



empirical evidence, we seek a neutral answer to the question that whether TCP is suitable for MMORPGs. To the best of our knowledge, this work is the first to analyze real-life game traces in terms of network performance problems.

Based on a 1,356-million-packet trace from *ShenZhou Online* [1], a TCP-based, commercial, mid-sized MMORPG, we analyzed the performance problems related to the choice of TCP in terms of protocol overhead, in-order delivery, congestion control, and loss recovery. Our evaluation indicates that, for various reasons, *TCP is unwieldy and inappropriate for MMORPGs*. The major findings are as follows:

- Because game packets are generally small, the TCP/IP header takes up 46% of the total bandwidth used. Specifically, packet headers take up *four times more bandwidth* than the application payload in client traffic, and consume *half the bandwidth* of server traffic.
- TCP ensures in-order delivery at the byte level, which is not actually needed for every game message. Because even a single dropped packet causes a stall in subsequent network data until that packet is successfully delivered, 7% of connections incur more than 20% additional average transmission latency, and 6% incur at least 200% additional delay jitter (standard deviation of transmission latencies).
- The congestion control mechanism is *ineffectual* since game traffic is *application-limited*, rather than network-limited. However, 12% of client packets and 18% of server packets face a congestion window reset before they are released. This leads to additional latency whenever a burst of a few commands is generated following a short period of thinking.
- The fast-retransmit algorithm is *ineffective* because of *low packet rate* and *bi-directional traffic*. As a result, more than 99% of dropped packets are not detected by the sender until retransmission timeout occurs. This leads to an average latency of 700 ms for dropped packets, while the average latency for normal (not dropped) packets is 185 ms. To overcome this problem, the selective acknowledgement (SACK) mechanism, which is relatively unaffected by distinctive features of game traffic, should be enabled whenever TCP is used.

Having found that TCP leads to a significantly increase in network performance measures, we continue to examine its impact on user behavior. According to a model describing the relationship between session time and network QoS factors [7], we estimate that the departure rate of game players will decrease by 20% if the additional delay jitters induced by TCP could be avoided. This corresponds to an increase of the median game playing time from 100 minutes to 135 minutes in our case. Finally, a number of design guidelines are proposed by exploiting the unique characteristics of game traffic.

The remainder of this paper is organized as follows. Section 2 describes related works. We discuss the collection and summary of trace in Section 3. In Section 4, we briefly elucidate the characteristics of MMORPG traffic. Next, in Section 5, we analyze the performance problems induced by TCP from several aspects. We discuss the implications of the analysis and protocol design guidelines in Section 6. Finally, Section 7 draws our conclusion.



Figure 1: A screen shot of *ShenZhou Online*

## 2. RELATED WORK

In [10], Mauve et al proposed Real-Time Application Level Support for Distributed Interactive Media (RTP/I). This is an extension of the Real-Time Transmission Protocol (RTP), and has been proposed as the standard protocol framework for distributed interactive applications, namely, shared whiteboards, distributed virtual environments, and network games. Claiming that TCP is too heavy due to its complex congestion control algorithm and byte-oriented window scheme, Pack et al [11] proposed the Game Transport Protocol (GTP), optimized to meet the various requirements of MMORPGs. GTP uses a packet-oriented, rather than byte-oriented, window scheme to accommodate the small packet size of game traffic. Furthermore, to meet the realtime constraints of transmission, it supports an adaptive retransmission scheme that controls the maximum number of retransmissions according to packet priority.

Our work differs from earlier studies in that we analyze the performance problems induced by the predominant transport protocol, TCP, rather than directly propose a new protocol. Having demonstrated that TCP is unsuitable for MMORPGs, based on the understanding gained from empirical analysis, we proposed a number of design guidelines that exploit the unique characteristics of game traffic.

