A REAL-TIME INTERNET STREAMING MEDIA TESTBED

Wolfgang Kellerer, Eckehard Steinbach, Peter Eisert, and Bernd Girod

Information Systems Laboratory
Department of Electrical Engineering
Stanford University
{kellerer,steinb,eisert,bgirod }@stanford.edu

ABSTRACT

In this paper we describe a real-time LAN-based testbed that allows us to investigate the behavior of streaming media applications under various network conditions. For commercially available streaming media applications, we are interested to see how they perform over next-generation wireline and wireless networks. For future streaming media applications, the testbed is an invaluable tool for the development and verification of new algorithms. Our testbed implementation is based on *Linux Divert Sockets* and supports a straightforward integration of various packet erasure and delay models. Individual IP-packets are diverted to a user process where they are delayed or deleted according to the desired channel model. The testbed has been used to investigate the flow-control behavior of existing streaming media systems over wireless networks. Our experiments confirm that flow-control algorithms that consider lost packets to be the result of network congestion, as employed today in wireline streaming, are not suited for wireless networks, where loss is mainly due to link impairments.

1. INTRODUCTION

Increasing transmission rates on access networks and improved source coding techniques have helped to make media streaming a widely used application. Even on low data rate channels, like modem connections, we can nowadays receive acceptable video quality. But, like most Internet applications and protocols, also media streaming applications have been optimized for wireline networks. Wireless transmission channels still provide a major challenge for streaming media due to their error characteristics. Instead of packet losses caused by congestion, in wireless networks random transmission errors prevail.

To develop and verify appropriate algorithms for wireless streaming media, extensive tests using various loss patterns are required. Since those networks are either in field trial or still under development, tests are extremely complex and costly. Off-line simulation is not sufficient. Interactivity, in particular, can only be studied when tests are performed in a real-time environment.

It is widely accepted that future mobile communication infrastructure will be all IP, e.g., [1]. This allows us to use our office LAN to study the behavior of wireless streaming media applications. By incorporating special network nodes in our LAN that apply predetermined loss and delay patterns, streaming media over arbitrary wireless transmission environments can be investigated.

This work was performed when Wolfgang Kellerer was a visitor at Stanford University, while on leave from the Institute of Communication Networks of Munich University of Technology. He is now with DoCoMo Euro-Labs, Munich. Eckehard Steinbach is now with Munich University of Technology. Peter Eisert is now with the Heinrich-Hertz-Institute, Berlin.

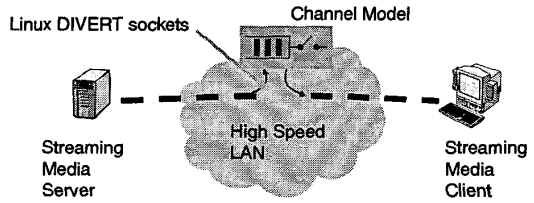

Figure 1: Testbed architecture.

In this work, we present such a LAN-based testbed that is based on *Linux Divert Sockets* [2]. Our testbed implementation modifies the IP-packet flow according to certain channel models. For this, IP-packets are diverted to a user process where corresponding packet loss and delay patterns are applied. Packet filtering mechanisms allow to modify individual IP-streams and transport protocols. Those are selected by specification of firewalling rules.

The testbed setup is particularly simple and requires in its most general form only one additional network node. Existing client and server software can be used without modification to test media streaming products. In case the operating system of the client or server is Linux, no additional network node is required. The packet diversion mechanism then runs on the client or server machine itself.

We have structured this paper as follows. In Section 2, we first present the testbed design and architecture. We describe details about the implementation in Section 3. The channel models currently supported in our implementation are discussed in Section 4. In Section 5, a sample experiment illustrates how the testbed can be used to study the behavior of commercially available streaming media systems. The experiment demonstrates the unsuitability of wireline flow control mechanisms when deployed in wireless transmission environments.

2. ARCHITECTURE

Figure 1 shows the principle architecture of our real-time testbed. A streaming client is interconnected with a streaming server via a high speed LAN. A network component that is traversed by the exchanged packets modifies the packet flow according to a certain channel model. We have selected *Linux Divert Sockets* to perform packet interception and reinjection.

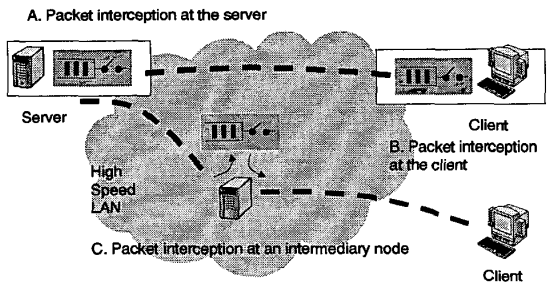

Figure 2: Testbed architecture variations.

