Latency reducing TCP modifications for thin-stream interactive applications

Andreas Petlund, Kristian Evensen, Carsten Griwodz, Pål Halvorsen
Simula Research Laboratory
Norway
{apetlund|kristrev|griff|paalh}@simula.no

1 Abstract

A wide range of Internet-based services that use reliable transport protocols display what we call thin-stream properties. This means that the application sends data with such a low rate that the retransmission mechanisms of the transport protocol are not fully effective. In time-dependent scenarios, packet loss can be devastating for the service quality. In order to reduce application-layer latency when packets are lost, we have implemented modifications to the TCP retransmission mechanisms in the Linux kernel. We have also implemented a bundling mechanism that introduces redundancy in order to preempt the experience of packet loss. The enhancements are applied only if the stream is detected as thin. This is accomplished by defining thresholds for packet size and packets in flight. We have implemented these changes in the Linux kernel (2.6.23.8), and have tested the modifications on a wide range of different thin-stream applications. Our results show that applications which use TCP for interactive time-dependent traffic will experience a reduction in both maximum and average latency, giving the users quicker feedback to their interactions.

Availability of such mechanisms will give Linux an edge when it comes to providing customizability for interactive network services. The quickly growing market for Linux gaming may benefit from lowered latency. As an example, most of the large MMORPG's today use TCP (like World of Warcraft [9] and Age of Conan [10]) and several multimedia applications (like Skype [11]) use TCP fallback if UDP is blocked.

2 Introduction

Reliable transport protocols that are in widespread use today, like TCP, are primarily designed to support connections with high throughput efficiently. The aim is to move lots of data from one place to another as fast as possible (like HTTP, FTP etc.). However, a wide range of networked applications do not follow this pattern. Important examples include interactive applications where small amounts of data are transmitted sporadically, e.g., multiplayer online games, audio conferences, and remote terminals. Due to the highly interactive nature of these applications, they depend on timely delivery of data. For example, data delivery latency in audio conferences should stay below 150-200 ms to achieve user satisfaction [1], and online games require latencies in the range of 100-1000 ms, depending on the type of game, to be able to provide a satisfying experience to the players [7]. Furthermore, when dealing with stock exchange systems, even low delays in delivery may constitute a large potential economic disadvantage.

To support different requirements for timeliness, distributed interactive applications have historically been developed for use either with transport protocols that can provide per-stream quality of service (QoS) guarantees, or with protocols that at least allow the sender to determine the timing of the data transmission. The QoS protocols for the first approach have not become widely available. The use of UDP (the protocol that provides for the second approach) has been widely criticized for its lack of

application (platform)	payload size (Bytes) average	min	max	packet interarrival time (ms) average	median	min	max	percentiles 1%	99%	avg. bandwidth requirement (pps)	(bps)
World of Warcraft	26	6	1228	314	133	0	14855	0	3785	3.185	2046
Anarchy Online	98	8	1333	632	449	7	17032	83	4195	1.582	2168
Age of Conan	80	5	1460	86	57	0	1375	24	386	11.628	12375
BZFlag[†]	30	4	1448	24	0	0	540	0	151	41.667	31370
Casa (sensor network)	175	93	572	7287	307	305	29898	305	29898	0.137	269
Windows remote desktop	111	8	1417	318	159	1	12254	2	3892	3.145	4497
Skype (2 users)[†]	236	14	1267	34	40	0	1671	4	80	29.412	69296
SSH text session	48	16	752	323	159	0	76610	32	3616	3.096	2825

[†] Application using TCP fallback due to UDP being blocked by a firewall.

Table 1: Examples of thin stream packet statistics based on analysis of packet traces.

congestion control mechanisms. Consequently, many distributed interactive applications today are built to work with TCP, and many applications that use UDP (in spite of said criticism) fall back to using TCP if, for example, a firewall is blocking UDP traffic. The disadvantage of using TCP is that applications that generate what we call "thin streams", can experience severe delays when TCP recovers from loss. A stream is called "thin" if at least one of the following criteria is fulfilled: **a)** The packet interarrival time (IAT) is too high to be able to trigger fast retransmissions. **b)** The size of most packets is far below the maximum segment size (MSS). The occasional high delays for thin streams are caused by the retransmission mechanisms common for reliable transport protocols [2].