## 3. TRACE COLLECTION

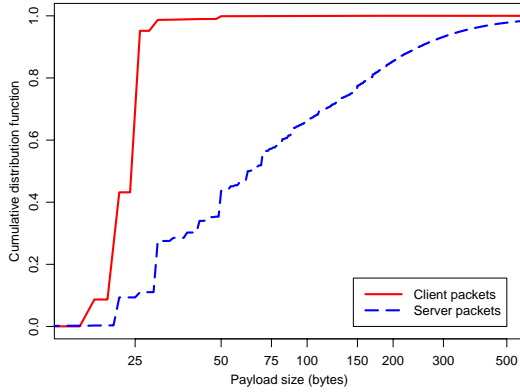
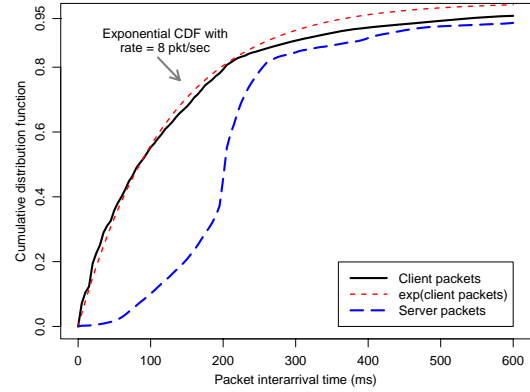
*ShenZhou Online* is a TCP-based, mid-scale, commercial MMORPG that is very popular in Taiwan [1], where there are thousands of players online at any one time. To play, the participants purchase “game points” from a convenience store or online. A screen shot of *ShenZhou Online* is shown in Fig. 1. As is normal in MMORPGs, a player can engage in fights with the other players or with random creatures, train himself in special skills, participate in marketplace commerce, or take on a quest. With the help of the *ShenZhou Online* staff, we set up a traffic monitor beside the game servers. The collected two traces,  $\mathcal{N}_1$  and  $\mathcal{N}_2$ , which spanned 8 and 12 hours respectively and contained more than 1,356 million packets, are summarized in Table 2. Due to the lack of space, we refer interested readers to [6] for the details of the traced game and measurement setup.

**Table 2: Summary of Game Traffic Traces**

| Trace           | Sets | Date    | Time  | Period | Drops <sup>†</sup> | Conn. (Cens.) | Pkt. (in / out / both) | Bytes (in / out / both) |
|-----------------|------|---------|-------|--------|--------------------|---------------|------------------------|-------------------------|
| $\mathcal{N}_1$ | 3    | 8/29/04 | 15:00 | 8 hr.  | 0.003%             | 57,945 (6.5%) | 342M / 353M / 695M     | 4.7TB / 27.3TB / 32.0TB |
| $\mathcal{N}_2$ | 2    | 8/30/04 | 13:00 | 12 hr. | ? <sup>‡</sup>     | 54,424 (3.5%) | 325M / 336M / 661M     | 4.7TB / 21.7TB / 26.5TB |

<sup>†</sup> This column gives the kernel drop count reported by *tcpdump*.

<sup>‡</sup> The reported kernel drop count was zero, but we found that some packets were actually dropped at the monitor.


**Figure 2: Payload size distribution**

**Figure 3: Packet interarrival distribution**

## 4. GAME TRAFFIC CHARACTERISTICS

In this section, we briefly elucidate the characteristics of MMORPG traffic within individual connections. Only information useful to the analysis of TCP performance is mentioned (readers can refer to [6] for more details of MMORPG traffic characteristics). For brevity, we denote packets sent by game clients, including data packets and TCP acknowledgement packets, as “client packets”, and all traffic sent by clients as “client traffic.” The terms “server packets” and “server traffic” are similarly defined.

Fig. 2 shows the cumulative distribution function (CDF) of the payload size. As the figure shows, client packets and server packets are drastically different in payload size. The discrepancy conforms to our intuition, since client packets contain *one* player’s commands, whereas server packets convey nearby characters’ actions and states as well as system messages. The client packets are extremely *small*: 98% of them have a payload size smaller than 32 bytes. The two modes 23 and 27 bytes, which comprise 36% and 52% of packets respectively, show that user actions are *dominated by a few popular commands*, such as “walk” and “attack.” In contrast, server packets have a much wider distribution with an average payload size of 114 bytes.

We now turn to the distribution of packet interarrival times. As Fig. 3 shows, most client inter-packet times are spread over 0 ms to 600 ms. We find that the best-fit exponential distribution with a rate of 8 pkt/sec approximately fits the empirical cumulative distribution function; however, the deviation of the exponential-fit is apparent at time scales larger than 200 ms. Detailed investigation shows that the deviation from the exponential distribution of inter-packet times is caused by the *diversity of user behavior* [6], which is a distinct feature of adventure-oriented games, including MMORPGs. On the other hand, server packet interarrivals

are much more regular—approximately 50% of the interarrival times are around 200 ms. The concentration of inter-packet times at certain intervals shows that servers broadcast information to clients on a regular basis.

## 5. PERFORMANCE ANALYSIS

TCP is sophisticated in that it comprises mechanisms such as reliable transmission, in-order delivery, congestion control, and flow control. In the following, based on the collected game traces, we analyze the performance problems induced by the interplay of TCP and the design of MMORPGs. We show that congestion control and loss recovery are ineffective for MMORPGs, while in-order delivery can cause unnecessary transmission latencies.