For this general setup the operating system of the server and the client machine does not have to be Linux. In case either server or client are Linux-based, the following variations, illustrated in Figure 2, are possible.

A. Packet interception and reinjection are performed on the Linux-based server machine. No intermediate node is required.

B. Packet interception and reinjection at the client. Again, no intermediate node is required.

C. Packet interception and reinjection are performed on an intermediary node. This makes the testbed independent of the server or client operating system. This alternative corresponds to Figure 1.

In comparison to other architectures that support real-time testing, e.g. [3], our architecture is transparent for addressing. This means, our architecture does not require any modification of the IP-header or recalculation of the checksum.

3. TESTBED IMPLEMENTATION

A key feature of the testbed is the controlled interception and reinjection of IP-packets. Linux Divert Sockets [2] enable IP-packet interception and injection on end systems as well as on routers. Packets are intercepted on an IP-layer and are available for user processes outside the kernel via a modified version of raw sockets (see Figure 3). Intercepted packets will not be able to reach their destination unless they are reinjected. The same sockets can be used for interception and for reinjection.

For packet filtering, Divert Sockets rely on the IP firewall mechanism. Instead of specifying a firewall target like ACCEPT or DENY, the filtered packets can be sent to a Divert Socket. Divert Sockets can be bound to a port. In this way multiple user processes can listen to different firewall filters.

The Linux firewalling mechanism *ipchains* is used to set up, maintain, and inspect the IP firewall rules in the kernel. These rules can be divided into four different categories: the IP input chain, the IP output chain, the IP forwarding chain, and user defined chains. A chain specifies a trigger point an IP-packet passes on its way through the kernel. Thus, e.g., packets that arrive at the IP-layer from outside can be intercepted using the input chain. A rule is appended to a chain and it specifies predicates, i.e., filter expressions. A chain may have many rules. In addition to filter expressions, each rule specifies a target that defines the action that is appended for a packet that matches a filter expression. Typical IP-firewall targets are ACCEPT or DENY. The modified version

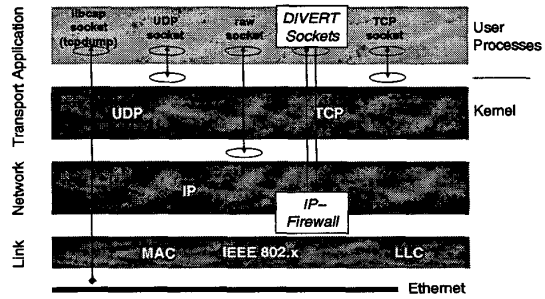

Figure 3: Linux Divert Sockets.

[2] of ipchains used in this work adds the target DIVERT. Figure 4 shows the various chains and the interworking with Divert Sockets.

In comparison to other socket mechanisms for IP-packet handling, Divert Sockets have special advantages that fit our requirements (cf. Table 1). *DummyNet* [4], which has been developed for real-time testing of networking protocols, supports the modification of IP-packet flow within the kernel and not in a user process. Therefore, it comes with pre-installed channel models. DummyNet always requires the use of an intermediary node (router). Netlink sockets are similar to Divert Sockets but lack the specification of ports. The convenient realization of packet filtering is missing in pure raw sockets [5]. Tcpdump does neither include filtering nor a packet reinjection mechanism.

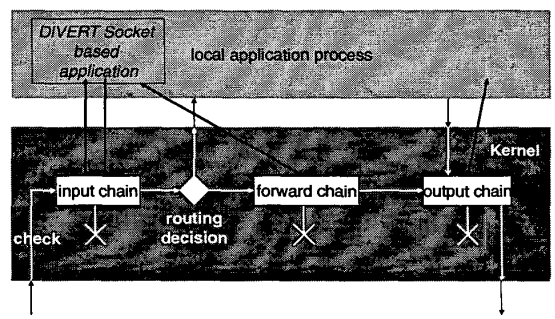

Figure 4: Interworking with IP firewall chains.

4. CHANNEL MODELS

So far, we have implemented and tested three different packet erasure and delay models, which are described in the following.

4.1. Constrained packet-rate model

The first model, illustrated in Figure 5, realizes a bottleneck link where the maximum packet transmission rate is limited. Incoming packets are queued in a packet buffer of fixed size. The queue is served with the deterministic service rate $\lambda_{channel}$. By modifying $\lambda_{channel}$, the maximum number of packets per second transmitted by the link is controlled. Buffer overflow leads to lost packets. This channel model approximates the behavior of a network router connected to a low data rate link.

