Latency Evaluation of Networking Mechanisms for Game Traffic

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

ABSTRACT

Large improvements in computer technology allow thousands of users to concurrently interact in a virtual game. Due to this development, the body of work analyzing game traffic has grown considerably in the recent past. However, little work has been done to examine and compare networking techniques with respect to meeting the stringent latency requirements that are common for networked games. Most interactive games need response times between 100 and 1000 ms depending on the game genre [6]. In this paper, we evaluate different techniques for delivering packets in a timely manner. In particular, we compare existing user-space middleware running on top of UDP and reliable, fair transport protocols like TCP and SCTP. In addition, we evaluate some "low latency" extensions to TCP, SCTP and one of the middleware platforms. We present results concerning packet latency and bandwidth requirements for the different approaches.

1. INTRODUCTION

In recent years, online gaming has grown to a huge industry as network technology allows users world-wide to interact in a virtual game. These applications enable users to form and maintain social bonds, compete against each other and have fun in a virtual environment. However, there are stringent latency requirements depending on the interaction model, which differs between game genres. Thus, the success of interactive games raises a very challenging system requirement, *low latency* for all users. As most games require a packet to arrive within less than one second to give the users a satisfactory gaming experience [6], it is vital to have appropriate networking support. A significant characteristic of this type of application is therefore a lack of resilience towards transmission delays [5, 10]. Interestingly, it is an aggravating factor that an individual game stream hardly ever consumes the bandwidth that constitutes its fair share according to the ideas of TCP-fairness. The reason is that these streams are very *thin* – they consist of small packets sent at low packet rates.

Many different networking mechanisms have been used to meet the challenging requirements to support gaming. These range from standard versions of the TCP/IP transport protocols to specialized user-space middleware. In this paper, we focus on mechanisms that provide some reliability. We do not compare the tested systems with pure UDP. Reliable and partially reliable communication has recently been ignored as researchers focus on increasing throughput for non-interactive applications with high bandwidth demands.

For the online gaming scenario, we have tested and compared two standard TCP versions, SCTP and two user-space middleware platforms that use UDP. Additionally, we have run tests applying "low latency" enhancements for TCP, SCTP and one of the middleware platforms to trade bandwidth for latency reduction. With the exception of SCTP, which generally has a higher latency, the approaches perform similarly with respect to the average latency. However, for the worst-case latencies that are devastating for the perceived gaming quality, there are large differences. The traditional transport protocols have large maximum values, while the user-space middleware platforms perform slightly better. Finally, our results show that we can efficiently trade off some bandwidth for lower latencies, both in user space and in the kernel.

The rest of this paper is organized as follows: Section 2 presents some stream characteristics for games. In section 3, we briefly describe some existing mechanisms with respect to latency, and we present our experiments and the results in section 4. In section 5, we discuss our findings, and finally, we give some conclusions in section 6.

2. GAME STREAM CHARACTERISTICS

The body of work that has analyzed game traffic has grown considerably in the recent past. For example, Claypool [5] has investigated how latency affects the perceived quality for real-time strategy games where the results show that some latency is tolerable. Moreover, Feng et al. [8, 3] provide a comprehensive analysis of Counter-Strike traffic and investigate traces of several games concerning the predictability of game workloads. There, the conclusion is that game traffic varies strongly with time and with the attractiveness of the individual game. Chen et al. [4] investigate the traffic of MMORPGs. They find that streams are in general very thin, but that they are also bursty and show a correlation of inter-arrival times on the minute scale within individual streams. Furthermore, fitting multi-player game traffic to probability distributions is described by Borella [2].

Submitted to NetGames'07 Melbourne, Australia

application (platform)	payload size (bytes)			packet interarrival time (ms)						avg. bandwidth requirement	
								percentiles			
	average	min	max	average	median	min	max	1%	99%	(pps)	(bps)
Anarchy Online (PC)‡	98	8	1333	632	449	7	17032	83	4195	1.582	2168
World of Warcraft (PC)	26	6	1228	314	133	0	14855	0	3785	3.185	2046
Counter Strike (PC)‡	36	25	1342	124	65	0	66354	34	575	8.064	19604
Halo 3 (Xbox 360)†‡	247	32	1264	36	33	0	1403	32	182	27.778	60223
Halo 3 (Xbox 360)†‡	270	32	280	67	66	32	716	64	69	14.925	35888
Gears of War (Xbox 360)‡	66	32	705	457	113	3	10155	14	8953	2.188	10264
Tony Hawk's Project 8 (Xbox 360)‡	90	32	576	308	163	0	4070	53	2332	3.247	5812
Test Drive Umlimited (Xbox 360)‡	80	34	104	40	33	0	298	0	158	25.000	22912

† For Halo 3 (beta version), we also show differences between intensive (the upper row) and moderate (the lower row) action.
‡ The presented values are average values over all players (sending minimum 1000 packets) within the period of the trace.