### 5.1 Protocol Overhead

TCP guarantees reliable transmission through a positive acknowledgement policy, which assumes that a segment is not successfully received by the destination host until it is acknowledged. By this policy, whenever a host receives a segment, it must generate an acknowledgment (ack for short) packet to notify the sender of receipt of the specific segment. To reduce the number of ack packets, [5] proposed a delayed ack option that only sends back an ack for alternate TCP segments received by a connection, unless the delayed ack timer (usually set to 200 ms) has expired. Ideally this design should reduce the number of ack packets to half the number of data packets if the majority of successive data segments arrive at the destination in intervals of less than 200 ms. This is the case with bulk data transfers, where the data packets are sent in bursts. However, game packets are generated at a low rate, so it is more unlikely that any further packets will arrive before the expiration of the delayed ack timer. We observe that the number of client

ack packets is much more than half the number of client data packets (the ideal case). This is because approximately 40% of server packets are released at intervals longer than 200 ms following the release of the preceding packet (cf. Fig. 3). In the overall trace, 38% of packets are pure TCP acks.

Owing to the relatively large proportion of pure ack packets and the small packet size, the overhead of the TCP/IP header is very large compared to the volume of the application payload. In the traces, the TCP/IP header takes up 46% of the total bandwidth used. Specifically, packet headers take up *four times more bandwidth* than the application payload in client traffic, and consume *half the bandwidth* of server traffic.

## 5.2 In-Order Delivery

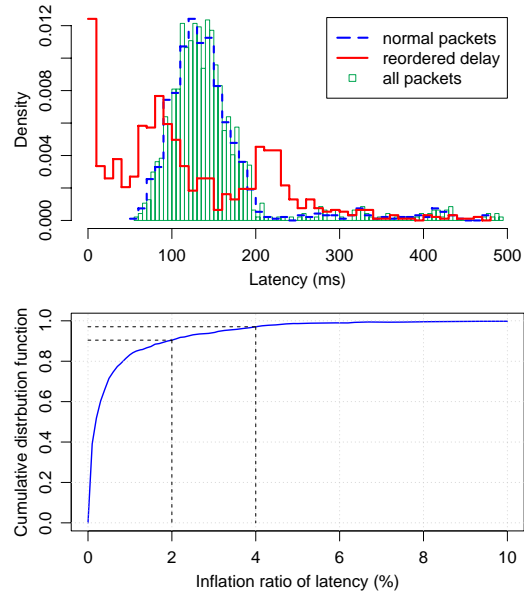
For many applications, such as file transfers, netnews, and the World Wide Web, in-order delivery is a *must-have*, since it ensures that the receiver application can process the data stream in the exact order generated by the sender. To guarantee in-order delivery, at the destination host, a packet cannot be processed until all of its preceding packets have been received and processed. However, the mechanism incurs unnecessary overhead if the data packets do not need to be processed in order. For example, suppose a game server sends out a series of 10 packets with position updates, but the first packet is lost so that only 9 packets are successfully delivered to the client. Until the client receives a retransmitted copy of the first packet, the other 9 packets must be buffered because they cannot be processed by the application. As a matter of fact, position updates in MMORPGs are often designed to be *accumulated*, so processing the 9 successfully delivered packets immediately upon receipt gives players a smoother gaming experience.

In practice, server packets primarily comprise accumulated state updates, dialogue messages, and responses to queries, such as information about certain equipment or the price of certain goods. Therefore, *server packets, except those carrying dialogue messages, can usually be safely processed in any order*. On the other hand, client packets primarily convey user actions, such as movement and fighting actions, which cannot be processed out of order. However, because most client commands relate to the most popular actions, and the same actions tend to occur in succession [6], a client packet is likely to repeat its preceding packet. In this case, the “repeated packets” can be processed out of order without affecting the correctness of the game. In view of the above, we remark that *in-order processing of all game packets is unnecessary*; consequentially, using TCP for MMORPGs can cause unnecessary latency.

To assess how much additional delay is induced by enforcing packet ordering, we assume an extreme case where game packets can be processed in any order. The two main sources of out-of-order delivery, *packet reordering in the network* and *packet loss*, are considered. As we captured the trace at the server side, we define “latency” as the time difference between the departure time of a server packet and the time the corresponding ack packet is received.

### 5.2.1 Packet reordering

We first observe the latency due to packet reordering in the network [4]. Fig. 4(a) depicts the distribution of the average latency of normal (ordered) packets, average additional delay due to reordered packets, and average latency



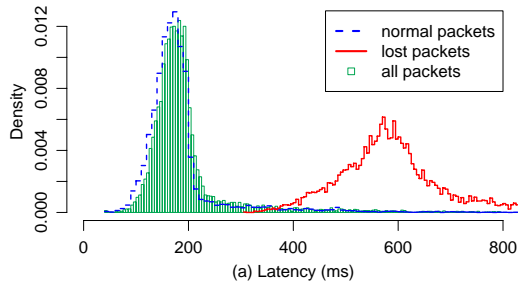
**Figure 4: The effect of packet reordering on network latency**

of all packets. On average, a packet was buffered for 140 ms if one or more of its preceding packets arrived late. Packet reordering only occurred in 2% of connections, and, within these connections, only 0.1% packets arrived at the destination later than their subsequent packets. Therefore, the overall latency did not increase significantly. In Fig. 4(b), we draw the expansion ratio of latency due to network reordering. In most connections, the reordering causes less than 4% additional delay to the average latency, which is acceptable.