Feature	Divert	DummyNet	Netlink	raw socket	libcap/ tcpdump
packet sniff	•	–	•	•	•
filter	local IP	local IP	local IP	local IP	segment
firewall filter	•	•	•	–	–
interception	•	•	•	•/–	–
support ports	•	–	–	–	–
processing	user program	kernel	user program	user program	display
injection	in-/outbound	in-/outbound	in-/outbound	outbound	–
operating system	Linux, FreeBSD	FreeBSD	Linux,...	all	all

Table 1: Comparison of different socket mechanisms

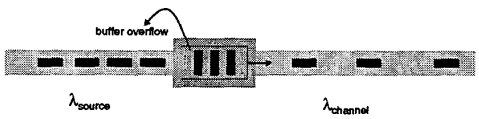

Figure 5: Low data rate channel model.

4.2. Packet burst loss model

The channel model shown in Figure 6 is a two-state Markov model also called Gilbert-Elliot packet erasure channel [6]. This channel is assumed to have memory which results in burst packet losses. The mean length of these burst losses is determined by the state transition probabilities [6]. In the state "Good", packets are assumed to be received correctly while in the state "Bad" packets are assumed to be lost.

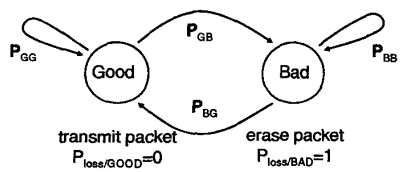

Figure 6: Packet burst loss channel model.

This channel model approximates a typical wireless transmission link where channel outages due to fading or frequent disconnections due to handover cause loss of successive packets. The Gilbert-Elliot packet erasure model is also often used to approximate congestion situations in wireline networks [7], [8].

4.3. Markov modulated Poisson model

The third channel model assumes link layer protection on a wireless transmission link. Link layer protection, e.g., by retransmission, basically provides an error free transmission of packets at the cost of highly variable delay.

We approximate this behavior using the two-state Markov Modulated Poisson Process in Figure 7. Again, the transmission quality is described using two states, a good state with high goodput and a bad state with reduced goodput. In comparison to the previous channel model, the interarrival times of packets in either state are drawn from a negative exponential distribution with the mean arrival rates λ_{Good} and λ_{Bad}.

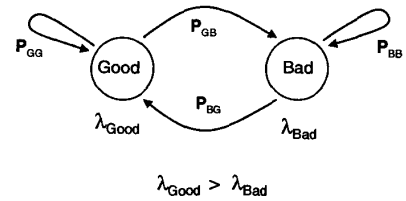

Figure 7: Markov modulated Poisson process.

5. EXPERIMENTS

So far, we have used our testbed implementation to study the performance of existing media streaming applications under various loss and delay patterns. In the following, we describe our hardware setup including some performance measurements and results from a real-time streaming experiment.

For our experiments, we used a Pentium 4 PC running Linux 2.2.16, that hosts both, the Divert Socket program and the client software. All components are located within the corporate LAN of Stanford University, which guarantees high throughput and little congestion due to cross traffic. The experiments were performed during night time in order to ensure that the background traffic was as low as possible.

In a first experiment, we measured the overhead introduced by diverting and reinjecting packets. It turned out that the additional mean delay on our platform is below 0.1 ms. We measured the difference of the mean round trip time of ping packets without setting a firewall rule and with intercepting and reinjecting the packets at the pinged server. Client and server have been selected within the same LAN segment. Table 2 shows the round trip times per packet averaged over 500 ping requests.

client	mean round trip time	
	without diverting	with diverting
A	0.188 ms	0.246 ms
B	0.102 ms	0.155 ms

Table 2: Influence of IP-packet diverting on the round trip time per packet for two different workstations A and B.

In a second experiment, we used the same testbed setup to evaluate the rate control behavior of commercially available media

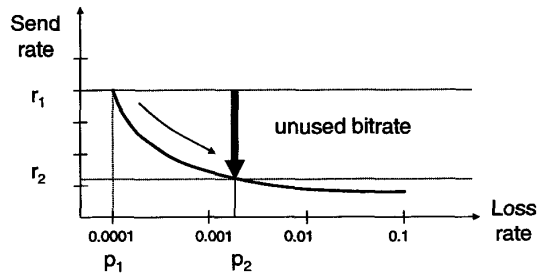

Figure 8: Problems of TCP-friendly rate control for wireless transmission.

streaming applications, e.g., RealServer and RealPlayer [9], over wireless Internet. The channel model employed in this experiment is the two state burst loss model described in Section 4.2. We varied the mean packet loss rate from 0 to 30% and fixed the mean burst loss length to 1000 ms. The server uses SureStream technology [10] where independently encoded streams are available at 25.5, 34.0, 180.0, and 240.0 Kbps.

