# Improving application layer latency for reliable thin-stream game traffic

Andreas Petlund[1,2], Kristian Evensen[1], Pål Halvorsen[1,2], Carsten Griwodz[1,2]
[1]Simula Research Laboratory, Norway [2]IFI, University of Oslo, Norway
{apetlund, kristrev, paalh, griff}@simula.no

## ABSTRACT

A wide range of networked games send data with very high interarrival-time between packets and with small payloads in each packet. We call these transmission patterns "thin streams". Reliability, when needed for game traffic, is usually achieved through TCP or by using retransmission schemes modelled on TCP. These retransmission schemes can result in very large delays if the stream is thin [9]. Viewed in the light of the time-dependent nature of game traffic, large delays can be severely impeding to the game experience. In order to reduce application-layer latency when packets are lost, we have implemented modifications to TCP in the Linux kernel. The changes are only active when thin-stream properties are detected, thus not affecting TCP behaviour when the stream is not thin. In this paper, we show the latency improvements from these thin-stream modifications. As a case study, we have used the game BZFlag to test the mechanisms, and present statistics from these tests. The experimental results show that our modifications allow TCP to recover earlier from packet loss. This latency reduction was then shown to improve the difference between perceived and actual player positions in the BzFlag game.

## 1. INTRODUCTION

Reliable transport protocols that are in widespread use today, like TCP, are primarily designed for connections with high throughput. 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 upon user interaction, e.g., multiplayer online games and audio conferences. Many such interactive applications use TCP either per default, or as a fallback if UDP is blocked.

Many of these applications generate what we call "thin streams". In this context, 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, and **b)** the packet sizes are usually far below the maximum segment size (MSS). In this case, lost packets are often recovered through timeout-retransmissions resulting in severely delayed data delivery [9].

*Accepted for NetGames'08* Worcester, Massachusetts, USA

Due to the highly interactive nature of many thin-stream applications, they depend on timely delivery of data. As an example, online games require latencies in the range of 100 to 1000 ms, depending on the type of game [6]. In this paper, we discuss thin-stream properties viewed in the light of game traffic by presenting analysis of traces from networked computer games. The traces were analyzed with respect to interarrival time between packets, packet size and consequently bandwidth requirement. The transmission characteristics are then discussed with respect to the shortcomings identified in standard TCP congestion control mechanisms. Moreover, we have implemented a set of modifications to TCP with the goal of reducing the data delivery latency for thin stream applications when retransmissions are necessary. We also show how the reduced position update latencies influence precision when firing shots in BzFlag (a first person shooter game) and how the proposed modifications can help increase the probability of correctly observing the position of an opposing player. In summary, our results show that the proposed modifications allow TCP to recover from packet loss more rapidly, which improves the experience of playing the game.

## 2. THIN-STREAM GAME TRAFFIC

A large selection of networked interactive applications display traffic patterns that we call thin streams. These applications generate data in such a way that packets are small and/or have high interarrival time. A wide range of multi-player networked games produce traffic that match the thin-stream patterns. Several papers have been written that analyse networked game traffic patterns and characteristics (e.g. [14, 4, 5, 6]). With the exception of content-streaming worlds (like Second Life [12]), most games have as a common factor that their generated datastreams display thin-stream properties. In order to understand thin-stream traffic better, we analysed packet traces from a selection of online games (table 1). The analysis shows that thin-stream applications may have difficulties providing proper service when they are sent using established reliable transport protocols (like TCP [9] or SCTP [11]) because normal congestion control and recovery mechanisms are rarely triggered and the stream does never fill the allowed send window. Below, we take a closer look at the implications of these traffic patterns for interactive game traffic.

