# On the Challenge and Design of Transport Protocols for MMORPGs

Chen-Chi Wu[†], Kuan-Ta Chen[‡], Chih-Ming Chen[†], Polly Huang[†], and Chin-Laung Lei[†]

[†]Department of Electrical Engineering, National Taiwan University
[‡]Institute of Information Science, Academia Sinica

**Abstract**

Although MMORPGs are becoming increasingly popular as well as a highly profitable Internet business, there is still a fundamental design question: *Which transport protocol should be used—TCP, UDP, or some other protocol?* In this paper, we first evaluate whether TCP is suitable for MMORPGs, and then propose some novel transport strategies for this genre of games. Our analysis of a trace collected from a TCP-based MMORPG called *ShenZhou Online* indicates that *TCP is unwieldy and inappropriate for MMORPGs*. We find that the degraded network performance problems are due to the following characteristics of MMORPG traffic: 1) tiny packets, 2) a low packet rate, 3) application-limited traffic generation, and 4) bi-directional traffic.

Since not all game packets require reliable transmission or in-order delivery, transmitting all packets with a strict delivery guarantee causes high delays and delay jitters. Therefore, our proposed transport strategies assign game packets with appropriate levels of transmission guarantee depending on the requirements of the packets' contents. To compare the performance of our approach with that of existing transport protocols, we conduct network simulations with a real-life game trace from *Angel's Love*. The results demonstrate that our strategies significantly reduce the end-to-end delay and delay jitter of packet delivery. Finally, we show that our strategies effectively raise satisfaction levels of the game players.

## I. Introduction

Massive Multiplayer Online Role Playing Games (MMORPGs) have become extremely popular in recent years, and now attract millions of players who participate in the activities with other gamers in a virtual world. The number of active subscribers worldwide increased from 12 million to 16 million between January 2007 and January 2008 [22]. However, degraded network performance, such as lengthy delays and excessive delay jitters affect players' willingness to continue in a game [6]. How to ensure that players have a satisfactory gaming experience is currently the most critical issue facing the game industry [8, 9].

A fundamental design question in the development of MMORPGs is—*Which transport protocol should be used—TCP, UDP, or some other protocol?* GameDev.Net offers the following advice [2]: "*For RTS (Realtime Strategy) games, you're usually best off using TCP, although UDP can make sense in extreme cases. For action-based games like first-person shooters or racers, you should absolutely use UDP. For role playing games, the story is less clear—action-based RPGs with lots of kinetics, like City of Heroes, use UDP, whereas slower RPGs and MUDs often stay with TCP.*" As shown in Table I, both TCP and UDP are widely used by popular MMORPGs, while other games may adopt a hybrid approach based on both protocols. Several other transport protocols can be candidates for MMORPGs, for example, RTP, the datagram congestion control protocol (DCCP) [15], the stream control transmission protocol (SCTP) [20], the game transport protocol (GTP) [17], and other middleware protocols [1, 12]. Because of the diversity of protocols in use and several newly designed protocols, there is no consensus on the best transport protocol for this genre of games. In this paper, we first assess whether TCP is suitable for MMORPGs, and then propose transport strategies that assign appropriate levels of transmission guarantee to each game packet to opportunistically improve the packet delivery performance.

Based on a real-life trace collected from a TCP-based MMORPG called *ShenZhou Online* [21], we investigate the performance problem caused by the interplay of TCP and the design of MMORPGs. The following findings show that *TCP is unwieldy and inappropriate for MMORPGs*.

1) Since *game packets are generally small*, TCP/IP headers occupy up to $46\%$ of the total bandwidth consumed by the game traffic.

Corresponding author: Kuan-Ta Chen, contact him at ktchen@iis.sinica.edu.tw.

TABLE I
THE TRANSPORT PROTOCOLS USED BY POPULAR MMORPGS

| Protocol | MMORPGs |
|---|---|
| TCP | World of Warcraft, Angel's Love, Lineage I/II, Guild Wars, Ragnarok Online |
| UDP | EverQuest, City of Heroes, Asheron's Call, Ultima Online, Final Fantasy XI |
| TCP/UDP | Dark Age of Camelot |

2) As some game packets can be processed out-of-order, forcing in-order delivery for all packets causes unnecessary transmission delays. In our evaluation, $7\%$ of connections experience more than $20\%$ additional average delay, and $6\%$ experience at least $200\%$ additional delay jitter.
3) The congestion control mechanism is ineffectual because game traffic is *application-limited*, rather than network-limited. In our trace, $12\%$ of client packets and $18\%$ of server packets encounter a congestion window reset, which prevents the packets from being transmitted immediately and induces additional latencies.
4) The fast-retransmit algorithm is ineffective because game traffic is *bi-directional* and packets are generated at a *slow rate*. We found that more than $99\%$ of dropped packets sent from servers were not detected until retransmission timeout occurred. This causes the average latency of lost packets to be much longer than that of normal (non-lost) packets.