Table 1: Examples of game stream packet statistics per stream based on packet traces

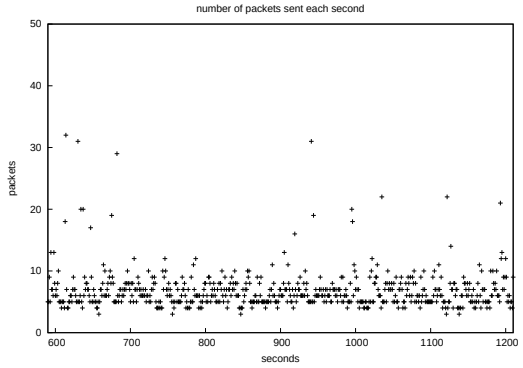


Figure 1: Packets per second for World of Warcraft

However, little work has been performed analyzing and comparing the various networking techniques with respect to meeting the stringent latency requirements. Games typically require tight timeliness, with latency thresholds (before the users starts getting annoyed) at approximately 100 ms for first person shooter (FPS) games, 500 ms for role playing games (RPG) and 1000 ms for real time strategy (RTS) games [6]. We analyzed traces from several types of games to understand more of the properties of thin stream traffic. We also wanted some more data as a supplement to the analysis found in [4, 7]. Some of the results of our game trace analysis are shown in table 1. The statistics show that the games have a small packet size and a relatively high interarrival time between packets. For example, Funcom's popular role playing MMOG *Anarchy Online* send less than one packet per RTT on average, that means, the packet interarrival time is large with a median of about 450 ms. Additionally, each packet contains only small game events, like position updates, giving an average payload of 93 bytes. A second example is a FPS game trace from a popular *Counter Strike* server. This game shows similar properties as Anarchy Online with a small payload (in average 142 bytes) and a large interarrival time (median about 50 ms) with about 20 packets per second (pps).

Considering that each game data stream has *few* packets per second and that each packet is very *small*, this demonstrates that the individual streams are thin. Moreover, it is also typical to occasionally have small bursts of packets. See for example the 10 minute I/O-graph in figure 1 showing the number of packets sent each second. We further observe that the RTTs are within the range of making the game playable according to [5], and it is not the loss rate itself that is unacceptable, but the occasional large delays when multiple retransmissions are needed. Some extreme examples are shown in the traces, for example, the Anarchy Online trace had a worst case delay of 67 (!) seconds when several retransmissions of the same packet were lost.

With the low latency requirements in mind due to the interactive nature of the games, supporting these kinds of applications is challenging. The task becomes even harder since a significant characteristic of this type of application is its lack of resilience towards network transmission delays [5]. Next, we therefore briefly outline some alternatives for sending game packets before we present our experimental evaluation.

3. NETWORKING MECHANISMS

Online games have many features that require appropriate networking support. For example, games have stringent latency requirements, and some of the data require reliable or in-order delivery. This makes the choice of mechanisms for communication important. The basic transport protocols TCP and UDP are meant for a subset of the features that an online game requires. It is therefore common to implement application layer libraries or middlewares that can extend on the basic services. In this section, we describe the services provided by the basic protocols and two chosen extensions regarding thin stream traffic. In addition, we describe protocol modifications that are meant for latency reduction in this scenario.

3.1 TCP

TCP provides means for flow control, congestion control, ordered delivery and reliability, and it is widely used for all kinds of applications. Important in the gaming scenario is that reliability comes at the price of retransmissions that increase the application-layer latency. Games like World of Warcraft and Anarchy Online use TCP, mainly due to firewall issues, and we also expect that TCP will be a popular choice for this kind of games in the future.

TCP comes with a large number of variations with a multitude of options designed for various scenarios. For the purpose of online game support, we used *New Reno*. We chose it due to the observation that the younger high-speed variants perform worse than the more basic approaches [10]. In addition, we tested *BIC* since it is the current default TCP variant of several Linux distributions.