Table 1 contains a statistical breakdown of applications with thin-stream properties. It is a selection from a range of interactive applications that we have analysed in order to better understand the properties of these streams. Common for all of the presented applications is that the average packet size is small. The interarrival times show greater variation, but are all high compared to more throughput-intensive datastreams.

As of today there are several TCP mechanisms implemented in the Linux kernel that affect the latency of thin streams. The most notable is *"Nagle's algorithm"* which delays small packets in order to bundle them together, thus avoiding the transmission of several small packets with relatively short interval. Another mechanism is *"delayed ACKs"*, which delays the sending of ACKs in order to ACK more segments with one message. Both of these mechanisms causes unwanted delay for interactive applications (especially combined), and the vigilant developer will turn them off in the case of an interactive application using TCP.

We have implemented a set of modifications to TCP in the Linux kernel with the goal of reducing the data delivery latency for thin stream applications when retransmissions are necessary. In the next section, we present the basics of our modifications and how they are applied.

3 Thin stream modifications

In this section, we will go into detail on our three TCP modifications. They have all been implemented and tested in the Linux 2.6.23.8-kernel.

3.1 Removal of exponential backoff

When thin stream applications send out packets with high IAT, most retransmissions are caused by timeouts (due to the lack of dupACKs). A retransmission timeout invokes exponential backoff, which doubles the time to wait for the next retransmission. If the number of packets in flight (i.e., unacknowledged packets) is less than the number required to trigger a fast retransmission, we remove the exponential element (i.e. do linear timeout intervals). Since streams that gain from this modification are very thin, the increase in bandwidth consumption is very small. The exponential backoff is located in *tcp_timer.c* and by modifying the if-test surrounding the update of the retransmission timer, we allow exponential backoff to occur only if the stream is considered "thick".

The mechanism can be activated system-wide using the sysctl *"net.ipv4.tcp_force_thin_rm_expb"*, and on a per stream basis using the IOCTL *"TCP_THIN_RM_EXPB"*

3.2 Faster fast retransmit

Instead of waiting several hundred milliseconds for a timeout retransmission and then suffer from exponential backoff, it is more desirable to trigger a fast retransmission. However, when the IAT is high, the time it takes to send enough packets to generate three dupACKs will often exceed the retransmission timeout timer. We have therefore reduced the number of required duplicate acknowledgments to one, provided that the stream is thin.

To reduce the number of required dupACKs, we have added another test to *tcp_time_to_recover()* in *tcp_input.c*. If the stream is thin, the kernel triggers a fast retransmission after only one dupACK.

The mechanism can be activated system-wide using the sysctl *"net.ipv4.tcp_force_thin_dupack"*, and on a per stream basis using the IOCTL *"TCP_THIN_DUPACK"*

3.3 Redundant Data Bundling

Many thin stream applications send packets that are significantly smaller than the MSS, as shown in table 1. Redundant data-bundling (RDB) is a technique that exploits the small packet size, and redundantly bundles unacknowledged data into packets that are to be sent (either new or retransmitted). By bundling, the chance of lost data arriving (and being acknowledged) before a retransmission increases.

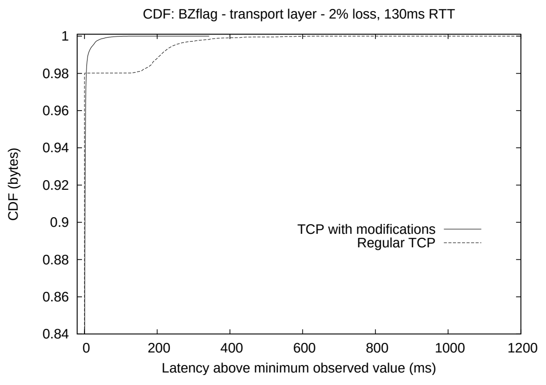
Out of the three modifications, RDB is the most complex, and its implementation touches three files: *tcp.c*, *tcp_output.c* and *tcp _input.c*. In *tcp.c*, we have modified *tcp_sendmsg()*, which is the entry point into the kernel for all packets that are sent using TCP. Provided that certain criteria are met (including that the new packet size will not exceed the MSS), the unacknowledged data in the previous packet in the send queue is copied into the one that is to be sent. Similarly, in *tcp _retransmit_skb()* (which is the function called when a packet is to be retransmitted) in *tcp _output.c*, the kernel performs the same check and copies the unacknowledged data contained in the next packet on the send queue (i.e., the reverse of what happens upon send).

