Translating Scalable Video Streams from Wide-Area to Access Networks

Carsten Griwodz, Steffen Fiksdal, Pål Halvorsen

IFI, University of Oslo, Norway
Email: {griff, steffenf, paalh}@ifi.uio.no

Abstract

Transmitting video over UDP has been considered advantageous because it allows for discarding of packets in favor of retransmissions, and sender-controlled timing. Using UDP has been criticized because it allows video streams to consume more than their fair share of bandwidth, which is typically associated with the back-off behavior of TCP. TCP-friendly algorithms are meant as a middle path. However, UDP delivery to end systems may still be prevented by firewalls or for other reasons, and TCP must be used. This in turn suffers from bandwidth fluctuations. Therefore, we investigate an architecture that separates the transfer of a video stream over long distances in a TCP-friendly transmission in the backbone and TCP transmission in access networks.

In this paper, we consider a proxy server to translate the traffic and two straight-forward approaches for the translation of a layered video stream transmission from the TCP-friendly transport protocol to TCP. In the first approach, we look at two strictly decoupled transfers with unlimited buffers at the proxy. In the second approach, we consider a single-threaded proxy implementation that uses blocking TCP sockets and experiences backpressure for forwarding the data. We do not expect that one of these two approaches is by itself suited for the task, but investigating them will provide with insights in their basic functions and help in discovering appropriate modifications. For the investigation, we use an experimental approach where network behavior and content are emulated.

Keywords

Scalable video, proxy server, TCP-friendliness, experiments

1 Introduction

The amount of streaming media services in the Internet has been increasing in the last years, and on-demand access to stored content makes up a major share of this. To reduce the increase in bandwidth demand on their up-link connection to the general Internet, many Internet service providers (ISPs) make uses of proxy caching at the edge of their network. Since these proxy caches cannot store all content, there will still be delivery of streams from original servers, but we propose that the proxy servers should be used to improve also this delivery.

Typically, streaming services in the Internet are envisioned to use UDP for data transfer because multimedia applications have other demands on bandwidth, reliability and jitter than offered by TCP. This is in reality impeded by firewalls installations in access networks and on end systems allowing TCP traffic only. Therefore, a solution that uses streaming of TCP in access networks is desirable. The transport of streaming media data over several backbone networks, however, would be hindered by the use of TCP because of its probing behavior, which leads to rapid reduction and slow recovery of its packet rate. Applications using UDP do not

have to follow this approach and have therefore been criticized because they do not leave other traffic its fair share of the available bandwidth. Since TCP is considered an appropriate reference for fairness, protocols and mechanisms that react to congestion in a way that is similar to TCP are classified TCP-friendly and considered more acceptable. Widmer et al. (2001) provide a survey of such protocols and mechanisms. One definition of TCP-friendliness is that the packet rate of a TCP-friendly application should be proportional to the inverse square root of the packet loss rate ($1/\sqrt{(p)}$). TCP-friendly transport achieves the same throughput as TCP on average but with less throughput variations.

We show considerations for combining the use of TFRC in the backbone with the use of TCP in access networks. In particular, we identify buffer use on the proxy and rate fluctuations in the network. We relay on scalable video, which makes it possible for real-time streaming services to adapt to variations in the packet rate. A variety of approaches exists, and in this paper, we consider an approach to course-grained scalable video that achieves high quality, Scalable MPEG (SPEG) (Krasic, 2004). Even though Krasic (2004) aims at an approach for streaming scalable video over TCP, it works with TCP-friendly approaches that have fewer variations in the packet rate than TCP and lead to reduced buffer use on server and clients. The rest of this paper is organized as follows: Section 2 looks at some related work. In section 3, we describe our experiments and our results. Based on our results, we outline different mechanism combinations in section 4, and finally, we conclude in section 5.