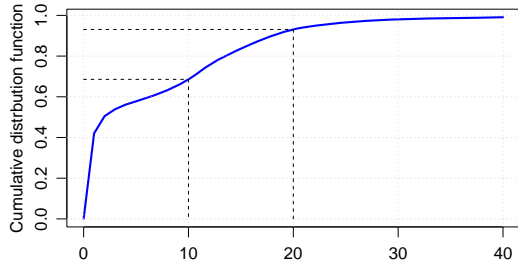
### 5.2.2 Packet loss

Compared with packet reordering, packet loss occurred more frequently and incurred longer delays. Fig. 5(a) shows the average latency of normal (non-lost) packets, lost packets, and all packets. Because it takes a great deal of time for the source host to detect and retransmit a dropped packet, the average latency of packets that have been lost (576 ms) is much longer than that of normal packets (186 ms). To illustrate the impact of packet loss on the overall latency, we compute the expansion ratio of latency due to packet loss for connections that experienced at least one packet loss. As shown in Fig. 5(b), 33% of connections incurred more than 10% additional latency, and 7% incurred more than 20% additional latency, which is much higher than that caused by packet reordering. The overall average latency due to packet loss increases from 186 ms to 199 ms, which we consider moderate and not detrimental.

However, delay jitters, i.e., *the standard deviation of latencies*, are more sensitive to the excessive latency of retransmitted packets. In Fig. 6(a), we show that delay jitters of lost packets spread more widely than those of normal packets. The average delay jitter of lost packets is 321 ms, and that of normal packets is 77 ms, which yields an expansion factor of four. As shown in Fig. 6(b), the distribution of the expansion ratio of delay jitters indicates that 22% of



(a) Latency (ms)



(b) Expansion ratio of latency (%)

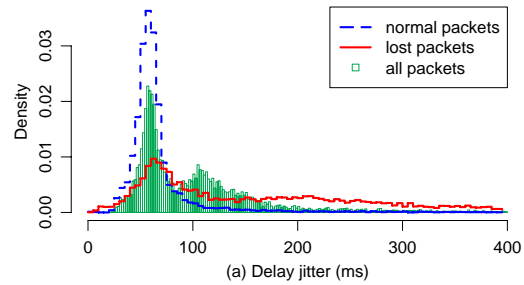
**Figure 5: The effect of packet loss on network latency**

connections incurred more than 100% additional jitters, and 6% incurred more than 200%. Overall the average delay jitter increased from 77 ms to 123 ms due to packet loss, an increase of 60%. According to a recent study [7], high delay jitters are *less tolerable* than high latencies, since it is easier for players to adapt to a slow game speed than to a continuously fluctuating game pace. Consequently, the increase in delay jitters due to the joint effect of the in-order delivery policy and packet loss could be one of the major causes which degrade the overall gaming experience.

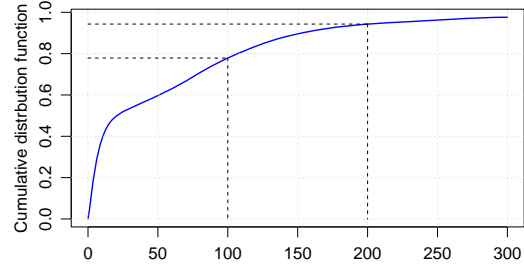
### 5.3 Congestion Control

In order to effectively use the network capacity and maintain fairness between different flows, TCP employs a feedback-controlled and window-based traffic regulation scheme. The approach uses a congestion window to limit the maximum number of packets that can be released within a round-trip time. In short, TCP’s congestion control are designed to adjust the size of the congestion window so that it faithfully reflects the bandwidth available for legitimate use by the current flow in terms of efficiency and fairness.

