RAPID: A Reliable Protocol for Improving Delay

Sanjeev Mehrotra, Jin Li, Cheng Huang
{sanjeevm,jinl,chengh}@microsoft.com

ABSTRACT

Recently, there has been a dramatic increase in interactive cloud based software applications (e.g. working on remote machines, online games, interactive websites such as financial, web search) and other soft real-time applications (traffic within data center). Compared to classical real-time media applications (VoIP/conferencing) and non real-time file delivery, these interactive software applications have unique characteristics as they are delay sensitive yet demand in order and reliable data delivery. Therefore existing protocols for delivery of lossless data (such as TCP) and other delivery protocols using UDP do not work well. In this demo, we show substantially improved performance for such traffic by using a transport protocol built on top of UDP which uses intelligent adaptive forward error correction (FEC) and improved congestion control (rate control). The transport protocol is made lossless by using a hybrid FEC/ARQ strategy. The congestion control technique improves the delay performance by preventing congestion induced loss and minimizing queuing delay while still fully utilizing network capacity and maintaining fairness across flows. In this demo, we present RAPID (a **R**eli**A**ble transport **P**rotocol for **I**mproving end-to-end **D**elay) and show show its effectiveness in improving the performance of interactive client-server applications.

Categories and Subject Descriptors

C.2.2 [**Network Protocols**]: Real-time transport protocols

Keywords

Real-time communications, Interactive applications, Cloud Computing, Forward Error Correction (FEC)

1. INTRODUCTION

Soft-real time applications (such as interactive applications) are those where there is a soft requirement on the end-to-end delay being small and where the application performance degrades as this delay becomes larger. Unlike hard real-time applications, there is no strict deadline for the data to remain useful. Application level end-to-end delay refers to the time between when the data is generated to the time it is received by the application. It can be measured on some granularity such as packets or messages (collection of packets).

Table 1: Media streaming vs Interactive app vs File delivery

Media streaming	Interactive App	File delivery
Strict deadline	Delay sensitive	No deadline
Best effort	Reliable delivery	Reliable delivery
No ordering	In-order	In-order
Low delay	Delay-aware	Delay agnostic

Examples of interactive applications are working remotely from one machine to another, online multi-player (or server hosted) games, interactive websites (financial, retail) and web search. Video-on-demand (VOD) and other classical media streaming is usually not considered real-time since the end-to-end delay can actually be very large (the video is generated much before it needs to be viewed for VOD and even for "live" TV is typically generated a few seconds in the past). We summarize the difference between media streaming, interactive applications, and file delivery in Table 1.

Interactive applications are multi-party (where a party can be a user or a machine) and are those where the data which each party generates is in response to actions or data generated by other parties. This data can be audio or video (from microphone or camera) or can be other data such as text, graphics primitives, bitmaps of screen captures, or input data from keyboard or mouse movements.

As an example, consider an application where a thin client is used for display and input (keyboard/mouse) purposes and the server is located in a distant data center. The server processes the incoming commands and the application responds by providing a screen update (via either text, HTML, graphics primitives, or screen bitmaps) to the client. The responsiveness of the application is directly related to the timely delivery of this response. Interactive websites such as banking or retail have similar characteristics.

As another example, consider the response to a web search request. The response is usually a small message (for example around 10KB) which for a 10Mbps link (which is very common these days) is only 8ms of network time. Thus the end-to-end delay for this message delivery (even including the server processing) can be on the order of 100-200ms. Such small messages are fairly common in today's web traffic as the number of small objects in web pages continues

ACM Multimedia 2010 Firenze, Italy

to increase [4]. Also, they are fairly common in data center environments where small messages are routinely passed around between machines running a distributed application. For such applications, latency can significantly reduce the throughput of the application.

Since most interactive applications operate as a state machine, the data has to be delivered losslessly and in-order so that the client and server state are in sync. Therefore most existing applications simply use TCP (New Reno) [1]. However, for interactive applications, any network queuing delay or packet loss can hurt performance by increasing end-to-end latency. With the use TCP like protocols, both of these will be present when congestion is present. In TCP, this is primarily caused by 1) the use of only retransmissions (ARQ) for reliability which results in large delays whenever there is loss (congestion or other) and 2) the use of a window based congestion control and only reducing rate in the presence of packet loss which results in large queuing delays and congestion-induced packet loss.