As expected, we observed that under these conditions the rate control algorithm misinterprets the lost packets to be caused by network congestion and continuously reduces the send rate. After a short time, this results in low user perceived quality which is due to the selection of the stream with the lowest send rate (25.5 kbps). This is because the rate control algorithms for wireline transmission typically rely on the TCP friendliness equation [11] where packet losses cause the server application to reduce network congestion by decreasing the send rate. This is obviously a misinterpretation of the actual situation where high throughput is available.

Figure 8 illustrates the aforementioned effect for send rate adaptation due to a sudden increase in loss rate. As long as the loss rate is small (p_1), the send rate is high (r_1). As soon as the loss rate increases to p_2, the server reduces its send rate to r_2. This is a good strategy for wireline transmission where the increase in loss rate is likely to be caused by network congestion. For wireless transmission, however, this strategy is inadequate, since the increase in loss is typically not due to network congestion but due to transmission link impairments. In this case, redundancy should be added (duplicate packets, FEC, etc.) which corresponds to an increase in sending rate.

Please note that the previous experiment is only one example of potential investigations that can be performed with our proposed test environment.

6. CONCLUSIONS

We describe a network testbed for real-time experiments with streaming media that is simple in its setup and low in its overhead. The testbed allows us to investigate the performance of commercial streaming applications over wireless networks without the need of complex and costly field trials. The testbed can be conveniently setup in a LAN environment. The particular behavior of wireless packet transmission can be modelled using a variety of channel models. New channel models beyond the ones we have implemented can easily be added since packet interception and reinjection are performed in a user program.

Our implementation is based on *Linux Divert Sockets*. We have observed that these sockets introduce very little overhead. We believe that this test environment will be of great use for streaming media application developers who plan to optimize their algorithms for wireless transmission. Our source code and a detailed description on how to setup the testbed and how to add new functionality can be found at http://www.lkn.ei.tum.de/testbed.

In future work we plan to use this test environment to develop and test optimal packet streaming strategies for wireless networks. We will study the real-time performance of recently proposed algorithms that dynamically adapt to network variations, e.g., adaptive media playout techniques [12]. For those investigations, it is crucial to bring the human user in the loop in order to be able to optimize the perceived quality. This is now easily possible using the testbed proposed in this paper.

7. REFERENCES

[1] H. Yumbia, K. Imai, and M. Yabusaki, "IP-Based IMT Network Platform," IEEE Personal Communications Magazine, pp. 18-23, Oct. 2001.

[2] Divert Sockets for Linux: http://www.anr.mcnc.org/~divert/index.shtml

[3] M. Link, S. Gruhl, M. Bauer, and M. Söllner, "A Real-Time Testbed for Adaptive Multimedia Applications over UMTS," Proceedings 3Gwireless 2001, San Francisco, 2001.

[4] DummyNet: http://info.iet.unipi.it/~luigi/ip_dummynet/

[5] W. Stevens, "UNIX Network Programming, Volume 1, Second Edition: Networking APIs: Sockets and XTI," Prentice Hall, 1998, ISBN 0-13-490012-X.

[6] L.N. Kanal and A.R.K. Sastry, "Models for channels with memory and their applications to error control, " Proc. of the IEEE, vol. 66, no. 7, pp. 724-744, July 1978.

[7] U. Horn, K. Stuhlmüller, M. Link, and B. Girod, "Robust Internet Video Transmission Based on Scalable Coding and Unequal Error Protection," Image Communication: Special Issue on Real-time Video over the Internet, pp. 77-94, September 1999.

[8] V. Parthsarathy, J.W. Modestino, and K.S. Vastola, "Design of a transport coding scheme for high-quality video over ATM networks," IEEE Trans. on Circuits and Systems for Video Technology, pp. 358-376, vol. 7, April 1997.

[9] RealNetworks at http://www.real.com/

[10] G.J. Conklin, G.S. Greenbaum, K.O. Lillevold, A.F. Lippman, and Y.A. Reznik, "Video coding for streaming media delivery on the Internet," IEEE Transactions on Circuits and Systems for Video Technology, pp. 269-281, vol. 11, no. 3, March 2001.

[11] S. Floyd, M. Handley, J. Padhye, J. Widmer, "Equation-Based Congestion Control for Unicast Applications," In Proceedings ACM SIGCOMM, 2000.

[12] E. Steinbach, N. Färber, and B. Girod, "Adaptive Playout for Low-Latency Video Streaming," Proceedings International Conference on Image Processing, ICIP-2001, pp. 962-965, Thessaloniki, Greece, Oct. 2001.