When an acknowledgment is received, the acknowledged data is removed from all packets containing it. Because of the bundling, packets can be "partially" acknowledged. To remove only some data, we had to modify *tcp _clean _rtx _queue()*, which deals with packets that are acknowledged. Now the function also removes acknowledged data (i.e., not just entire packets).

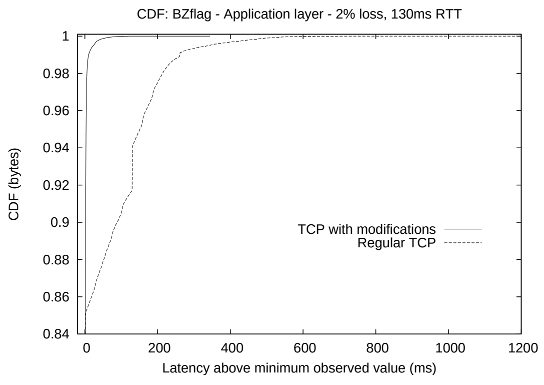
One of the challenges that we faced during the development of RDB, was scatter-gather. The early versions were developed on machines with old hardware that did not support this feature, thus, we did linear memory copy. RDB now supports network cards with and without scatter-gather. If the feature is supported, the kernel copies (and removes) pages instead of the actual data.

The RDB mechanism can be activated system-wide using the sysctl *"net.ipv4.tcp_force_thin_rdb"*, and on a per stream basis using the IOCTL *"TCP_THIN_RDB"*

A major concern with RDB is bandwidth consumption. If the IAT is low and packet size small, a stream will consume large amounts of bandwidth (at least compared to if RDB was switched off). To avoid this in scenarios where bandwidth consumption is important (e.g., mobile), we support limiting the maximum bundle packet size. By providing a bundling limit to the sysctl *"net.ipv4.tcp_rdb_max_bundle_bytes "*, the kernel will never allow the bundled packets to exceed this size. We are currently looking into finding a sweet spot between the tradeoffs of bandwidth consumption, IAT, packet size and link characteristics.



(a) Transport layer latency.



(b) Application layer latency.

Figure 1: CDFs of BzFlag latency.

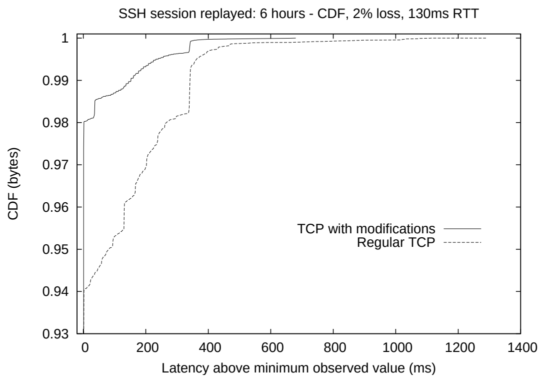


Figure 2: CDF of SSH application layer latency.

4 Experiments and results

Our work on the topic of latency for reliable thin streams to this date is summarized in a series of papers that discuss the problems that thin streams pose to reliable transport protocol performance, possible ways to improve on this condition and tests performed to show the effects of the modifications [2, 3, 4, 6, 5]. Here, we will present two typical examples of scenarios where all of the three modifications are tested.