### 2.1 Game Characteristics

We have analysed game traces for several titles including Anarchy Online (AO), Age of Conan (AoC) and BzFlag with respect to their network traffic characteristics. AO and AoC are massively multiplayer online role-playing games (MMORPGs) whereas BzFlag belongs to the first person shooter (FPS) genre.

| application (platform) | prot-ocol | payload size (bytes) | | | packet interarrival time (ms) | | | | | | avg. bandwidth requirement | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | percentiles | | | |
| | | average | min | max | average | median | min | max | 1% | 99% | (pps) | (bps) |
| Anarchy Online (Server side dump) | TCP | 98 | 8 | 1333 | 632 | 449 | 7 | 17032 | 83 | 4195 | 1.582 | 2168 |
| World of Warcraft (PC) | TCP | 26 | 6 | 1228 | 314 | 133 | 0 | 14855 | 0 | 3785 | 3.185 | 2046 |
| Age of Conan (PC) | TCP | 80 | 5 | 1460 | 86 | 57 | 0 | 1375 | 24 | 386 | 11.628 | 12375 |
| Counter Strike (PC) | UDP | 36 | 25 | 1342 | 124 | 65 | 0 | 66354 | 34 | 575 | 8.064 | 19604 |
| Halo 3 - high intensity (Xbox 360) | UDP | 247 | 32 | 1264 | 36 | 33 | 0 | 1403 | 32 | 182 | 27.778 | 60223 |
| Halo 3 - moderate intensity (Xbox 360) | UDP | 270 | 32 | 280 | 67 | 66 | 32 | 716 | 64 | 69 | 14.925 | 35888 |
| Tony Hawk's Project 8 (Xbox 360) | UDP | 90 | 32 | 576 | 308 | 163 | 0 | 4070 | 53 | 2332 | 3.247 | 5812 |
| Test Drive Unlimited (Xbox 360) | UDP | 80 | 34 | 104 | 40 | 33 | 0 | 298 | 0 | 158 | 25.000 | 22912 |
| BZFlag | TCP | 30 | 4 | 1448 | 24 | 0 | 0 | 540 | 0 | 151 | 41.667 | 31370 |
| Skype (2 users, TCP fallback) | TCP | 236 | 14 | 1267 | 34 | 40 | 0 | 1671 | 4 | 80 | 29.412 | 69296 |

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

In the AO trace, extreme worst-case delays were occasionally experienced by the players. This was determined to be caused by the retransmission mechanisms of TCP due to the thin-stream characteristics of the application data [9]. From table 1, we can see that the AO streams interarrival times are extremely high, with an average of 632 ms. The payloads of the packets are noticeably small with an average of 98 bytes. Similarly, the newly released AoC also displays thin stream characteristics. The average IAT is 86 ms with an average payload of 80 bytes. The FPS game BzFlag shows somewhat different characteristics due to more updates being generated from higher-paced action. Table 1 shows that the interarrival times between packets are significantly lower than for AO with an average of 24 ms. The payload sizes, however, are the smallest of all the analysed applications averaging 30 bytes per packet.

The small packets indicate that only short updates (like position updates and fired shots) are relayed in each packet. They are, however, required to be transmitted with short intervals in order to make the game playable. The (relatively) low packet IAT of BzFlag is still high enough to classify as a thin stream due to the occasional inability to trigger fast retransmissions, i.e., depending on periodic changes in data patterns and network round trip time (RTT). The high intensity of FPS games also means that extra delays will more severely impact the experience of gameplay [6]. Thus, with these stream characteristics and the strict latency requirements in mind, supporting thin-stream interactive applications is a hard task. In the following section, we will therefore discuss the challenges that arise when using TCP for thin streams.

## 2.2 TCP Shortcomings

Griwodz et al [9] describe how, for streams of certain characteristics, TCP retransmission schemes result in higher latency when loss occurs. The reason is that the fast retransmit mechanism in TCP, which enables retransmission of lost segments before a timeout occurs, depends on feedback from the receiver (ACKs). The mechanism requires three duplicate acknowledgements (dupACKs) in order to initiate a fast retransmission [13, 2]. The reason for waiting until the third indication of loss is to avoid spurious retransmissions when reordering happens on the network. For thin stream scenarios, where interarrival times between sent packets are relatively high, the consequence is that many (or all) retransmissions will be caused by timeouts. This is because there are seldom enough packets in flight (packets on the wire) to generate the necessary feedback to trigger a fast retransmit. In addition, the retransmission timeout (RTO) is exponentially increased when multiple losses are detected (exponential backoff), which results in a still larger latency penalty. This mechanism is designed to ensure that an acceptable send rate is found, and to prevent a stream from exceeding it's fair share of the bandwidth resources. Thin streams, however, do not use their fair share, most of the time they stay within the worst-case throughput of 2 packets per RTT. The large IATs make it impossible for the thin stream to back off when packets are lost multiple times, resulting in a situation where the RTO value is very high, without any actual benefit with regard to resource distribution on the network. The result for the thin stream is that the retransmission is delayed by seconds (or even minutes [9]) if segments are lost several times.