2 Related Work

Proxies have been used for improved delivery of streaming media in several earlier works. Prefix caching has addressed issues of latency and backbone throughput (Sen et al., 1999). In multicast works such as receiver-driven layered multicast (McCanne et al., 1996), multicast routers which may also be proxy servers are meant to forward packets only to those subtrees of the multicast tree that have subscribed to those packets. In more recent work, proxies have filtered out layers of a scalable format to adapt to end-system capabilities (Zink, 2003). While the idea of using scalable codecs for adaptivity is rather old, scalable codecs for streaming video that are able to maintain a high quality over a wide range of bandwidths have been developed only in the last few years. Horn and Girod (1994) and Krasic (2004) have worked on course-grained scalability, more recently fine-grained scalability and multiple description coding have attracted interest.

In this paper, we want to draw attention to issues that occur when a proxy is used to translate transport protocols in such a way that TCP-friendly transports mechanisms can be used in backbone networks and TCP can be used in access networks to deliver streaming video through firewalls. Krasic (2004) argues that the most natural choice for TCP-friendly traffic is using TCP itself. While we agree in principle, their priority progress streaming approach requires a large amount of buffering to hide TCP throughput variations. In particular, this smoothing buffer is required to hide the rate-halving and recovery time in TCP's normal approach of probing for bandwidth which grows proportionally with the round-trip time. To avoid this large buffering requirement at the proxy, we would prefer an approach that maintains a more stable packet rate at the original sender. The survey of Widmer, Denda, and Mauve (2001) shows that TFRC is a reasonably good representative of the TCP-friendly mechanisms for unicast communication. Therefore, we have chosen this mechanism for the following investigation.

3 Experiments

To investigate the effects of translating a video stream from a TCP-friendly protocol in the backbone to TCP in the access network, we have used emulation. Since the throughput of the TCP-friendly protocol and TCP will be different, a proxy implementation has four means of handling congestion in the access network:

1. It can buffer data that it receives from the server until the access network recovers from congestion.
2. It can create backpressure on the link between proxy and server by refusing to accept packets.
3. It can adapt the bandwidth between proxy and client by dropping packets from the server in a controlled manner.
4. A combination of the above.

In this investigation, we have considered the first two options. We expect that the third option would yield results similar to our investigation for TCP-friendly transport in backbone and access networks in Zink et al. (2003). The fourth option requires understanding the first three and is future work.

3.1 Experimental Setup

In our experimental investigation, we have emulated a real system using the server, proxy and client implementations of the KOMs Streaming System (*komssys*, Zink et al. (2001)), the NistNET network emulator (Carson and Santay, 2003) to emulate delay in backbone and access network links as well as packet loss in the access network, and the *tg* traffic generator to model cross traffic in the backbone [1]. For the investigation, an implementation of RTP over TCP was added to *komssys*. This option was preferred over the in-band delivery of video-data in the RTSP control channel in order not to break the separation of control and data flow in *komssys*. To support TCP-friendly behavior between server and proxy, we also use the TFRC implementation in *komssys*, which is an RTP variation in which we use application-specific RTCP messages that are used to acknowledge each received packet. Note that TFRC is modelled after TCP Reno, but as TCP implementation we used the default of Linux 2.4, which is TCP FACK.

The content used consists of dummy videos that are modeled after the SPEG encoding that is used in priority progress streaming. The dummy videos have 4 layers of constant equal bandwidth of 0.25 Mbps. Only the server in our scenario discards packets in a controlled manner according to their priority. In particular, and in contrast to Zink (2003), the proxy in this scenario uses priority-aware filtering. By considering layered video, we achieve a situation where controlled packet loss can allow the reduction of bandwidth that is required for transmission of the video and maintain the videos' original frame rate. We intend to maintain this frame rate. We ignore badput at the proxy, i.e. we do not discard high layer packets at the proxy when their corresponding lower layer packets have gotten lost.