UDP is another widely used transport protocol, but it cannot be applied to MMORPGs directly due to the lack of reliable and in-order transmission. To deal with packet loss and the reordering of game messages that need strict reliability and in-order processing, an excessive amount of effort is required if UDP is adopted.

In practice, the most appropriate protocol for MMORPGs is a *hybrid* that provides different levels of transmission guarantee based on the requirements of game packets, so they can be transmitted efficiently, i.e., the delays and delay jitters are as low as possible. For example, chat messages should be transmitted reliably and in an orderly manner, while the loss of certain position updates for avatars is tolerable. To meet these requirements, *we propose content-based transport strategies that provide each type of packet content with the appropriate level of delivery guarantee*. Based on a trace collected from *Angel's Love*, we conducted network simulations to compare our strategies with some existing transport protocols in terms of network performance. The results show that our strategies incur much lower end-to-end delay and end-to-end delay jitter than most of the protocols we evaluated. Finally, based on a model that describes the relationship between network quality and users' playing time [6, 7], we designed the Game Satisfaction Index (GSI) to quantify the gaming experience of players under different network conditions. We show that the level of player satisfaction under our transport strategy is $3.57$ times better than that of TCP.

Our contribution in this work is three-fold:
1) Based on a realistic game trace, we analyze the performance of TCP and demonstrate that it is unsuitable for MMORPGs.
2) We propose content-based transport strategies for MMORPGs, and evaluate their performance improvement over existing protocols.
3) We present the Game Satisfaction Index, which explicitly indicates that the proposed transport strategies significantly improve player satisfaction in MMORPGs.

The remainder of this paper is organized as follows. Section II contains a review of related works. In Section III, we analyze the performance problems caused by TCP. Section IV details the proposed transport strategies. We evaluate the performance of existing and proposed transport protocols using trace-driven network simulations with a real-life game trace in Section V. Then, in Section VI, we summarize our conclusions.

## II. RELATED WORK

With the growing popularity of MMORPGs, a number of works focus on evaluating the performance of transport protocols in this genre of games. For example, in [14], Harcsik et al. evaluated the packet delivery latency of TCP, SCTP, and two middleware protocols based on UDP. Because of the small packet sizes and low packet rates in MMORPGs, game applications never consume all of the available bandwidth. Therefore, traditional transport mechanisms, such as TCP, are not suitable for MMORPGs because they assume that packet generation is network-limited, rather than application-limited. Harcsik et al.'s evaluation results demonstrate that UDP-based middleware protocols perform better than TCP and SCTP.

Other studies have designed transport protocols for MMORPGs. In [13], the authors proposed two approaches to make TCP more suitable for MMORPGs. The first sets a more aggressive retransmission timeout, and the second

TABLE II
SUMMARY OF GAME TRAFFIC TRACES

| Trace | Date | Period | Drops[†] | Session | Pkt. (in / out) | Bytes (in / out) |
|---|---|---|---|---|---|---|
| $\mathcal{N}_1$ | 8/29/04 (Sun.) | 8 hr. | 0.003% | 7,597 | 342M / 353M | 4.7TB / 27.3TB |
| $\mathcal{N}_2$ | 8/30/04 (Mon.) | 12 hr. | ?[‡] | 7,543 | 325M / 336M | 4.7TB / 21.7TB |

[†] This column gives the kernel drop count reported by *tcpdump*.

[‡] The reported kernel drop count was zero, but we found that some packets were actually dropped at the monitor.

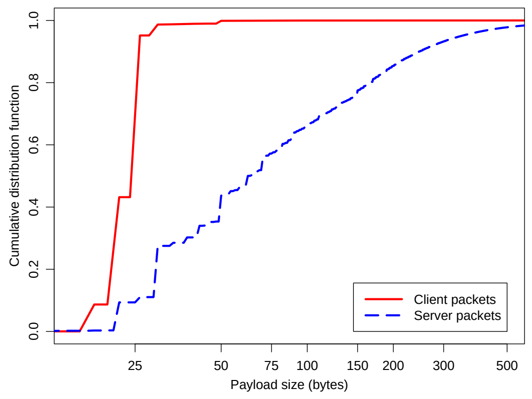

Fig. 1. Payload size distribution

deploys a proxy to multiplex streams sent to clients into a single connection. Both approaches reduce packet delay in games. However, the type of packet is not considered thus there must be extra overhead for certain packets as the QoS levels required by game packets are not equivalent. In another work [17], Pack et al. proposed the Game Transport Protocol (GTP), which is designed to meet the various requirements of MMORPGs. GTP uses a packet-oriented, rather than a byte-oriented, window scheme to accommodate the small packet size of game traffic. In addition, to satisfy the real-time constraints of packet delivery, GTP supports an adaptive retransmission scheme, which controls the maximum number of retransmissions based on each packet's priority. Although the work incorporates the concept of multiple QoS levels, it does neither address, in practice, how to map a variety types of game packets with different QoS levels, and nor provide a performance evaluation of the proposed design.