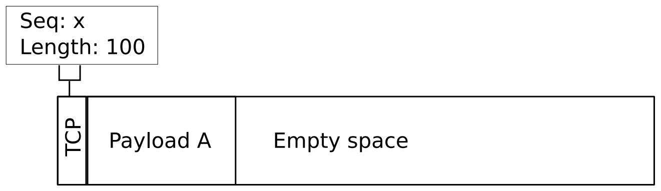
Applications that provide no interactive service may do well under such conditions. Thin-stream applications like games, however, are often interactive and may severely suffer from the extra delay. We will now describe our proposed changes to TCP which aims to improve the latency for thin streams when loss occurs.

## 3. TCP ENHANCEMENTS

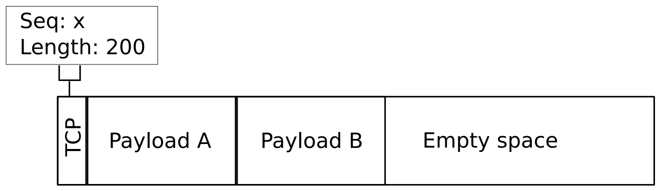
The results of our earlier investigations of reliable transport protocols [9, 11, 7] show that it is important to distinguish between thick and thin streams with respect to latency. Where high-rate streams are only concerned with throughput, the perceived quality of most thin streams depends on timely delivery of data. Working from the assumption that there is potential for large performance gains by introducing modifications that tune TCP for thin streams, we have tested the combination of several such mechanisms on typical thin-stream applications. In short, if the kernel detects a thin stream, we trade a small amount of bandwidth for latency reduction by enabling modifications to TCP. The modifications are designed in such a way that they are transparent to the receiver (i.e., a server can run the modified TCP, and unmodified clients may still receive the benefits). Any client should therefore be able to receive the stream sent by the modified TCP version, regardless of operating system and version. The modifications can also be transparent to the application running on top of TCP (they can be enabled by using a */proc* variable, and will thus be active for all networked applications). The following modifications have been implemented in the Linux kernel (v.2.6.23.8):

- **Removal of exponential backoff:** Since most thin streams send out packets with a high IAT, more or less all retransmissions are caused by timeouts. A timeout retransmission invokes exponential backoff, which doubles the time to wait for the next retransmission.

  If the number of packets in flight (i.e., unacknowledged pack-



(a) First sent packet.



(b) Second packet: Bundled data.

**Figure 1: Method of bundling unacknowledged data.**

ets) is less than the number required to trigger a fast retransmission, we remove the exponential factor. If more than four packets are on the wire, the possibility for triggering a fast retransmit increases, and therefore exponential backoff is employed as usual. Since the streams that gain from this modification are very thin, the increase in bandwidth consumption due to the removal of exponential backoff in these cases is very small. That is, the stream does still not have to use its allowed send window.

- **Faster Fast Retransmit:** Instead of having to wait several hundred milliseconds for a timeout retransmission and then suffer from the exponential backoff, it is much more desirable to trigger a fast retransmission. This requires the connection to wait for three duplicate acknowledgements (four acknowledgements of the same packets), which is not ideal for many thin streams. Due to the high IAT in our scenario, sending three packets often takes longer than the timeout.

  We have therefore reduced the number of required duplicate acknowledgements to one, provided that the number of packets in flight is less than four. Otherwise, the chance of receiving three dupACKs increases, and regular (three dupACKs) fast retransmit is employed.

- **Redundant Data Bundling:** As shown in table 1, many thin-stream applications send small packets. As long as the combined packet size is less than the MSS, we copy (bundle) data from unacknowledged packets in the send buffer into the new packet. As many of the remaining unacknowledged data segments as possible are bundled with each new packet. This increases the probability that a lost payload will be delivered with the next packet. Figure 1 shows an example of how a previously transmitted data segment is bundled with the next packet. Notice how the sequence number stays the same while the packet length is increased. If packet a) is lost, the ACK from packet b) will ACK both segments, making a retransmission unnecessary.