The problem is that the design of TCP congestion control is based primarily on the assumption that data transmission is *network-limited*, i.e., the sender always has data to send before the connection terminates. In contrast, data generation is *application-limited* in network games, i.e., the application often has a smaller amount of data to send compared to the capacity it could use. The congestion window (cwnd) evolves based on an additive increase and multiplicative decrease policy [3]. For bulk transfers, the cwnd inflates on receipt of an ack packet that acknowledges new data, and shrinks whenever TCP detects a packet loss, which is seen as a signal that the window size is beyond the legitimate share of bandwidth. As a result, the cwnd always oscillates between a minimum size and a value corresponding to the available bandwidth. However, since network games are



(a) Delay jitter (ms)



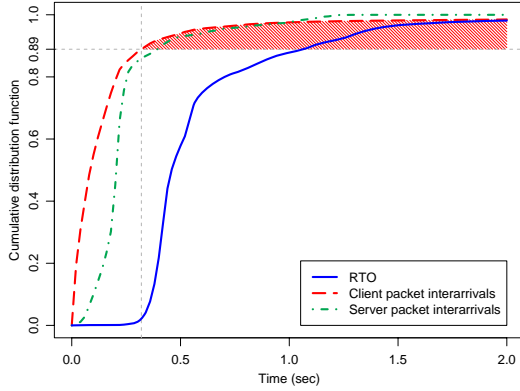
(b) Expansion ratio of delay jitter (%)

**Figure 6: The effect of packet loss on delay jitters**

application-limited, and their traffic load is usually much smaller than the network capacity, packet loss may never happen. In this case, the cwnd *increases indefinitely and cannot faithfully reflect the condition of the network path*. Actually, in our traces, the maximum and average window sizes in the server to client direction are 1.7 Mbps and 372 Kbps assuming MTU of 1500 bytes, which are unreasonably large to reflect the true bandwidth available to individual flows. Furthermore, 36% of connections experienced no packet loss, which could never happen in network-limited applications with sufficient data. The inappropriately large window size can be harmful if an application does not remain application-limited at all times, i.e, it may occasionally generate a large burst of packets in a small period. In this case, the large window would allow excess traffic to be sent, which is beyond the network capacity and leads to unnecessary congestion.

Although the congestion window tends to increase indefinitely in MMORPGs, occasionally *the window inappropriately restricts the departure of game packets*. According to the policy of *restart after idle periods* [3], the cwnd should reduce to two packets before beginning transmission, if the sender host has not sent data in an interval exceeding the retransmission timeout (RTO). This policy prevents potentially inappropriate bursts of traffic being transmitted due to an out-of-date cwnd that does not correctly reflect current network conditions. However, as we have seen, the generation of game packets is slow so that intervals between successive packets can be longer than retransmission timeout. In this case, TCP assumes that the present connection is idle and *unnecessarily resets the congestion window*.

Fig. 7 depicts the distributions of RTO, client packet interarrival times, and server packet interarrival times. As can be seen, RTO is generally longer than packet interarrival times in both directions. The upper portion of the client packet interarrival times is shaded to indicate that they are likely to be longer than RTO, which causes resets of the congestion



**Figure 7: The comparison of retransmission timeouts (RTO) and packet interarrival times**

window. Empirically, in our traces, 12% of client packets and 18% of server packets faced a cwnd reset before they were released, which explains why the congestion window did not increase unboundedly. The restart-after-idle-periods policy can induce additional transmission latency. For example, a player might be accustomed to issuing a series of three commands, “sneak,” “move,” and “attack” following a short period of thinking that is longer than RTO. In this case, he will be *penalized* because the third command cannot be released until the acknowledgement for the initial command has been received. This explains that resetting the congestion window after idle periods may degrade the interactivity and responsiveness of network games.

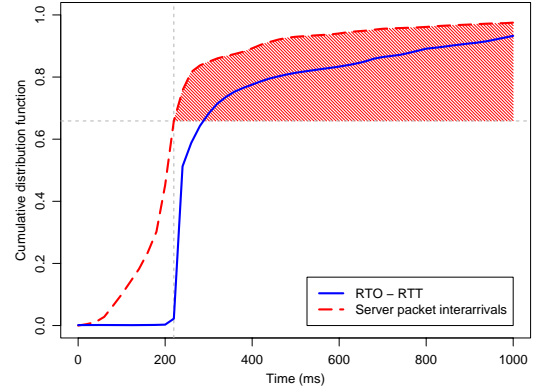
## 5.4 Loss Recovery

TCP relies on three strategies [3] (depending on the TCP version used) to infer whether a packet has been received by the destination host: 1) retransmission timeout, i.e., a packet is deemed as dropped if it has not been acknowledged in a time interval of RTO; 2) the fast-transmit mechanism, i.e., a packet is deemed as dropped if four successive and identical ack packets arrive without the arrival of any other intervening packets, and the four packets do not acknowledge all outgoing packets; and 3) the selective acknowledgment (SACK) mechanism [9], where the receiver can inform the sender about all segments that have arrived successfully, so the sender only needs to retransmit the segments that have actually been lost. As the game servers we traced did not enable the SACK option, we were able to analyze the performance of the former two strategies. We show that though the fast-retransmit algorithm is designed to alleviate long delays due to loss detection via retransmission timeout, *it is ineffective in the case of MMORPGs*.

