandwidth of 10 Mbps, then the native IP Multicast scenario gets only slightly worse with raising bandwidth, whereas the

Overlay Multicast scenario has high losses with bandwidth values above 10 Mbps. Delays in the Overlay Multicast scenarios are much higher than for native IP Multicast, due to capturing, processing, replication and injection of packets using the Multicast Middleware and the overlay network. But we can see that this approach is valid because the introduced delay for most of the bandwidth and packet size scenarios is tolerable as described in Section IV-B.

In the tree scenarios, Overlay Multicast compared to native IP Multicast behaves almost the same, having the bandwidth limit for Overlay Multicast below 24.8 Mbps for packet payload size of 32 bytes and below 75.2 Mbps for payload sizes of 512 and 1024 bytes. The impact of capturing, processing, replication and injection packets by the Multicast Middleware on the delays in the tree scenarios is the same as for the chain scenario. Compared to native IP Multicast, the delays are higher, but behave less steady and raise if the bandwidth is augmented but are still tolerable as described before for many packet payload size and bandwidth combinations.

V. Conclusion

In this paper we compared native IP Multicast and Overlay Multicast (also called Application Layer Multicast) regarding performance. Our Overlay Multicast solution, called Multicast Middleware, is able to bridge IP Multicast with Overlay Multicast. It allows to efficiently disseminate multimedia data for video broadcasting, IPTV, VoIP and multiplayer games. The IP Multicast traffic was generated and measured using a network performance analysis system called Smartbits. Compared to previous measurements, we were now able to make more accurate and delay-related measurements. We concentrated our measurements on local networks to analyze the maximum throughput and the delays introduced by processing the packets, which would not have been possible in a distributed network over the Internet on a larger scale. We performed the measurements in different topologies (chain and tree) with variable network load and payload. The results of the measurements show that the Multicast Middleware can process around 18000 incoming packets per second in the chain topology and at least 9000 incoming packets per second in the tree topology before it starts to drop packets at a significant level. Incoming streams up to 75.2 Mbps with a packet payload size of 1024 bytes can be supported in the chain topology, whereas at least incoming streams with 49.6 Mbps can be supported in the tree topology with the same payload.

Acknowledgments

This work was partly funded by the European Union 6th Framework Programme under contract IST FP6 IP 004503 EuQoS Integrated Project.

References

[1] S. Deering, "Host extensions for IP multicasting," RFC1112, August 1989.

[2] B. Cain, S. Deering, I. Kouvelas, B. Fenner, and A. Thyagarajan, "Internet Group Management Protocol, Version 3," RFC3376, October 2002.

[3] Website, "Video Lan Client," Available online: http://www.videolan.org, 2009.

[4] M. Hosseini, D. T. Ahmed, S. Shirmohammadi, and N. D. Georganas, "A survey of application-layer multicast protocols," *Communications Surveys & Tutorials, IEEE*, vol. 9, no. 3, pp. 58–74, 2007.

[5] S. Fahmy and M. Kwon, "Characterizing overlay multicast networks," in *Network Protocols, 2003. Proceedings. 11th IEEE International Conference on*, 2003.

[6] Y. hua Chu, S. G. Rao, S. Seshan, and H. Zhang, "A Case for End System Multicast," 2002.

[7] K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim, "A survey and comparison of peer-to-peer overlay network schemes," *Communications Surveys & Tutorials, IEEE*, pp. 72–93, 2005.

[8] A. Rowstron, A.-M. Kermarrec, M. Castro, and P. Druschel, "Scribe: The design of a large-scale event notification infrastructure," in *Networked Group Communication, Third International COST264 Workshop (NGC'2001)*, ser. Lecture Notes in Computer Science, vol. 2233, November 2001, pp. 30–43.

[9] M. Castro, P. Druschel, A. M. Kermarrec, and A. I. T. Rowstron, "Scribe: a large-scale and decentralized application-level multicast infrastructure," *Selected Areas in Communications, IEEE Journal on*, vol. 20, no. 8, pp. 1489–1499, 2002.

[10] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," in *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, November 2001, pp. 329–350.

[11] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron, "One ring to rule them all: Service discover and binding in structured peer-to-peer overlay networks," in *SIGOPS European Workshop*, September 2002.

[12] C. G. Plaxton, R. Rajaraman, and A. W. Richa, "Accessing nearby copies of replicated objects in a distributed environment," in *SPAA '97: Proceedings of the ninth annual ACM symposium on Parallel algorithms and architectures*. New York, USA: ACM, 1997, pp. 311–320.

[13] Website, "Freepastry," Available online: http://freepastry.rice.edu/, 2009.

[14] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, F. M. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: a scalable peer-to-peer lookup protocol for internet applications," *IEEE/ACM Trans. Netw.*, vol. 11, no. 1, pp. 17–32, February 2003.

[15] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*. New York, NY, USA: ACM Press, 2001, pp. 149–160.

[16] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. D. Kubiatowicz, "Bayeux: an architecture for scalable and fault-tolerant wide-area data dissemination," in *NOSSDAV '01: Proceedings of the 11th international workshop on Network and operating systems support for digital audio and video*. New York, NY, USA: ACM Press, 2001, pp. 11–20.

[17] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph, "Tapestry: An infrastructure for fault-tolerant wide-area location and routing," Tech. Rep., 2001.

[18] Sirpent, "Portable high-density network perf. analysis system: Smartbits 600b," Available online: http://www.spirent.com/documents/1374.pdf, 2009.

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

[20] Website, "EuQoS project," Available online: http://www.euqos.eu, 2009.

[21] T. Braun, M. Diaz, J. Enrquez-Gabeiras, and T. Staub, *End-to-End Quality of Service Over Heterogeneous Networks*. Springer, 2008.

[22] D. Milic, M. Brogle, and T. Braun, "Video broadcasting using overlay multicast," in *ISM '05: Proceedings of the Seventh IEEE International Symposium on Multimedia*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 515–522.

[23] M. Krasnyansky and M. Yevmenkin, "Universal TUN/TAP driver," Available online: http://vtun.sourceforge.net/tun/, 2009.

[24] Website, "Smcroute," Available online: http://www.cschill.de/smcroute/, 2009.

[25] B. Fenner and et. al, "mrouted 3.9-beta," Available online: ftp://ftp.parc.xerox.com/pub/net-research/ipmulti/, 2009.

[26] U. N. R. Laboratory, "Multi-generator (MGEN)," Available online: http://cs.itd.nrl.navy.mil/work/mgen/, 2009.