In the following two sections, we show the observations that we made in the case of using buffering and backpressure, respectively. We use the term packet rate when we refer to the amount of data that is sent from the server to the proxy. This is appropriate because the transmission is performed in independent UDP packets, which is an important detail for the

[1] The packages used are available from http://komssys.sourceforge.net, http://snad.ncsl.nist.gov/nistnet and http://www.kom.tu-darmstadt.de/rsvp, respectively.

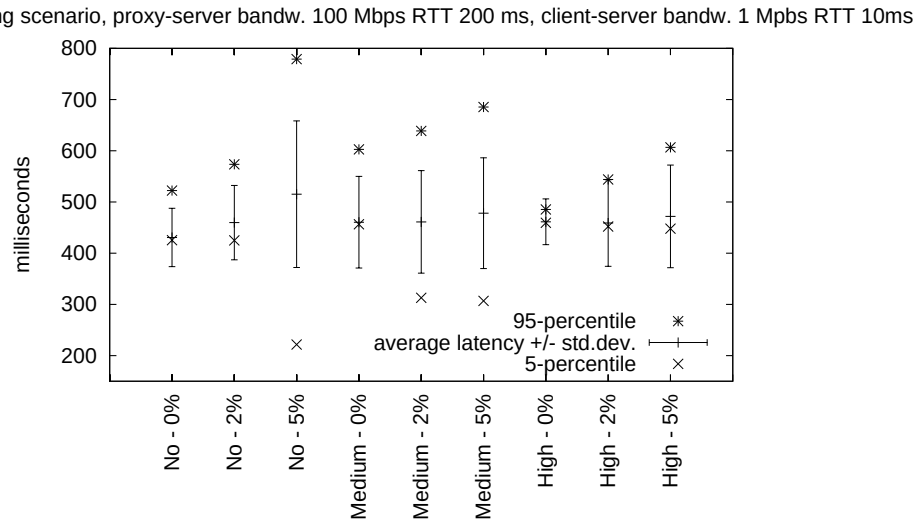

Figure 1: **Latency comparison for 0%, 2%, 5% access net packet loss, and no, medium, high backbone cross-traffic**

interpretation of section 3.3. We use a constant MTU size of 1500 bytes, and therefore, the packet rate translates directly to a bandwidth.

In each of the tests shown here we investigate 9 different scenarios. These are combinations of packet loss in the access network and cross-traffic in the backbone. We consider 0%, 2% and 5% packet loss in the access network, and no, medium or high cross traffic in the backbone. Both medium and high cross-traffic are modeled by one Pareto source with a Hurst parameter of 0.5 for the average self-similarity and two greedy TCP sources.

3.2 Buffering

In this investigation, we look at scenarios where the proxy buffers all data received from the server until it can be forwarded to the client. No data is discarded at the proxy. This implies that the packet rate between server and proxy is decoupled from the transmission of data from the proxy to the server. So, in case of congestion that is experienced between proxy and client, buffer growth at the proxy cannot be avoided. However, the bandwidth between proxy and client must be sufficient to accommodate the full data rate of the video that is sent from the server to achieve a stable scenario. If this is not the case, the buffer consumption at the proxy grows infinitely. This condition must still be fulfilled on average when packet loss between proxy and client leads to a bandwidth reduction. In these scenarios, two resources are critical, the buffer size in the proxy and the jitter experienced at the client:

- The buffer size can become large at the proxy even under the condition that the average data rate that is sent from the server is lower than the average TCP throughput between proxy and client. Since we perform neither packet dropping or reordering at the proxy, the latency of packet arrivals at the client grows with the buffer size at the proxy, and it must be accommodated by pre-buffering the beginning of earlier data at the client. The user will experience a large startup latency, unless the client implementation ignores these buffering delays and chooses to break the continuous playback of the video.
- The jitter experienced at the client is near exclusively due to the delays introduced by