In [18], Shirmohammadi et al. proposed to transmit update messages according to multiple QoS levels in collaborative virtual environments (CVE). In CVEs, some messages require reliable transmission because their loss will incur collaboration failures. For example, the last state of a shared object in a CVE must be delivered reliably so that all participants can perceive the last position of the object correctly. On the other hand, some update messages can be sent by a best effort delivery as they can be lost without impacting the interaction between participants. For example, the loss of most move updates is tolerable if the last one has been received correctly. In view of different delivery requirements, Shirmohammadi et al. classify the messages into two types: key update messages and regular update messages. The authors implement their proposed approach based on SCTP and conduct subjective experiments to evaluate the performance of the proposal. The results show that their approach achieves reliable and timely transmission for key messages even in a congested and lossy network environment.

## III. Analysis of TCP Performance

TCP is a connection-oriented transport protocol that provides reliable transmission, in-order delivery, congestion control, and flow control. In this section, we analyze the performance of TCP in MMORPGs based on real-life game traces. We find that in-order delivery causes unnecessary transmission latencies and high delay jitters, and the protocol's congestion control and loss recovery mechanism are ineffective in MMORPGs.

### A. Trace Description

First we describe the traces used for evaluating the performance of TCP in MMORPGs. *ShenZhou Online* is a TCP-based MMORPG that is popular in Taiwan [21], where there are thousands of players online at any one time. In this game, players can engage in fights with other players or random creatures, train their avatars to acquire

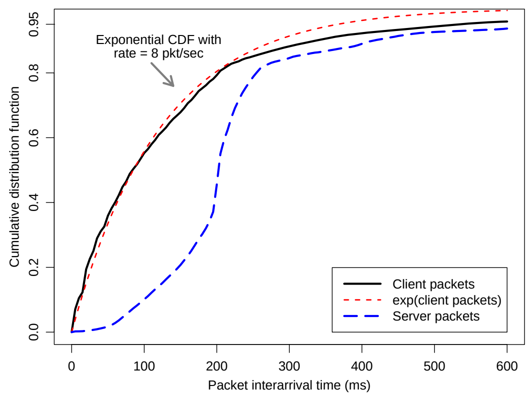

Fig. 2. Packet interarrival time distribution

special skills, participate in commercial activities, or take on a quest. With the assistance of *ShenZhou Online* staff, we set up a traffic monitor to capture packets sent from and received by the game servers. The collected traces are summarized in Table II. Although the game servers were located in Taiwan, we found that the players in the traces were spread over 13 countries and hundreds of autonomous systems. The wide distribution of clients manifests the heterogeneity of the network paths' characteristics and the generality of the traces. Because of space limitations, we refer interested readers to [5] for details of the game description and measurement setup.

Next, we investigate the characteristics of the collected traces. For brevity, we denote packets sent by game clients as *client packets*, and all traffic sent by clients as *client traffic*. The terms *server packets* and *server traffic* are defined similarly. Fig. 1 shows the cumulative distribution function (CDF) of the payload size. Clearly, client packets and server packets are drastically different in terms of the payload size. The reason is that a client packet only contains *one* player's commands, whereas a server packet for a player may contain the actions and states of nearby avatars, and even system notifications. Client packets are relatively *small*. In the traces, $98\%$ of the packets have a payload size smaller than $32$ bytes. Specifically, the payload sizes of 23 and 27 bytes comprise $36\%$ and $52\%$ of the packets respectively. This distribution evidences that user actions are *dominated by a few popular commands*, such as attack and walk, so that a large proportion of client packets have certain payload sizes. In contrast, server packets have a wider distribution and the average payload size is $114$ bytes.

Figure 2 shows the distribution of packet interarrival times. For client traffic, most packet interarrival 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. According to [5], the deviation of interarrival times is caused by the *diversity of players' behavior*, which is a specific feature of adventure-oriented games like MMORPGs. On the other hand, approximately $50\%$ of the interarrival times for server packets are around $200$ ms, which indicates that servers broadcast information to clients periodically.