As mentioned, the first two modifications are only applied when there are less than four packets in flight. Thus, we avoid that streams already using their fair share of the available resources (in accordance with TCP) consume even more using our proposed mechanisms. Redundant data bundling (RDB) [7], on the other hand, is limited by the IAT and packet size of the stream. If the packets are large, which is typical for bulk data transfer, bundles can not be made, resulting in normal TCP operation.

## 4. TEST RESULTS

The TCP modifications are most effective for interactive thin-stream applications like online games. To evaluate the effect of the mechanisms, we wanted to benchmark the position updates in a real game environment. BzFlag is an open source, FPS game where players challenge each other using tanks on a virtual battleground. As shown in table 1, it generates thin streams with an average packet IAT of 24 ms and an average payload size of 30 bytes. Thus, it is a game well suited to demonstrate the benefits of our thin stream modifications to TCP.

To collect the data needed to generate the results presented in this paper, we constructed a network consisting of five machines. Three acted as clients running computer controlled opponents (bots), one ran the server application, while the fifth machine acted as a network emulator between the server and the clients. We needed to impose loss and delay on the link between the server and the clients so that we could evaluate the effects of the modifications. After performing several measurements from different Norwegian ISP's to machines in the US and various European countries, we chose an average loss rate of 2 % and a round trip time (RTT) of 130 ms, as representative emulated network values. The three clients ran 26 bots alltogether, which represent a typical high-paced BzFlag-multiplayer scenario. To get a sufficiently large number of samples, we ran six, one hour long, tests (three with the modifications enabled and three without).

We will first present the results from analysing the dumps of the network traffic, then show how the modifications affected the difference between observed and actual positions in the game.

### 4.1 Transport layer latency

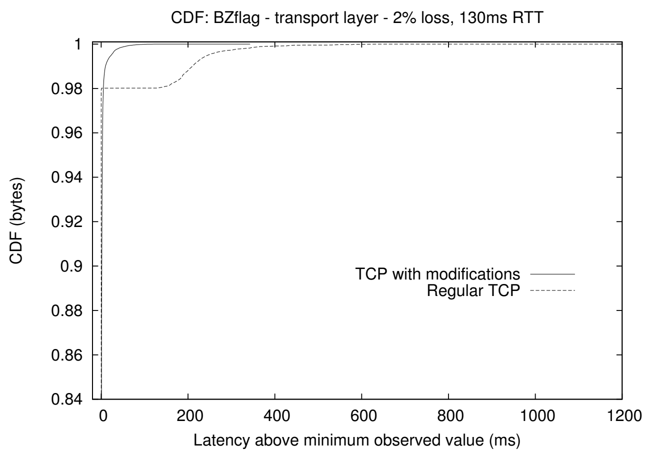
From the packet traces, we measured how much later than the one-way delay (OWD) each data segment arrived at the receiver. Figure 2 shows cumulative density functions (CDFs) of the data delivery latency. We have analysed dumps from tests performed both with and without the thin-stream modifications.

The transport layer latency is shown in figure 2(a). Both connections received 98 % of the data at minimum time (meaning the only delay experienced was the OWD). This reflects the loss rate of 2 %. When the connection experiences loss, however, the modified system recovers the data significantly faster than the unmodified TCP.