TCP New Reno [9] is the most basic TCP variant included in the Linux kernel. The two main means for triggering retransmissions are fast retransmit and timeout. The fast retransmit mechanism [1] triggers a retransmission when three duplicate ACKs (dupACKs) are received. However, games have large packet interarrival times (see table 1), and retransmission is often triggered by timeout (waiting too long for an acknowledgment). An issue for time-dependent traffic is then the exponential backoff whenever the same segment

is lost several times, as it causes very large application-layer delays. Congestion mechanisms do not affect thin game streams at all because their bandwidth requirements can be fullfilled by sending with the smallest possible congestion window at all times.

We include TCP's binary increase congestion control (BIC) [15] only for completeness. This variant tries to find the fair share of the bandwidth faster than New Reno by performing a binary search for the number of packets that can be sent per RTT without packet loss. However, this happens only between two thresholds for the congestion window size. Games traffic stays below the lower threshold where New Reno's congestion window development applies.

In addition to the existing TCP variations in Linux, we have tested two "low latency" enhancements for thin streams. First, the exponential backoff is removed to avoid the extreme latencies that occur when experiencing multiple consecutive packet loss. In addition, the fast retransmission mechanism is modified to trigger a fast retransmission after only one duplicate ACK is received. This modification is labeled *modified TCP* in our tests. Second, when the packet sizes are small (as is common for online game traffic), an aggressive bundling scheme can help to reduce the application delay. TCP with redundant data bundling (RDB), labeled *TCP with RDB* in our tests, is a modification of TCP that enables bundling of unacknowledged data if there is room in the packet. Thus, a lost packet may be recovered when the next original packet is received without a retransmission.

3.2 SCTP

The Stream Control Transmission Protocol (*SCTP*) [14] was originally developed to support public switched telephone network signaling traffic over IP networks, but is supposed to offer services like low latency delivery that can be of use to other applications as well. A salient difference from TCP is that it is message-oriented (not byte-oriented). SCTP offers the option of bundling several messages in one packet and also supports multi-homing for enhanced fault tolerance. With respect to retransmissions, they are mainly handled using fast retransmission and timeout mechanisms as in TCP. However, with respect to our scenario, important issues include a high minimum retransmission timeout (minRTO) value of 1000 ms and delayed ACKs, which both increase the latencies.

As with TCP, we have also tested "low latency" enhancements to SCTP. Means for bundling are already included, so we have included the exponential backoff and fast retransmission modifications described for TCP. In addition, the minRTO value is lowered to 200 ms (the standard TCP value). This modification is labeled *modified SCTP* in our tests.

3.3 UDP-based Libraries and Middleware

For applications where reliability is not needed, UDP is the most common alternative. There are, however, no congestion control mechanisms, and a batch of aggressive UDP streams can create bottlenecks and congestion in the network. When using UDP in a game, critical data (where reliability is needed) has to be supported through other means. This is done by implementing reliability, congestion control, and support for ordering on the application layer. The control data is in the payload, creating a small overhead, but it has the advantage that the degree of reliability can be chosen for each transmitted packet. Such network libraries and middleware come in many variants and levels of abstraction. The simplest ones are tiny libraries that provide a small set of services. More complex middleware platforms provide a range of services with a high level abstraction layer for easy integration with different components. Some examples include ACE, ENet, HawkNL, Plib, SDL, ClanLib, Net-Z, OpenTNL, RakeNet, ReplicaNet, UDT and ZoidCom. Testing all these variations is beyond the scope of this paper. We have therefore selected two approaches which use different mechanisms to schedule the packets. We wanted one low level network library, and one middleware platform with a higher level of abstraction. In addition, we needed to be able to look at how they perform retransmissions, and we selected two open source mechanisms: the UDP-based Data Transfer Protocol (UDT) and ENet.

UDT [11] provides various features based on a UDP connection. A structured (socket-like) interface provides a high level of abstraction making it easy for application developers to use. Partial reliability and in-order delivery are supported, and congestion control mechanisms enables UDT to maintain TCP fairness. There are also a wide range of parameters that can be set in order to achieve the combination of options that is best for each application. Moreover, UDT divides its packets into control and data messages. In order to keep track of the status of the remote host, keep-alive messages are an integral part of the framework. This combined with aggressive bundling strategies, used if less than the estimated bandwidth is consumed, contributes to a large redundancy rate for the UDT platform. Retransmissions are managed as for TCP with timeouts and dupACKs, and additionally using negative acknowledgments. The range of options makes this a promising choice for game developers. Although the middleware was developed for data-intensive applications, the aggressive transmission policy should provide ample support for thin stream applications.