We begin with some statistics of how TCP detected dropped packets in our traces. For server packets, only 0.08% of dropped packets were detected by the fast-retransmit mechanism; in other words, the game servers did not detect 99.92% of the packet loss until retransmission timeout occurred. As this result is counter-intuitive, even unbelievable, we illustrate how it can happen in detail. Our analysis indicates that fast retransmit failed for two reasons: 1) insufficient duplicate acks were received; 2) the counting of duplicate acks was interrupted by non-duplicate acks. As

**Table 3: The reasons why fast retransmit failed to trigger**

| Cause                                       | Ratio  |
|---|--------|
| Insufficient duplicate acks                 | 50.96% |
| Duplicate ack accumulation were interrupted | 49.04% |
| New data                                    | 48.90% |
| New ack                                     | 0.02%  |
| Window size change                          | 0.12%  |



**Figure 8: Server packet interarrival times are comparable to average RTO - average RTT.**

listed in Table 3, both cases occurred with approximately the same frequency. With regard to the first cause, the figure indicates that in approximately half the cases insufficient duplicate acks were received. This is because *the server packet rate was too low to elicit sufficient duplicate acks*. The design of fast retransmit assumes packets are sent in bursts so that when a packet is dropped, each of its successive packets will elicit a duplicate ack. In other words, to trigger the fast retransmit before a retransmission timeout, there should be at least three additional packets released within an interval of  $(RTO - RTT)$  following the departure of the dropped packet. However, because the server packet rate is too low, it is very unlikely that four packets could be released within that short period. In Fig. 8, we contrast the intervals  $(RTO - RTT)$  with the server packet interarrival times. We observe that both measures are comparable, where 34% of server packet interarrival times (shaded area) are likely to be longer than their corresponding  $(RTO - RTT)$ , i.e., no subsequent packets were released within that interval. Since insufficient server packets were released in succession following the dropped packet, insufficient duplicate acks were generated; consequently, fast retransmit was not triggered.

The second reason fast retransmit failed to trigger is that *game traffic is bi-directional*. While not clearly defined in [3], according to [14] and the implementation of 4.4BSD, the definition of duplicate acks is strict in that each ack packet must not contain *data*, must have the same receiver window size, and must have the same acknowledged sequence number. The definition implies that if the opposite party transmits a data packet to the host, which is waiting for more duplicate acks, then the counting of duplicate acks will be reset to zero. In other words, the fast retransmit may not

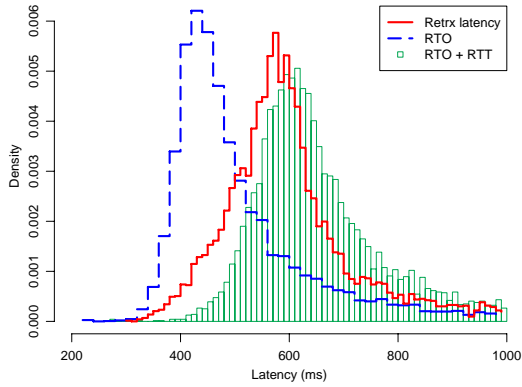


Figure 9: Average latency of dropped packets

be triggered, even if sufficient duplicate acks are generated. According to Table 3, the counting of duplicate acks was interrupted in approximately half the cases before sufficient duplicate acks were accumulated. In particular, most duplicate ack interruptions were caused by data segments sent from the game clients, i.e., they were apt to release more packets before sufficient duplicate acks were generated. As shown in Fig. 3, the client packet interarrival times can be approximately described by an exponential distribution with an average rate of 8 pkt/sec. Now, suppose that the accumulation of sufficient duplicate acks takes approximately one round-trip time. We can then compute that the probability of a game client generating at least one additional packet within a round-trip time is  $\Pr(X < 0.185) \approx 0.77$ , where  $X$  is an exponential random variable with mean =  $1/8$ , and 185 ms is the overall average RTT taken from traces. Because of the low server packet rate and the relatively high client packet rate, client packets usually arrived before sufficient duplicate acks could be accumulated; therefore, the fast retransmit failed to trigger in most cases.

In addition to the above argument, we provide more evidence to back up the observation that fast retransmit was not triggered properly. Theoretically, a dropped packet detected by a retransmission timeout would incur a transmission latency of  $(RTO + RTT)$ . As shown in Fig. 9, the distribution of the latency of retransmitted packets and that of  $(RTO + RTT)$  are approximately equivalent. This again evidences that most dropped server packets were detected and recovered by retransmission timeouts.