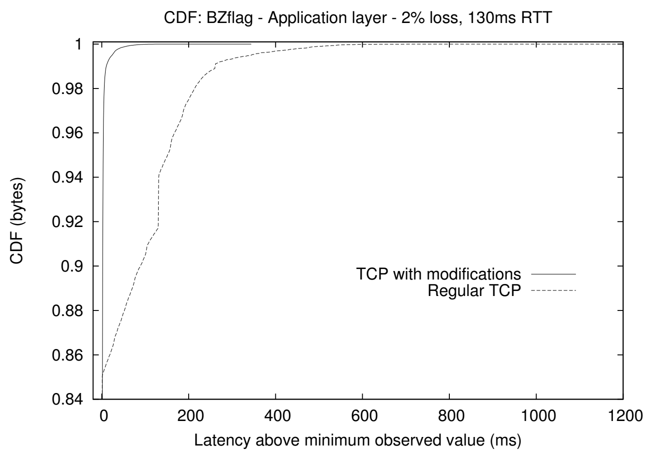
### 4.2 Application layer latency

TCP's in-order requirement states that received data segments must be stored in a buffer until a complete range of data is received before the data is delivered to the application. This means that successfully delivered packets still have to wait for any lost segments to be retransmitted and received before the application can use them. Figure 2(b) shows the application layer latency which takes the in-order requirement into account.

We can see that much smaller latencies were experienced when the modifications were enabled. Close to 99 % of the data was available to the application at the best possible latency, compared to only 85 % when using ordinary TCP (a large increase, since only 2 % of the packets are delayed at the transport level). Unmodified TCP



(a) Transport layer latency.



(b) Application layer latency.

**Figure 2: CDFs of BzFlag latency.**

is not able to deliver 99 % in less than 261 ms. It is also worth mentioning that the modifications in this case make sure that the differences between application layer delivery time and transport layer delivery time are minimal.

## 4.3 Impact of latency on perceived player positions

The reduced application layer latency also affected the user experience. Therefore, to see how the latency influenced the perceived player positions, we collected the actual and perceived position of the other players each time a chosen tank (reference tank) fired a shot. We then calculated the difference (angle) between the two positions as viewed from the reference tank. Figure 3 shows how the two vectors representing perceived and actual position was found. The angle $v$ was then calculated using the law of cosines. Position *A* represents the reference tank, *B* represents the perceived position of an opponent at the time of the shot, while *B'* represents the actual position of the opponent. Figure 5 shows that the angle of difference between perceived and actual position is smaller when the modifications were applied for a large majority of the measurements. On average, the angle between the perceived and actual position was reduced by 0.7 degrees when the modifications were

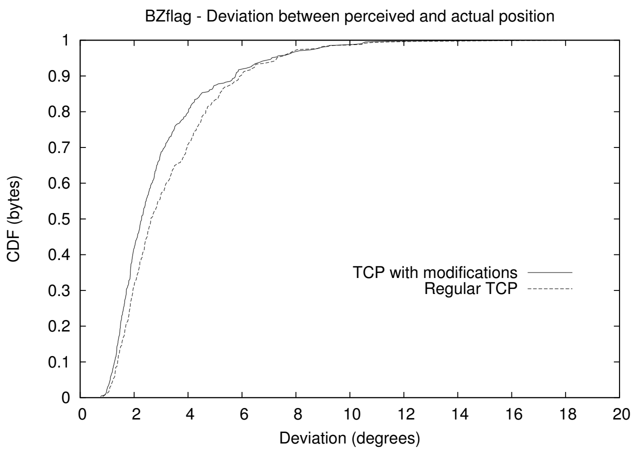


**Figure 5: CDF of difference**

enabled (from 3.5 to 2.7 degrees). The dimensions of the game field (the virtual battleground) was $200 \times 200$ world units ($wu$, BzFlag's internal distance metric). Provided that the player aimed at the centre of the opponent's tank (a "perfect shot") based on the perceived position, the angle of difference between perceived and actual position may be so substantial that a would-be hit actually evaluates as a miss. Figure 4 shows how we calculate the $wu$ deviation caused by the deviation angle $v$. Using the formula $x = n \times \tan v$, we can extrapolate the deviation in $wu$ when the distance $n$ to the target increases. Here, $n$ is the distance between the player and the opponent, and $x$ is the deviation from the actual position of the observed tank. In BzFlag, each tank is 2.8 $wu$ wide and 6 $wu$ long. A "perfect shot" would have a 100 % chance of hitting enemies when the distance to the target is less than 30 $wu$ using the modifications. Using regular TCP, the distance to guarantee a hit would have to be reduced to 23 $wu$. In practise this means that the modifications increase the chances of hitting your opponent due to a smaller deviation between perceived and actual position. The 7 $wu$ improvement is, for reference, equal to the width of 2.5 tanks in the game, a deviation that is large when trying to hit your target. We also investigated how the difference in angle affects the chance of the shot hitting when we varied the distance between the tanks.