Figure 1 shows the analysis with regard to delivery latency from dumps of BZFlag [8] game traffic. This game traffic has relatively low interarrival time and very small packets (as shown in table 1). This makes the potential for latency gain by bundling large. The tests were performed by replaying captured data from a BZFlag server over a link where we created loss and delay using a network emulator. Figure 1(a) shows the delivery latency on the transport layer. This reflects when each segment actually arrived at the receiver. We can see that for 2% of the packets, the delay increases abruptly for unmodified TCP (reflecting the test loss rate). The version implementing the modifications, however, is able to deliver many more segments at a lower latency. When we consider TCP's in-order requirement, we get the graph shown in figure 1(b). The graph takes into account that segments that have arrived must wait for earlier, delayed segments before being delivered to the application. This creates a delay in the delivery of as much as 14% of the packets when the loss rate is 2% for unmodified TCP. For the TCP version that implements the thin-stream modifications, the application delay is nearly identical to the transport layer delay.

Figure 2 shows the analysis of replayed data from an SSH-session on the same emulated network (RTT130ms, loss 2%). As shown in table 1, this stream has much higher interarrival time between packets, and will therefore benefit less from the bundling mechanism. The modified retransmission schemes, however, improve the retransmission latencies and the maximum latencies.

Unfortunately, our three TCP modifications may also become active when a network bottleneck is responsible for reducing long-term average throughput to less than 4 packets per RTT, activating the modifications. In this case, our modifications will visibly violate the fairness principle. RDB hides many packet losses from the receiver entirely, packet loss will not necessarily result in a reduction of the send rate, and the stream will consume more bandwidth than one without RDB. There is a fairly narrow window for this to happen, because TCP will bundle small send buffer anyway when transmission is blocked, thereby deactivate RDB. Removing exponential backoff and reducing the number of required dupACKs makes a thin stream (that experiences loss) send additional packets, and this will increase the number of transmission chances and lead, in this situation, to an increased average throughput, violating fairness. Determining exactly how bad the violation is is one of our top priorities, however, due to the thinness of the streams we do not believe it to be that bad.

5 Conclusion

Although there are non-reliable transport protocols available for developers of interactive neworked applications, many still choose to use TCP. In the Linux kernel, TCP has a lot of customization options, but most are aimed at maximizing throughput. Support in the Linux kernel for tuning TCP for interactive thin streams will help make Linux a better platform for hosting interactive services. A server could implement the features, and give its clients the benefit of reduced latency (at least one way). Seen in the light of the increasing market for Linux gaming, general inclusion of thin-stream features will also give users the possibility to benefit from the modifications in both directions of the datastream.

References

[1] International Telecommunication Union *One-way Transmission Time*, ITU-T Recommendation G.114, 2003.

[2] Griwodz, C., and Halvorsen, P. *The fun of using TCP for an MMORPG*, In International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV), May 2006, ACM Press, pp. 1-7.

[3] K. R. Evensen, A. Petlund, C. Griwodz, and P. Halvorsen *Redundant Bundling in TCP to Reduce Perceived Latency for Time-Dependent Thin Streams*, In IEEE Communications Letters 12(4):334 – 336, 2008.

[4] A. Petlund, K. Evensen, C. Griwodz, and P. Halvorsen. *TCP mechanisms for improving the user experience for time-dependent thin-stream applications*, In The 33rd Annual IEEE Conference on Local Computer Networks (LCN), 2008.

[5] A. Petlund, K. Evensen, P. Halvorsen, and C. Griwodz. *Improving application layer latency for reliable thin-stream game traffic*, In Netgames, Worchester, Ma. US, 2008, 2008.

[6] A. Petlund, K. R. Evensen, C. Griwodz. *TCP Enhancements For Interactive Thin-Stream Applications*, In NOSSDAV 2008 (ISBN: 978-1-60588-157-6/05/2008), 2008.

[7] M. Claypool and K. Claypool. *Latency and Player Actions in Online Games*, Communications of the ACM 49, 11, Nov. 2005, 40-45.

[8] http://www.bzflag.org/ - BZFlag website

[9] http://www.worldofwarcraft.com - World of Warcraft website

[10] http://www.ageofconan.com/ - Age of Conan - Hyborian adventures website

[11] http://www.skype.com/ - Skype website