We remark that while fast retransmit is shown to be ineffective, SACK is immune to traffic bi-directionality, and only one ack packet elicited by a further packet is sufficient to make the source host aware of the dropped packet. Thus, we highlight the importance of the SACK algorithm in network games, and recommend that every network game employing TCP should guarantee the SACK option is enabled at both ends.

## 6. DISCUSSION

In this section, we discuss the generality of our analysis and its implications, and present how performance degradation due to TCP affects users' gaming experience. Finally we propose a number of guidelines in designing efficient transport protocols for online games.

### 6.1 Generality of Analysis

Although our analysis is confined to traces from a single game, we argue the generality of our findings from two aspects: traffic characteristics and client distributions. The game traffic characteristics which lead to TCP performance problems are mainly 1) tiny packets, 2) low packet rate, 3) application-limited traffic generation, and 4) bi-directional traffic; however, these properties are essentially shared by all real-time interactive network games, rather than specific to this particular game [8]. With respect to client distributions, while the traced game servers are located in Taiwan, players were spread over 13 countries and hundreds of autonomous systems. As a matter of fact, the average RTTs range from 95 ms to 580 ms, and the loss rates range from no loss to 20% (computed by one percentile and 99 percentile, respectively). The heterogeneity of network path characteristics manifests that our analysis result is not particular to a specific environment but rather generalizable.

### 6.2 Performance Impact on User Experience

Having shown that TCP cause degradation in terms of network performance metrics, such as packet delay and delay jitters, we now discuss how did the degraded performance impact users' willingness to continue a game. In [7], with an in-depth analysis of the relationship between session time and network QoS factors, the authors established that *all of network delay, delay jitters, and packet loss significantly affect game playing time*. Based on the survival model developed in that paper, we can quantify the impact of degraded performance on user departure rate by

$$\text{hazard function} \propto \exp(19.2 \times \text{rtt.min} + 4.54 \times \text{rtt.sd} + 0.7 \times \log(\text{closs}) + 0.45 \times \log(\text{sloss})).$$

The hazard function gives the *instantaneous rate* at which users leave the game for sessions that have lasted for time  $t$ , where  $\text{rtt.min}$ ,  $\text{rtt.sd}$ ,  $\text{closs}$ , and  $\text{sloss}$  stand for minimum RTT, standard deviation of RTT, client packet loss rate, and server packet loss rate, respectively.

To our traces, the delay jitters were raised from an average of 77 ms to 124 ms due to the joint effect of in-order delivery policy and packet loss. The increase in hazard rate due to the increase in delay jitters can be computed by  $\exp((0.124 - 0.077) \times 4.54) \approx 1.24$ , where 4.54 is the coefficient of factor  $\text{rtt.sd}$ . That is, the increase of delay jitters has raised the probability that a player will leave the game in every instant by 24%. In other words, assuming an ideal case that the additional delay jitters induced by TCP could be avoided, the user departure rate is expected to decrease by 20% ( $1/1.24 \approx 0.8$ ). Concretely speaking, this corresponds to an increase of the median game playing time from 100 minutes to 135 minutes in our case. By the above computation, we can see that there is much room for improvement by making loss recovery more efficient and only ordering packets whenever necessary.

### 6.3 Protocol Design Guidelines

Based on the understanding of game traffic and its interaction with TCP, we propose the following guidelines for the design of game transport protocols:

- **Supporting both reliable and unreliable delivery:** Not every game packet needs to be reliably transmitted. For example, the state updates which contain gestures

or motions of a character *faraway from the notified character* need not be reliable as the acting character is unapparent on screen. The *level of detail* [13] could be mapped to a maximum tolerable loss rate of state updates, i.e., the closer the object is, the more exact the object's state should be.

- **Supporting both in-order and out-of-order delivery:** We have shown that the in-order delivery policy leads to higher latencies and delay jitters due to unavoidable packet reordering and loss. This overhead can be reduced by *only ordering packets when it is absolutely necessary*. For example, repeated attack commands on an opponent, which are frequent in game play, could be processed out of order, as they are semantically and visually equivalent.
- **Accumulative delivery:** Many types of game messages are *accumulative* in nature, i.e., subsequent information will override the earlier ones. For example, state updates, especially position updates, are usually accumulated so that a missing one would not matter, unless it is the last in a series of updates. Thus, a series of accumulated commands, except for the last command, could be delivered in an unreliable and out-of-order way, which will certainly reduce the network load and application latency.
- **Multiple Streams:** Game messages should be designed *as independent as possible*, so that a delayed message will not affect the subsequent ones. One solution is putting unrelated messages into parallel streams, and only ensuring the packet ordering in each of which. For example, chat messages are apparently independent of game play commands and should be put into a separate stream.
- **Coordinated congestion control:** Given that numerous flows, e.g., more than tens of thousands, are frequently seen on MMOG servers, it would be difficult for these flows to achieve an efficient bandwidth sharing in terms of fairness and responsiveness. We consider it is especially important for game flows to do *congestion control in a coordinated, rather than individual, manner*. For example, avoid dispatching game messages synchronously could alleviate the traffic burstiness and decrease the probability of packet loss [6].