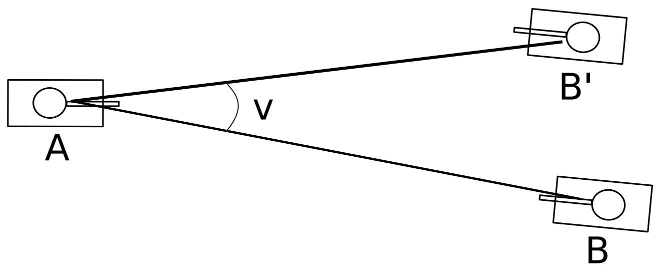


**Figure 3: Difference angle calculation.**

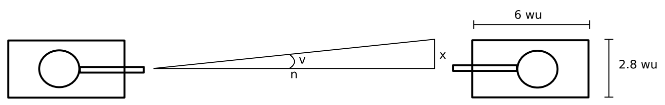
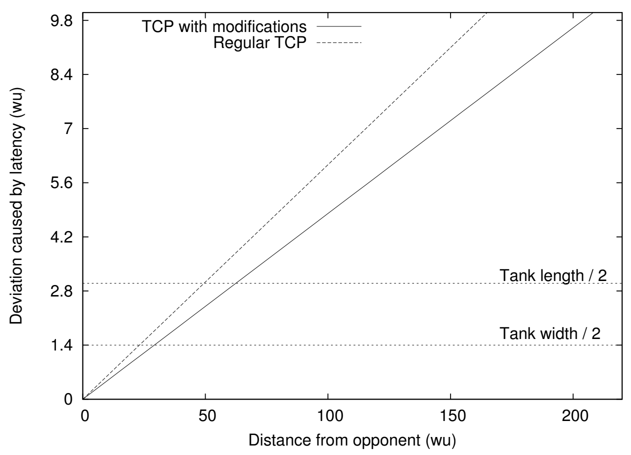
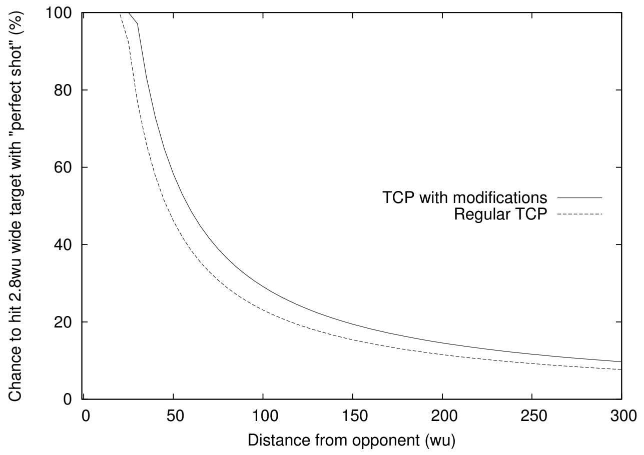


**Figure 4: Hit limit calculation.**



(a) BzFlag hit limits (world units).



(b) BzFlag hit limits (percent).

**Figure 6: BzFlag hit limits for "perfect shot".**

Figure 6(a) shows how large the radius of the hit box has to be to guarantee that the "perfect shot" is a hit given a specific distance to the opponent. The dotted lines at 1.4 $wu$ and 3 $wu$ represents the tank hit box when seen from the front and from the side. The graph shows how the effect of the modifications increases with the distance to the opponents you are trying to hit.

Figure 6(b) shows the chance of hitting your opponent with a "perfect shot" at different distances using regular TCP and the modifications. If you, as an example, fire at an opponent at a distance of 50 $wu$, there is 12 % greater chance of a hit using the modifications than without.

## 5. DISCUSSION

The thin-stream modifications to TCP are based on the concept that the applications in question never use their fair share of the bandwidth. The price paid for the reduced latency is a greater measure of redundant transmissions. For the exponential backoff- and dupACK modifications, this comes in the form of more aggressive retransmissions. For RDB there is a constant factor of redundancy. The increase is dependent on the RTT and IAT; Connections experiencing a high RTT and low IAT will have a high level of redundancy. Several packets will be sent between each ACK and thus, bundles can be made.