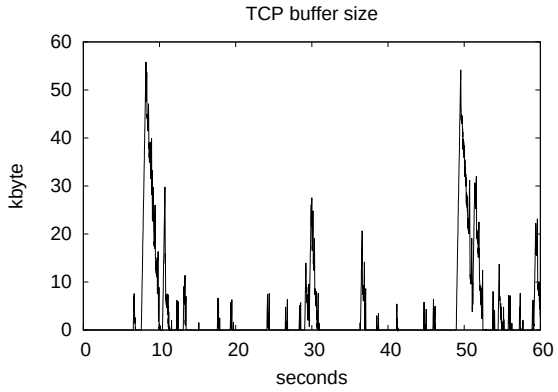

Figure 2: **Buffer development on a lossy TCP link**

the TCP congestion control between proxy and client. In addition to the buffering at the client that is necessary to overcome the delay that is introduced by the variation in average throughput on the TCP link, buffering is also necessary to hide jitter from the end user.

The graph in figure 1 compares the latency under various network conditions. It shows that the cross-traffic between server and proxy, which leads to loss and packet rate reduction in TFRC, affects the end-to-end latency even though the latency between server and proxy remains nearly constant. Thus, the change is only due to the development of buffer space at the proxy. It is noteworthy that the affect on the minimal and average latency is limited, while the standard deviation and 95-percentile is considerably smaller when the TFRC algorithm reduces the bandwidth between server and proxy. This implies that the server reduces the bandwidth of the video at the source by sending only the lower layers of the layered video.

The reason for the limited effect on the average delay is that the buffer at the proxy must be empty most of the time. Since the average bandwidth between proxy and client is limited to a bandwidth that can be sustained even in the case of high packet loss between proxy and client, we have guaranteed that the buffer on the proxy is frequently empty even if no packets are lost between server and proxy (see figure 2). Thus, the buffer is small in the average case and grows only under temporary bandwidth reduction due to the TCP congestion control algorithm. Then, the rate of packet arrival from the server determines how quickly the buffer grows.

3.3 Backpressure

In this section, we consider a single-threaded implementation that uses blocking TCP write for transmission from the proxy to the client. This variation is advantageous because of its straightforward implementation. We can also expect that congestion in the access network affects the connection between server and proxy. For this scenario, we increase the TFRC bandwidth limit in such a way that the access network cannot support the maximum data rate that can be sent from the server. The sending rate of the server must be limited by backpressure from the proxy. This means for TFRC that the proxy must report loss.

In our experiments, we observe huge fluctuations in the TFRC transmission rate that far exceed the bandwidth variations of TCP even in the best case, without packet loss in the access network and no cross traffic in the backbone. Obviously, this means that the server changes the number of layers that it sends for video frequently. Even though all packets are delivered to the client, it experiences frequent quality variations. We found that the reason for this is the

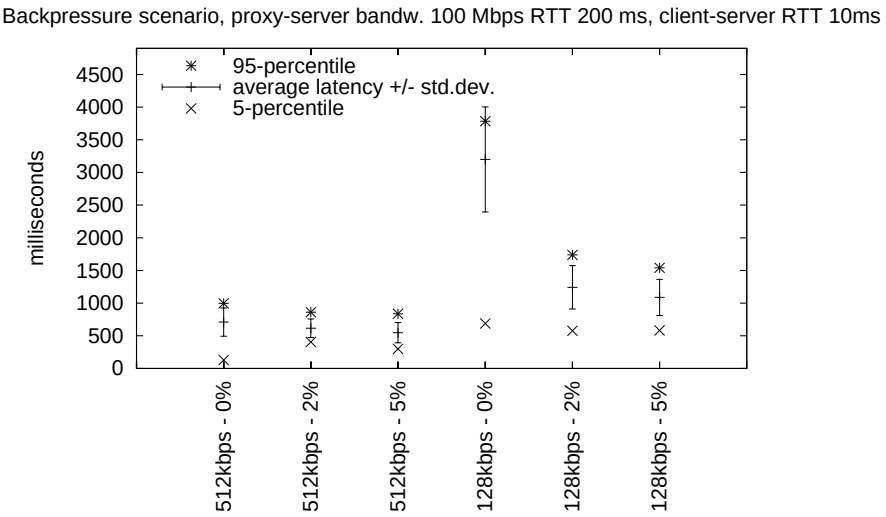

Figure 3: **Latency variation over access networks with 512 kbps and 128 kbps and 0%, 2%, 5% packet loss**