Given the unique demand in game packet delivery, none of standardized protocols are readily fit to our needs. In practice, the desired protocol could be designed as a *hybrid*. For example, using UDP, RTP, or DCCP as a basis for unreliable transmission, and using SCTP and RDP as a basis for reliable transmission. We remain the detailed design and evaluation of transport protocols for online games as part of future work.

## 7. CONCLUSION

In this paper, we have analyzed the performance of TCP in of *ShenZhou Online*, a commercial, mid-sized MMORPG. Our study indicates that, though TCP is full-fledged and robust, simply transmitting game data over TCP could cause unexpected performance problems. This is due to the following distinctive characteristics of game traffic: 1) tiny packets, 2) low packet rate, 3) application-limited traffic generation, and 4) bi-directional traffic.

We have shown that because TCP was originally designed for unidirectional and network-limited bulk data transfers,

it cannot adapt well to MMORPG traffic. In particular, the window-based congestion control mechanism and the fast retransmit algorithm for loss recovery are ineffective. This suggests that the selective acknowledgement option should be enabled whenever TCP is used, as it significantly enhances the loss recovery process. Furthermore, TCP is overkill, as not every game packet needs to be transmitted reliably and processed in an orderly manner. We have also shown that the degraded network performance did impact users' willingness to continue a game. Finally, a number of design guidelines have been proposed by exploiting the unique characteristics of game traffic.

## Acknowledgments

This work would not have been possible without the extensive traffic trace of *ShenZhou Online*. The authors are much indebted to the following people who helped us gather the trace: Tsing-San Cheng, Lawrence Ho, Chen-Hsi Li, and especially to Yen-Shuo Su, who between them made the datasets available. The authors also wish to thank the anonymous referees for their constructive criticisms.

## 8. REFERENCES

- [1] ShenZhou Online. <http://www.ewsoft.com.tw/>.
- [2] FAQ - Multiplayer and Network Programming. GameDev.Net, 2004.
- [3] M. Allman, V. Paxson, and W. Stevens. TCP Congestion Control. RFC 2581, Apr. 1999.
- [4] J. C. R. Bennett, C. Partridge, and N. Shectman. Packet reordering is not pathological network behavior. *IEEE/ACM Trans. Netw.*, 7(6):789–798, 1999.
- [5] R. Braden. Requirements for Internet Hosts - Communication Layers. RFC 1122, Oct. 1989.
- [6] K.-T. Chen, P. Huang, and C.-L. Lei. Game traffic analysis: An MMORPG perspective. *Computer Networks*, 51(3), 2007. Article In Press.
- [7] K.-T. Chen, P. Huang, G.-S. Wang, C.-Y. Huang, and C.-L. Lei. On the sensitivity of online game playing time to network QoS. In *Proceedings of IEEE INFOCOM'06*, Barcelona, Spain, Apr. 2006.
- [8] W. C. Feng, F. Chang, W. C. Feng, and J. Walpole. A traffic characterization of popular on-line games. *IEEE/ACM Transactions on Networking*, 13(3):488–500, June 2005.
- [9] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP Selective Acknowledgement Options. RFC 2018, Oct. 1996.
- [10] M. Mauve, V. Hilt, C. Kuhmünch, and W. Effelsberg. RTP/I - toward a common application level protocol for distributed interactive media. *IEEE Transactions on Multimedia*, 2001.
- [11] S. Pack, E. Hong, Y. Choi, Ilkyu Park, J.-S. Kim, and D. Ko. Game transport protocol: lightweight reliable transport protocol for massive interactive on-line game. In *Proceedings of the SPIE*, volume 4861, pages 83–94, 2002.
- [12] P. Quax, P. Monsieus, W. Lamotte, D. D. Vleeschauwer, and N. Degrande. Objective and subjective evaluation of the influence of small amounts of delay and jitter on a recent first person shooter game. In *Proceedings of ACM SIGCOMM 2004 workshops on NetGames '04*, pages 152–156. ACM Press, 2004.
- [13] S. Singhal and M. Zyda. *Networked Virtual Environments: Design and Implementation*. ACM Press, Siggraph Series, New York, 1999.
- [14] R. W. Stevens. *TCP/IP Illustrated, Volume 2: The Implementation*. Addison-Wesley, 1995.
- [15] B. S. Woodcock. An analysis of MMOG subscription growth – version 18.0.