However, the increase in bandwidth is still small due to the fact that the applications only transmit sporadically (or at a steady low rate), also, currently RDB bundles as much data as possible into one packet. To reduce the packet size and limit the redundancy, one could set a limit for the size of bundled packets. This would help avoid the bundling of too many segments on, for instance, a high RTT connection. We can also trace a tendency in RDB to send more packets than a regular TCP connection operating under the same conditions. The reason for this increased packet rate is that RDB, unlike plain TCP, does not necessarily slow down when loss occurs (since the lost data might arrive in the next packet and be acknowledged before a retransmission is triggered), thus not halving the *cwnd*.

Even though we alter the congestion control, preliminary tests indicate that fairness is indeed preserved for the retransmission modifications. The preliminary results show no indication that the modifications inhibit the competing streams that use unmodified TCP. It will be more challenging to tell how the streams would perform if a large number of thin-streams using modified TCP should compete for the same bottleneck. In the future, we will run more complex experiments (or simulations) to determine how such a scenario will affect the balance of fairness. Towards that end, we are currently working on simulations of larger networks using thin-stream TCP-modifications.

This paper only discusses changes to TCP. The thin-stream problems will, however, be relevant to most end-to-end transport protocols that require reliability. The most used approach (for instance when implementing reliability on top of UDP) has been to model retransmission mechanisms after TCP. Viewed in the light of the latency problems of TCP retransmission mechanisms for thin streams, the proposed changes should be considered also for other protocols and applications that implement end-to-end reliability.

## 6. RELATED WORK

A huge amount of work has been performed in the area of TCP performance. The vast majority of this work, however, addresses bulk transfer, whereas time-dependent thin-stream applications are, to the best of our knowledge, hardly looked at. Nevertheless, we present here a couple of ideas that are related to our latency reduction mechanisms.

The "Limited Transmit" Mechanism [1] is a TCP mechanism that tries to do more efficient recovery when a connection's congestion window is small, or when a large number of segments are lost in a single transmission window. If the receiver's advertised window allows the transmission of more segments and the amount of outstanding data would remain less than or equal to the congestion window plus two segments, the sender can transmit new data upon the arrival of the first two duplicate ACKs. This happens without changing the congestion window.

In an IETF draft[1], Allman et al. suggest that measures should be taken to recover lost segments when there are too few unacknowl-

[1]IETF Draft draft-allman-tcp-early-rexmt-07: Mark Allman, Konstantin Avrachenkov, Urtzi Ayesta, Josh Blanton, Per Hurtig, "Early Retransmit for TCP and SCTP", June 2008, expires December 2008.

edged packets to trigger Fast Retransmit. They propose Early Retransmit (ER), which should reduce waiting times for any of four situations: **1)** the congestion window is still initially small, **2)** it is small because of heavy loss, **3)** flow control limits the send window size, **4)** or the application has no data to send. The draft proposes to act as follows whenever the number of outstanding segments is smaller than 4: if new data is available, it follows Limited Transmit [1], if there is not, it reduces the number of duplicate packets necessary to trigger fast retransmit to as low as 1 depending on the number of unacknowledged segments. Our fast retransmit-triggering mechanism has no stepwise escalation, but is fully applied when there are few packets in flight. This is because we can expect the thin stream to keep its properties throughout its lifetime and also because so many thin-stream applications are interactive with strict latency requirements. Allman et al. try to prevent retransmission timeouts by retransmitting more aggressively, thus keeping the congestion window open. If their limiting conditions change, they still have higher sending rates available. Our main goal is to keep the delivery latency low. We have no motivation to prevent retransmission timeouts in order to keep the congestion window open and retransmit early to reduce application-layer latencies.

The removal of the exponential back-off can of course result in spurious retransmissions when the RTT changes. The proposed method of TCP Santa Cruz [10] uses TCP timestamps and TCP options to determine the copy of a segment that an acknowledgement belongs to and can therefore provide a better RTT estimate. Since the RTT estimate can distinguish multiple packet losses and sudden increases in actual RTT, TCP Santa Cruz can avoid exponential back-off. The ability of Santa Cruz to consider every ACK in RTT estimation has minor effects in our scenario where few packets are generated. The ability to discover the copy of a packet that an ACK refers to would still be desirable but would require receiver-side changes that we avoid.