TCP implementation, in particular the way in which the kernel notifies the application about free space in the TCP send buffer. The reason is found in the Linux implementation of Clark's algorithm: a sender that blocks in a TCP write operation is woken up only when the send buffer becomes one third empty. This behavior affects the TFRC algorithm indirectly. While the proxy blocks in a TCP write operation, UDP packets continue to arrive from the server. However, UDP packets are not queued in the kernel and since the receiving thread blocks in the TCP write, the UDP packets are dropped. Consequently, they are reported as lost packets to the TFRC algorithm which reduces that data rate. Subsequently the lower data rate can be handled by the TCP send buffer without running full and blocking the proxy thread. No more misses are reported to the server and TFRC algorithm increases the bandwidth gradually.

An interesting detail of this problem is that it worsens when the average bandwidth is reduced. Thus, when the bandwidth of the access network is lower, the server will on average send no more packets than can be supported by the access network but the intensity of quality changes that become visible at the client is larger. Figure 3 shows this effect. The difference in the figure appear because it takes longer to transfer one third of the constant-sized TCP send buffer on a 128 kbps line than on a 512 kbps line. In the time it takes on a 128 kbps line, more packets from the server are lost and TFRC reduces the bandwidth much further than in the 512 kbps case. Subsequently, the same number of packets are successfully copied into the TCP send buffer (even though of lower video layers), the same low packet loss numbers are reported to TFRC as in the 512 kbps case, and the sending rate of the server rises in a similar manner. Another detail is that the situation improves somewhat when some loss occurs in the access network.

4 Evaluation

We conclude from the **buffering** experiments in section 3.2 that it would be very beneficial for the latency at the client to buffer only the lower layers of the video in case of temporary congestion on the proxy-client link, and to discard the higher layers in a controlled manner. We

can imagine two approaches for reducing the effects of the TCP congestion control mechanism:

- We can send feedback to server to reduce bandwidth at the server side. However, this may take too long because the time for sending an entire TCP send buffer of 64 kbytes which contains between 62.5 ms and 250ms of video data, depending on the number of video layers transmitted. We assume big round-trip times between server and proxy (200 ms in our experiments), and we have seen in figure 2 that even in the worst case, the proxy buffer does not exceed the size of one additional maximum TCP send buffer. Thus, this would be counter-productive and only prevent higher layers from reaching the client after the temporary congestion has disappeared. However, reporting to the server the average rate that is supported between proxy and client would be a good alternative to the fixed bandwidth limit of our experiments.
- We can filter layers at the proxy. Filtering at the proxy is an alternative for this. By examining the timestamps and priorities that are associated with the packets that arrive from the server, packets of the higher layers can be discarded. In this way, lower layers packets are buffered that have later timestamps, and thus, the effective latency at the time of arrival at the client is reduced.

As an alternative to this approach which decouples the backbone and access network traffic completely, we can look at a strict coupling without blocking as well. In *Priority-aware dropping*, we drop higher layers from the queue in a controlled manner to prevent jitter when the application level buffer grows. Approaches that could be adapted to this have been presented by Zink (2003) and by Krasic (2004).

Furthermore, the proxy could filter packets based on a jitter threshold. It requires that the client informs the proxy about the amount of artificial latency that it introduces to reduce jitter, and the amount of buffer space that it uses for this. The proxy can use this information to make decisions about packet forwarding. It can extract the expected playout time of packets arriving from the server, and discard packets from its send queue if they would have to be discarded at the client. As shown by Zink (2003), the best solution is not to discard the oldest packet, but the number of layers that are forwarded should be maintained constant for as long as possible. This prevents foremost the jitter but also smooths out the layers so that the quality variations can be kept at a minimum.