ENet[1] aims for online gaming support. It was developed for the Cube game engine and is later also used by other networked games (e.g., the commercial SilentWings flight simulator). ENet provides a relatively thin, simple and robust network communication layer on top of UDP. It provides optional, reliable, in-order delivery of packets. ENet is a small library that provides some functionality without supplying a high level of abstraction and can therefore not be considered to be a middleware platform. The services include a connection interface for communicating with the foreign host. Delivery is optionally stream oriented or message oriented. The state of the connection is monitored by pinging the target, and network conditions such as RTT and packet loss are recorded. Partial reliability is supported, and retransmissions are triggered using timeouts based on the RTT like similar to TCP. The congestion control implements exponential backoff, making it vulnerable to bursts of loss, and ENet also applies bundling of queued data if the maximum packet size is not reached.

4. EXPERIMENTS AND RESULTS

To acquire statistics of the performance of various Linux networking mechanisms concerning latency in a gaming scenario, we ran several tests. Small packets (100 bytes) are sent at a low rate (packet interarrival times between 50

[1]http://enet.cubik.org

and 200 ms) in accordance with the game characteristics described in section 2. To emulate the network, we used *netem* to introduce delays (RTTs between 50 and 200 ms) and packet loss (between 0.1 and 2.5% in each direction). Each of the tests had a 2 hour duration.

For our comparison, we tested the protocols and mechanisms briefly described in section 3. New Reno had no options turned on whereas BIC used SACK, FACK and DSACK. SCTP uses SACK by default. For this interactive scenario, we turned off Nagle's algorithm [12] for both TCP and SCTP. Additionally, we tested the TCP and SCTP with thin-stream modifications and the RDB algorithm. Finally, we have compared the transport protocol techniques with user space approaches (UDT and ENet) that provides the same reliability using UDP.

4.1 Perceived Latency

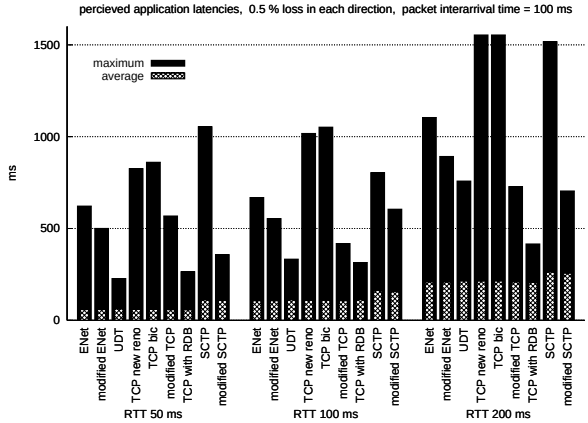
One of the most important aspects of the interactive gaming scenario, is the system's ability to deliver data in time. The measured results for the different mechanisms listed above are shown in figure 2[2]. The first observation is that the average results are very similar, except for SCTP which generally has higher latencies. Thus, with respect to average latency, all the TCP and UDP based approaches seem to be usable.

However, looking at the worst case scenario, we can see large differences due to the way data recovery is performed (see section 3). These maximum latencies are dependent on the number of retransmissions of the same packet. Since the loss is random, the maximum number of consecutive lost transmissions of a packet varies. The figures nevertheless give a representative picture of the performance of the different mechanisms. The plots in figures 2(a), 2(b) and 2(c) show the latency varying the RTT, packet interarrival time and loss, respectively. When the interarrival time is equal to (or higher than) the RTT, we see that retransmissions from timeouts and backoffs give very large values for ENet, TCP New Reno, TCP BIC and SCTP. By changing the retransmission mechanisms as described where applicable, we can achieve large latency improvements. This comes at the cost of a possible increase in spurious (unnecessary) retransmissions. Another (orthogonal) way to improve the latency is to send multiple copies of a data element by bundling unacknowledged data in succeeding packets like in UDT (when using less than the estimated bandwidth), TCP with RDB and modified SCTP. The modifications increases aggressiveness in (re)transmissions, and may have an impact on fairness. We will therefore next examine the bandwidth tradeoff resulting from these changes.

