Transparent Protocol Translation and Load Balancing on a Network Processor in a Media Streaming Scenario

Håvard Espeland, Carl Henrik Lunde, Håkon Kvale Stensland,
Carsten Griwodz, Pål Halvorsen
Simula Research Laboratory, Norway
Department of Informatics, University of Oslo, Norway
{haavares, chlunde, haakonks, griff, paalh}@ifi.uio.no

1. INTRODUCTION

Today, major newspapers and TV stations make live and on-demand audio/video content available, video-on-demand services are becoming common and even personal media are frequently uploaded to streaming sites. The discussion about the best transport protocol for streaming has been going on for years. Currently, HTTP-streaming is usual although the transport of streaming media data over TCP is hindered by TCP's probing behavior, which results in the rapid reduction and slow recovery of the packet rates. On the other hand, UDP has been criticized for being unfair against TCP, and it is therefore often blocked by access network providers.

To exploit benefits of both TCP and UDP, we have implemented a proxy that performs transparent protocol translation in such a way that the video stream is delivered to clients in a TCP-compatible and TCP-friendly way, but with UDP-like smoothness. The translation is related to multicast-to-unicast translation [3] and to voice-over-IP proxies that translate between UDP and TCP. Furthermore, it is also similar to the use of proxy caching that ISPs employ to reduce bandwidth demands. The unique advantage of our approach is that we avoid full-featured TCP handling on the proxy server but still achieve live protocol translation at line-speed in a TCP-compliant, TCP-friendly manner. Although we discard packets just like a sender of non-adaptive video over TCP, we achieve higher user-perceived quality because our proxy can avoid receive queue underflows in the proxy, while also achieving the same average bandwidth as a TCP connection between proxy and client.

In this demo, we present our prototype implemented on an Intel IXP2400 network processor. The prototype proxy does not buffer outgoing packets, yielding data loss in case of a congested TCP side. Comparing HTTP-streaming from a web-server and RTP/UDP-streaming from a video server shows that, in case of some loss, our solution using UDP from the server and a proxy that translates to TCP delivers a smoother stream at playout rate while the end-to-end TCP stream oscillates heavily.

NOSSDAV '08 Braunschweig, Germany

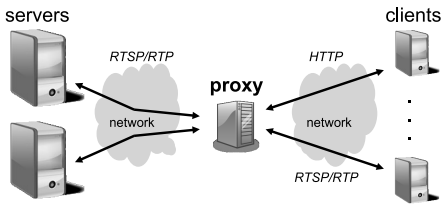


Figure 1: System overview

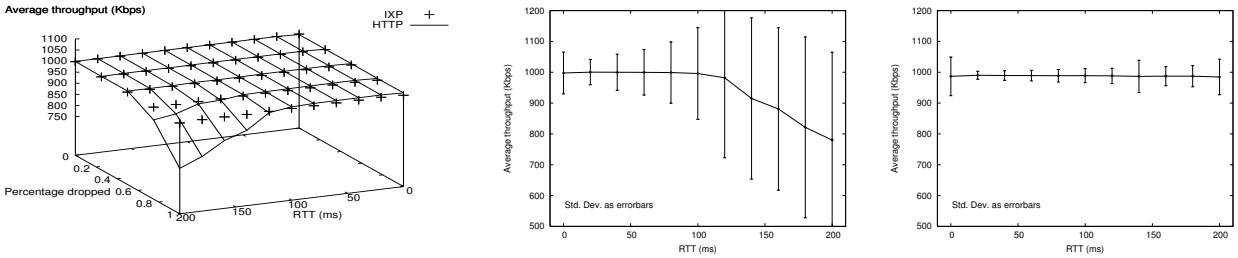
2. PROXY

An overview of our scenario with a proxy that performs transport protocol translation and additional load balancing is shown in figure 1. The idea is that the client can receive a video stream using HTTP or RTP/UDP. The server, however, supports only RTSP for stream control and RTP/UDP for streaming. For every client that accesses a video with HTTP, the proxy performs transparent translation of the stream. The translation adheres to TCP's flow control mechanisms and applies TFRC [1] for rate control. It also offloads TCP connection management from the server. Additionally, the proxy transparently directs new sessions to an appropriate server by rewriting packet addresses. In this demonstration, we use a round-robin strategy to select the server. More complex algoritms can be added easily using the implemented monitor functions.

3. NETWORK PROCESSOR

The protocol translation and load balancing ideas are far from new. However, current implementations are implemented on top of full operating systems in general-purpose computers. They introduce overheads like data transfers and copying, interrupts and full protocol stack processing.