For FPS games, a successful approach for reducing latency is to choose servers that are favourably situated related to your location [8, 3]. For more massive, world-spanning games like MMORPGs, however, this may not be an option. Server discovery and selection approaches is, however, orthogonal to the use of our modifications and they could successfully be applied in combination.

Another interesting avenue of investigation would be to implement Early Retransmit and Limited Transmit, and perform tests to determine how the performance of these mechanisms compares to the proposed thin-stream mechanisms. It may also be possible to successfully merge a subset of known techniques to efficiently handle many of the known issues that cause latency due to congestion control and retransmissions for thin streams and special cases of a thick stream lifetime.

## 7. CONCLUSION

We have described how applications that generate small packets with high IAT suffer because of the way in which TCP congestion control handles retransmissions. We have evaluated a set of proposed mechanisms with respect to delivery latency and evaluated the effect of the mechanisms when applied to a BzFlag game using TCP.

The proposed modifications are compliant with TCP standards and transparent to the both the receiver and the application, which makes it possible to use the modifications for existing applications without any changes (as is done in the tests presented).

Analysis of the packet traces from the BzFlag sessions show that the application layer latency when loss occurs is greatly reduced when using the proposed mechanisms. Further analysis of in-game positions reveal that the chance of hitting your opponent is increased as a result of the lowered latency.

## 8. REFERENCES

[1] ALLMAN, M., BALAKRISHNAN, H., AND FLOYD, S. Enhancing TCP's Loss Recovery Using Limited Transmit. RFC 3042 (Proposed Standard), Jan. 2001.

[2] ALLMAN, M., PAXSON, V., AND STEVENS, W. TCP Congestion Control . RFC 2581 (Proposed Standard), Apr. 1999. Updated by RFC 3390.

[3] ARMITAGE, G. Optimising online fps game server discovery through clustering servers by origin autonomous system. In *International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)* (May 2008).

[4] C. CHAMBERS, W. FENG, S. S., AND SAHA, D. Measurement-based characterization of a collection of on-line games. *In the Proceedings of the 5th ACM SIGCOMM Workshop on Internet measurement, Berkeley, CA, USA* (October 2005), 1–14.

[5] CLAYPOOL, M. The effect of latency on user performance in real-time strategy games. *Elsevier Computer Networks 49*, 1 (Sept. 2005), 52–70.

[6] CLAYPOOL, M., AND CLAYPOOL, K. Latency and player actions in online games. *Communications of the ACM 49*, 11 (Nov. 2005), 40–45.

[7] EVENSEN, K., PETLUND, A., GRIWODZ, C., AND HALVORSEN, P. Redundant bundling in tcp to reduce perceived latency for time-dependent thin streams. *Communications Letters, IEEE 12*, 4 (April 2008), 324–326.

[8] GARGOLINSKI, S., PIERRE, C. S., AND CLAYPOOL, M. Game server selection for multiple players. In *NetGames '05: Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games* (New York, NY, USA, 2005), ACM, pp. 1–6.

[9] 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.

[10] PARSA, C., AND GARCIA-LUNA-ACEVES, J. J. Improving TCP congestion control over internets with heterogeneous transmission media. In *International Conference on Network Protocols (ICNP)* (Nov. 1999), pp. 213–221.

[11] PEDERSEN, J., GRIWODZ, C., AND HALVORSEN, P. Considerations of SCTP retransmission delays for thin streams. In *IEEE Conference on Local Computer Networks (LCN)* (Nov. 2006), pp. 1–12.

[12] SECOND LIFE. Second life. http://www.secondlife.com/, June 2008.

[13] STEVENS, W. TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms. RFC 2001 (Proposed Standard), Jan. 1997. Obsoleted by RFC 2581.

[14] W. FENG, F. CHANG, W. F., AND WALPOLE, J. Provisioning on-line games: a traffic analysis of a busy Counter-strike server. *In the Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurement, Marseille, France* (November 2002), 151–156.