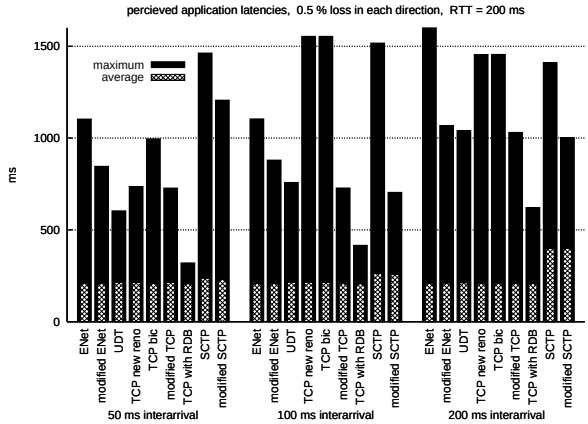
4.2 Bandwidth Requirement

Adding support for reliability comes at the price of retransmitting lost packets, and trying to reduce the retransmission latency increases the bandwidth requirement further. To quantify the tradeoff, we have measured the number of bytes per second (Bps) and the number of packets per second. Figure 3 shows the required bandwidth corresponding to the achieved latencies in figure 2(a) where the packet interarrival time is 100 ms and the loss rate is 0.5%

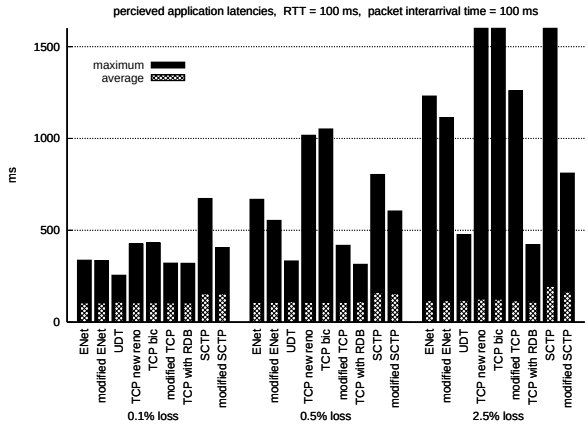
[2]When the loss rate is large (2.5%) and RTT is high (200ms), the standard TCP variants and SCTP have maximum values well above 2000 ms, although the scale of the figure is limited at 1500.



(a) Latency vs. RTT



(b) Latency vs. packet interarrival time



(c) Latency vs. loss rate

Figure 2: Perceived latencies

in both directions. On the right y-axis, the figure also shows the relative bandwidth compared to the IP payload of pure UDP packets. With respect to the number of bytes sent, the traditional TCP variants are best regardless of loss, packet interarrival time and RTT. Compared to the user space libraries, the kernel TCP and SCTP transport protocols do not add an additional header. SCTP and ENet are both sightly higher than TCP. The "modified TCP" has stan-

dard TCP headers, but may send more (and more frequent) retransmissions. However, the data and loss rates are low, so the increase is negligible. The most resource consuming approaches are UDT and TCP with RDB. The reasons are that UDT always tries to use all the estimated bandwidth, and RDB bundles previous packets as long as there are unacknowledged data in the queue and the size limit for the new packet is not reached. Thus, when the RTT is low, UDT will resend a packet multiple times to fill the pipe, and the overhead for TCP with RDB will increase with the amount of unacknowledged data. In terms of the number of bytes, TCP with RDB will be expensive with higher packet rates and longer RTTs.