To avoid these overheads and offload the proxy machine, our prototype is implemented on a programmable network processor using the IXP2400 chipset [2] that is designed to handle a wide range of access, edge and core applications. With respect to different processors, an XScale running Linux is typically used for the control plane (slow path) while eight μEngines perform general packet processing in the data plane (fast path). Thus, using such a network processor, we have low-level access to each packet. Interrupts and data copying in the data path are avoided and only a light-weight packet processing is performed. We can operate at Gb line-speed without host overhead.



(a) HTTP and translation results (b) HTTP streaming (c) Protocol translation

Figure 2: Achieved bandwidth varying drop rate and link latency with 1% loss

4. IMPLEMENTATION

According to the intended use of the different processors, incoming packets are classified by the μEngine based on their header. RTSP and HTTP packets are enqueued for processing on the XScale core (control path) where the session is initiated and maintained. The handling of RTP packets, including protocol translation and address rewriting, is performed on the μEngines (fast path). Here, the protocol translation is performed by replacing headers (see figure 3). The translation reuses the space holding the RTP and UDP headers for the TCP header. It updates only the checksum and protocol number in IP before forwarding the packet to the clients.

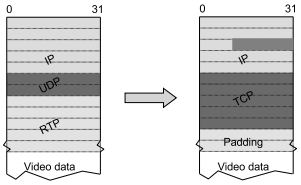


Figure 3: RTP/UDP to TCP translation

With respect to the TCP implementation, the prototype proxy has a minimal TCP implementation to manage retransmissions. The content of a retransmitted packet is empty or refreshed with new data from the server. To deal with congestion control and fairness, we use TFRC [1], which is a specification for best effort flows that compete for bandwidth, designed to be reasonably fair to TCP flows. TFRC achieves a slightly lower average data rate than TCP Cubic. It uses a formula [1], which defines the outgoing bandwidth threshold and is calculated from the current RTT and packet loss rate. The rate control is performed on a μEngine using fixed point arithmetics, and packets in excess of this rate are dropped. The proxy does not buffer unacknowledged packets, thus lost video data is never retransmitted. In case of live and interactive streaming scenarios, delays due to retransmission may introduce dropped frames and delayed play out. This can cause video artifacts, depending on the codec used. However, this problem can easily be reduced by adding a limited buffer per stream sufficient for one retransmission on the proxy and is left as further work.

5. DEMONSTRATION

In our demo, we demonstrate a proxy that performs transparent transport protocol translation to utilize the strengths of both the TCP and UDP protocols in a streaming scenario. The demo system can also balance the load of the servers.

We demonstrate the performance of our proxy that translates RTP over UDP to HTTP over TCP compared to a setup that uses end-to-end HTTP-streaming. The proxy is present in both scenarios, but it forwards TCP packets unchanged. We have performed experiments under various link latencies (0 - 200 ms) and packet drop rates (0 - 1 %). In one configuration, packets are dropped between the server and the proxy, simulating a proxy installation close to the client where backbone traffic leads to random packet loss. In the other configuration, packets are dropped between proxy and client, simulating a proxy close to the server that could aim at off-loading a server cluster from TCP processing load.

Figure 2 shows examples of packet dropping between the server and the proxy. Dropping packets between the proxy and client has similar results. Figure 2(a) shows the achieved average throughput for the different combinations of loss and link delays. Additionally, figures 2(b) and 2(c) show the respective results for the HTTP and protocol translation scenarios under a constant loss rate of 1%. Keeping the link delay constant gives similar results. From the figures, we see that protocol translation achieves an average throughput that is hardly affected by link delays. In contrast, if using TCP end-to-end, the performance drops with an increasing loss rate and link delay, and the bandwidth oscillates heavily. Additionally, our implementation forwards packets in the range of microseconds whereas typical existing systems require several milliseconds. With respect to the load balancing functionality, our implementation works fine as a proof of concept. We are also able to take servers offline, and the framework enables implementing more complex policies.

6. CONCLUSION

The experimental results show that our protocol translation system delivers a smoother stream than HTTP-streaming, whose the bandwidth oscillates heavily. With respect to video quality assessment, this is subject to ongoing work.

7. REFERENCES

[1] Handley, M., Floyd, S., Padhye, J., and Widmer, J. TCP Friendly Rate Control (TFRC): Protocol Specification. RFC 3448 (Proposed Standard), Jan. 2003.

[2] Intel Corporation. Intel IXP2400 network processor datasheet, Feb. 2004.

[3] Parnes, P., Synnes, K., and Schefström, D. Lightweight application level multicast tunneling using mtunnel. *Computer Communication 21*, 515 (1998), 1295–1301.