2. TRANSPORT PROTOCOL DEMO

When the requirement on the end-to-end delay is much larger than the network round-trip time (RTT), protocols such as TCP which fully utilize the network bandwidth while suffering minimal waste (since they only retransmit lost packets) work well even though they suffer from significant queuing delays and some congestion induced packet loss. However, when the end-to-end delay requirement is on the order of RTT, then network queuing delay and packet loss become a main cause of performance issues.

In previous work, we have proposed the use of a protocol running on top of UDP as shown in Fig. 1 which uses a hybrid FEC/ARQ protocol [2, 3] along with an improved congestion control strategy (to determine the network transmission rate). The proposed network protocol improves end-to-end delay of packets or messages by using the following:

- Using a delay based congestion control (which backs off prior to loss thus resulting in network buffers being not completely full) to minimize network queuing delay and congestion induced packet loss.
- Correcting for any remaining packet losses by proactively inserting FEC packets.

Although FEC can be used to reduce delay due to retransmission of lost packets for packets or messages which have already been sent, using it arbitrarily can actually hurt overall delay performance since for a fixed rate channel, the insertion of an FEC packet delays sending of future packets and messages waiting to be sent. Therefore we use a cost function to compute a transmission policy to optimize the total delay seen by all packets (or messages). The transmission strategy portion of the protocol in Fig. 1 attempts to find an optimal transmission policy and the congestion control portion attempts to find an appropriate transmission rate of packets into the network (by controlling the transmission rate and window size for unacknowledged data).

This demo will present the improvements that can be achieved when using such a protocol for use in an interactive client-server communication such as working on a remote machine or interactive web browsing. There are a host of applications for working remotely which allow for connecting from a client (a machine, lightweight device, or browser) to a server (a physical machine, virtual machine, or server in a data center). The client sends keyboard and mouse commands to the server (perhaps in a distant location) and the server sends display updates (in the form of text, dynamically generated HTML, graphics, or bitmaps) to the client.

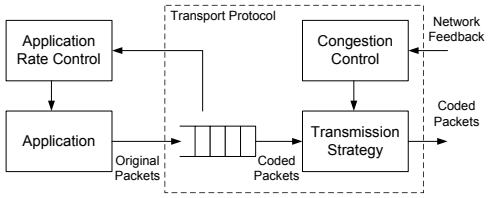

Figure 1: Block Diagram.

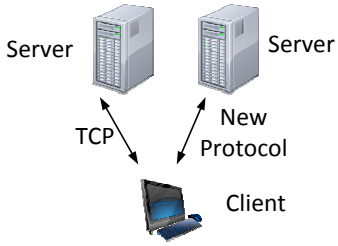

Figure 2: Demo Setup.

This demonstration will show the performance improvements when using this protocol compared to using a standard protocol such as TCP for interactive applications over various network conditions. The demo setup is as shown in Fig. 2 and will compare the performance using two simultaneous interactive client-server sessions which the user can control. The network conditions will be varied using a network emulation tool to simulate realistic network environments. The performance difference between using the two protocols will be easily observable.

3. CONCLUSION

In this demo, we have shown the performance improvement that can be achieved when using intelligent transport protocols designed for applications with low end-to-end latency (interactive applications, short message delivery). Although the demo shown has been for interactive applications with text and graphics data, such a protocol can be widely used wherever there is a requirement on low end-to-end delay. With rich, immersive multimedia communications (audio/video/haptics) ready to take off, such a protocol can find a wide range of uses.

4. REFERENCES

[1] S. Floyd and T. Henderson. *The NewReno Modification to TCP's Fast Recovery Algorithm*, Apr. 1999. RFC 2582.

[2] Y. Huang, S. Mehrotra, and J. Li. A hybrid FEC-ARQ protocol for low-delay lossless sequential data streaming. In *Proc. Int'l Conf. Multimedia and Expo*, pages 718–725. IEEE, June 2009.

[3] S. Mehrotra, J. Li, and Y. Huang. Minimizing delay in lossless sequential data streaming. In *Proc. Int'l Conf. Multimedia and Expo*. IEEE, July 2010.

[4] SPDY: An experimental protocol for a faster web. `http://dev.chromium.org/spdy/spdy-whitepaper`.