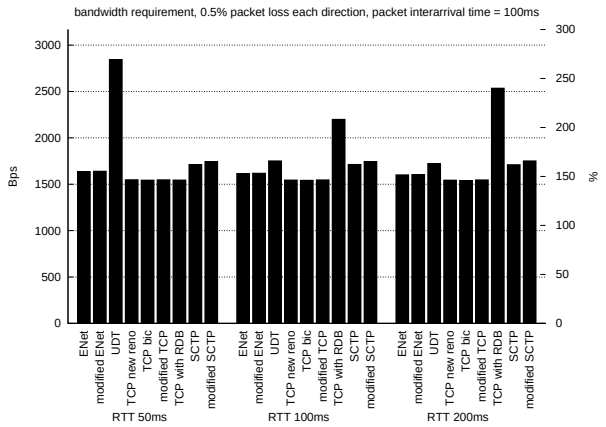


Figure 3: Measured bandwidth

Another way of comparing the overhead is to look at the number of packets sent (see figure 4). Many consider this as more relevant because of the fixed time between packets the on transmission media. For example, the minimum size of an Ethernet frame is 64 bytes corresponding to the Ethernet slot time of 512 bits used for carrier sensing and collision detection at 10 and 100 Mbps. For gigabit Ethernet, the slot is increased from 512 to 4096 bit. Thus, it can be said that space may be wasted if the packets are not filled – at least to the slot size. In our plots, the application sends approximately 10 packets per second (actually marginally less in average due to the timing granularity in user space). Traditional TCP follows this rate and since only a few packets are lost, the measured packet rate is approximately 9.6 pps. ENet and SCTP (as well as the modified versions) both send a few more packets. As UDT always try to fill the estimated pipe, the packet rate is large, e.g., for an RTT of 50 ms, UDT sends about 29 packets per second. Finally, TCP with RDB sends slightly less packets compared to standard TCP since we did not experience any retransmissions using RDB.

5. DISCUSSION

The loss induced by netem in our tests is random without tendency for burstiness. This results in only a small number of 2nd and 3rd retransmissions. If burstiness had been introduced in the loss-generating mechanisms, we would see more consecutive retransmissions, and the latency due to exponential backoff would be increased. However, with respect to the maximum values, our results are representative, and a higher frequency of multiple losses would only slightly increase the average values.

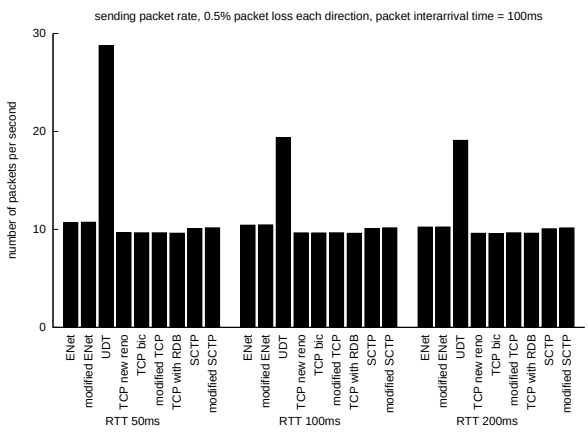


Figure 4: Number of packets sent

TCP with RDB performs very good in all our tested scenarios with respect to latency. This is due to the small packet size that makes it possible to bundle many data "segments". With larger packet sizes, more retransmissions would have occurred, and latency performance would have dropped. UDT on the other hand makes use of frequent "spurious" retransmissions to fill its estimated bandwidth. Designed for high-bandwidth environments, the maximum latency performance of UDT drops when the packet interarrival time and the packet size increase.

The modified TCP and SCTP performs better than its unmodified counterparts both for max and average values. The difference increases with larger interarrival times and RTTs. This is due to the degenerating performance of the retransmission mechanisms when fewer packets are sent in an RTT. The modifications helps improving the situation without the large amount of redundancy introduced by RDB and UDT. This can be an important aspect when very many concurrent streams compete for the bandwidth. SCTP is constantly above its TCP counterparts due to the (non-optional) delayed ACK mechanism. This is a point of concern since SCTP originally was intended for time-dependent signaling traffic [14, 13].

When the loss rate is high, all mechanisms that implement exponential backoff suffer severely with regard to latency. The experimental removal of exponential backoff for ENet results in better performance. We can therefore assume that the other modifications (adjustment of minRTO and number of needed dupACKs) can also be successfully integrated in middleware platforms intended for networked games.

From figure 2(b) we can see that the performance of different mechanisms varies with the packet interarrival time. As shown in section 2, the game genres all have different packet interarrival times. Thus, the packet generation pattern for each game should be taken into consideration when choosing the networking platform.

As we can see from the results, there may be large differences in the latencies achieved by the tested mechanisms under the different conditions. In general, standard TCP variations are outperformed by the other mechanisms when there are few packets in flight. This is one of the reasons why game developers use middleware on top of UDP to meet their requirements. However, our results also show that large improvements with respect to latency can be achieved

with small modifications by trading off some bandwidth. On the other side, fairness with respect to sharing the resources equally is highly valued in the networking community and increasing the consumed resources for some applications may therefore be a critical point. However, the data rates are low, and looking at the latency gain, we consider this a worthwhile tradeoff.