From the **backpressure** experiments in section 3.3 we can conclude that an implicit limitation of the bandwidth between server and proxy is not at all a viable solution without a better means of reporting availability of TCP send buffer space to the application. One such means is the TCP Minbuf patch cited in Krasic (2004). Their approach reduces the TCP send buffer to the congestion window size and applies Clark's algorithm only to that buffer. Thereby, application that perform TCP write are awaken more frequently when the available bandwidth is low. In our backpressure approach, this would lead to more frequent, but less severe TFRC loss reports to the server. The server would then determine a less varying packet rate and send more lower layer and less higher layer packets.

The Priority-aware dropping and Jitter Threshold approaches have the disadvantage that they decouple backbone and access network like the buffering approach, even though the effect quality is reduced. But to some extent, packets that are dropped at the proxy could have been dropped from the transmission of the server in the first place. One possibilty would be the use of Early Congestion Notification (ECN) extensions to TFRC which makes it possible to reduce the sending rate without actual losses.

5 Conclusion and Future Work

In this paper, we have investigated two straight-forward approaches for the translation a layered video stream transmission from a TCP-friendly transport protocol in the backbone to TCP in the access network. As expected, we found that neither of these unmodified approaches are particularly well suited for the task, but we have gained valuable insight into the reasons for the failures of these two approaches.

Specifically, we found for the case of strict decoupling of backbone and access network that unpredictable amounts of buffering are necessary at the proxy. These would force clients to prefetch large amounts of data before starting playback or to accept interruptions in playout.

For the case of strict coupling of backbone and access network, we traced the problems to the typical implementation of TCP. We looked at forwarding in a single thread using blocking TCP write. Typical TCP implementations allow applications to copy data into the TCP send buffer in burst, which leads to long phases of blocking and long phases of non-blocking in the proxy thread, which results in inconclusive feedback to the server and frequently quality variations at the client. We have therefore discussed a set of options that lead to a middle way and can improve performance. We plan to investigate alternative options like those described in section 4 in the future.

References

M. Carson and D. Santay. NIST Net: a Linux-based network emulation tool. *ACM Computer Communication Review*, 33(3):111–126, July 2003.

U. Horn and B. Girod. Pyramid coding using lattice vector quantization for scalable video applications. In *Proceedings of the International Picture Coding Symposium (PCS)*, pages 183–185, Sacramento, CA, USA, Sept. 1994.

C. Krasic. *A Framework for Quality Adaptive Media Streaming*. PhD thesis, OGI School of Science & Engineering at OHSU, Feb. 2004.

S. McCanne, V. Jacobson, and M. Vetterli. Receiver-driven layered multicast. *ACM Computer Communication Review*, 26(4):117–130, Aug. 1996.

S. Sen, J. Rexford, and D. Towsley. Proxy Prefix Caxching for Multimedia Streams. In *Proceedings of the Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pages 1310–1319, New York, NY, USA, Mar. 1999. IEEE Press.

J. Widmer, R. Denda, and M. Mauve. A survey on TCP-friendly congestion control. *Special Issue of the IEEE Network Magazine "Control of Best Effort Traffic"*, 15:28–37, Feb. 2001.

M. Zink. *Scalable Internet Video-on-Demand Systems*. PhD thesis, Darmstadt University of Technology, Darmstadt, Germany, Nov. 2003.

M. Zink, C. Griwodz, J. Schmitt, and R. Steinmetz. Scalable TCP-friendly video distribution for heterogeneous clients. In *Proceedings of SPIE/ACM Conference on Multimedia Computing and Networking (MMCN)*, pages 102–113, San Jose, CA, USA, Jan. 2003. SPIE.

M. Zink, C. Griwodz, and R. Steinmetz. KOM player - a platform for experimental vod research. In *IEEE Symposium on Computers and Communications (ISCC)*, pages 370–375, July 2001.