Finally, the analyzed game traces show that the packet size and rate typically vary according to the number of players and the interaction pattern. This indicates that different means should dynamically be applied to the stream of packets to meet the latency requirements without trading off too much bandwidth (and thereby influencing the fairness ideal). For example, if there are few players and they hardly interact, there will be very few packets. In this case, retransmissions are mainly due to timeouts, and modifications to the fast retransmit and exponential backoff mechanisms could be applied if this situation occurs. On the other hand, if many players come into the same region and interact, the packet rate increase rapidly, and modifications like the ones mentioned above could be turned off if it jeopardizes the fairness. Thus, integrated monitoring of the streams should be used to dynamically apply the appropriate mechanisms when the stream is thin.

6. CONCLUSIONS

Interactive online games raises a challenging requirement with respect to *low latency*. However, the systems today lack appropriate networking support, because most online games will never consume the full bandwidth that constitutes their fair share. The data streams are very *thin*, meaning that they have small packets and low packet rates. Thus, traditional mechanisms fail in this scenario, because they assume that the application will always fill the pipe.

In our work comparing different networking techniques, we found that the average latencies for the majority of the approaches make the game playable. The figures for worst-case latency, however, show large variations. The high maximum values are due to multiple consecutive retransmissions and can be devestating to the perceived gaming experience. The traditional transport protocols have large maximum values, while the user-space middleware platforms perform slightly better. Finally, by trading off some bandwidth for lower latencies, both in user space and in the kernel, better results may be achieved.

With respect to ongoing and future work, we are currently investigating several in-kernel mechanisms reducing the latency and how to dynamically apply them according to the oscillating behavior of the different streams. For this work in particular, we will try to evaluate a larger number of the existing user-space middlewares and compare them to the kernel modifications.

7. ACKNOWLEDGEMENTS

Funcom has provided traces from Anarchy Online, Wu-chang Feng and Wu-chi Feng have provided traces from Counter Strike. Kristian Evensen has provided traces for the other games analyzed in this paper.

Kristian Evensen has also implemented the RDB technique for TCP, and Jon Pedersen has implemented the modifications for SCTP.

Andreas Petlund has been partially sponsored by the Norwegian Research Council under contract number 159992/V30 – the MiSMoSS project.

8. REFERENCES

[1] M. Allman, V. Paxson, and W. Stevens. TCP Congestion Control . RFC 2581 (Proposed Standard), Apr. 1999. Updated by RFC 3390.

[2] M. S. Borella. Source models of network game traffic. *Elsevier Computer Communications*, 23(4):403–410, Feb. 2000.

[3] C. Chambers, Wu-chang Feng, S. Sahu, and D. Saha. Measurement-based characterization of a collection of on-line games. In *Proceedings of the USENIX Internet Measurement Conference (IMC)*, pages 1–14, 2005.

[4] K.-T. Chen, P. Huang, C.-Y. Huang, and C.-L. Lei. Games traffic analysis: An MMORPG perspective. In *Proceedings of the International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, pages 19–24. ACM Press, 2005.

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

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

[7] M. Claypool, D. LaPoint, and J. Winslow. Network analysis of counter-strike and starcraft. In *Proceedings of the IEEE International Performance Computing and Communications Conference (IPCCC)*, pages 261–268, Apr. 2003.

[8] W.-c. Feng, F. Chang, W.-c. Feng, and J. Walpole. Provisioning on-line games: a traffic analysis of a busy Counter-strike server. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurement*, pages 151–156, 2002.

[9] S. Floyd, T. Henderson, and A. Gurtov. The NewReno Modification to TCP's Fast Recovery Algorithm. RFC 3782 (Proposed Standard), Apr. 2004.

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

[11] Y. Gu and R. L. Grossman. UDT: UDP-based Data Transfer for High-Speed Wide Area Networks. *Computer Networks (Elsevier)*, 51(7), May 2007.

[12] J. Nagle. Congestion control in IP/TCP internetworks. RFC 896, Jan. 1984.

[13] L. Ong, I. Rytina, M. Garcia, H. Schwarzbauer, L. Coene, H. Lin, I. Juhasz, M. Holdrege, and C. Sharp. Framework Architecture for Signaling Transport. RFC 2719 (Informational), Oct. 1999.

[14] R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, and V. Paxson. Stream Control Transmission Protocol. RFC 2960 (Proposed Standard), Oct. 2000. Updated by RFC 3309.

[15] L. Xu, K. Harfoush, and I. Rhee. Binary increase congestion control for fast long-distance networks. In *Proceedings of the Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, 2004.