## B. Protocol Overhead

TCP provides reliable data transmission through a positive acknowledgement policy. When a host receives a data packet, it must generate an acknowledgment (ack) packet as feedback for the sender. Consequently, a large number of ack packets are generated and transmitted over the network. In [4], the author proposed a delayed ack mechanism that allows the host to only send an ack for alternate data packets received in a connection, unless the delayed ack timer (usually set at $200$ ms) has expired. In bulk data transfer scenarios, where packets are sent in bursts, the majority of successive packets arrive at the destination before the delayed ack timer expires. Therefore, the number of ack packets should be reduced to half the number of data packets. However, because of the low packet rate in MMORPGs, the subsequent game packet may not arrive in $200$ ms from the time the preceding packet was received.

In our traces, the number of client ack packets is much more than half the number of server data packets. This is because approximately $40\%$ of the interarrival times for server packets are longer than $200$ ms (as shown in Fig. 2). Furthermore, $38\%$ of the packets in the traces are pure ack packets. Because of the relatively large proportion of

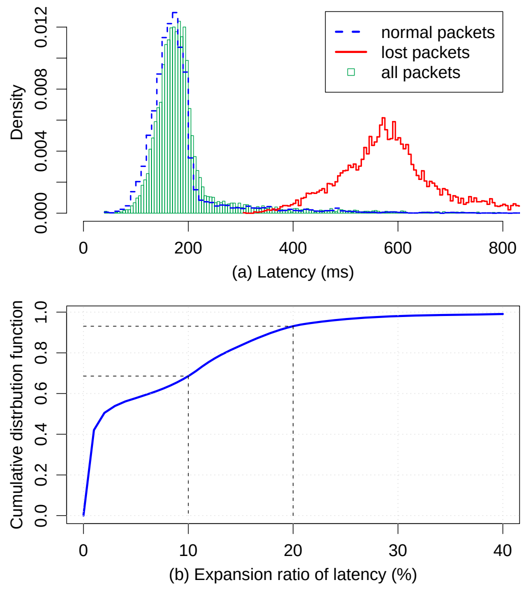

Fig. 3. The impact of packet loss on network latency

pure ack packets and the small payload size of game packets, the network bandwidth occupied by the TCP/IP header constitutes an excessive overhead. For example, in the traces, TCP/IP headers occupy $46\%$ of the bandwidth consumed by the game traffic.

## *C. In-Order Delivery*

TCP has an in-order delivery policy that guarantees the receiver will process data packets in the order generated by the sender. Under this mechanism, a packet must be buffered at the receiver until all of its preceding packets have been received and processed. In other words, if a packet is lost in the network, all of its subsequent packets that have been received already would suffer additional delay. However, certain game packets do not require in-order processing, so the mechanism may cause unnecessary delay. For example, if a player attacks an opponent repeatedly with the same weapon and the same action, the game application will generate a number of duplicate packets with attack messages and transmit them to the server. In this case, the packets can be processed out of order at the server without affecting the outcome of the game. Moreover, position or state updates in MMORPGs are often designed to be *accumulative*; that is, subsequent updates override earlier ones. As a result, an update packet can be processed immediately on arrival, instead of being buffered until all of its preceding packets arrive at the destination host.

To evaluate how much additional delay is caused by strict in-order delivery, we consider an extreme case in which all game packets can be processed in an arbitrary order. We investigate the influence of packet loss, which is the main source of out-of-order delivery, in terms of transmission latency and delay jitters (i.e., the standard deviation of the latency). Since the traces were collected 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.

In Fig. 3(a), we show the average latency of normal (non-lost) packets, lost packets, and all packets. Before retransmitting a dropped packet, the sender needs extra time to detect a lost packet. Thus, the average latency of packets that have been lost ($576$ ms) is much longer than that of normal packets ($186$ ms). Fig. 3(b) shows the expansion ratio of latency due to packet loss for connections that experienced at least one packet loss. We observe that $33\%$ of them suffered more than $10\%$ additional latency, and $7\%$ suffered more than $20\%$. Overall, packet loss caused the average latency to increase from $186$ ms to $199$ ms.

Although packet loss does not raise the average latency excessively, delay jitter is seriously affected. As shown in Fig. 4(a), the delay jitter of lost packets is spread more widely than that of normal packets. The average delay jitter of lost packets ($321$ ms) is four times longer than that of normal packets ($77$ ms). Figure 4(b) shows the expansion ratio of delay jitter due to packet loss. We observe that $22\%$ of connections suffered more than $100\%$ additional jitter, and $6\%$ suffered more than $200\%$. The overall average delay jitter increased from $77$ ms to $123$

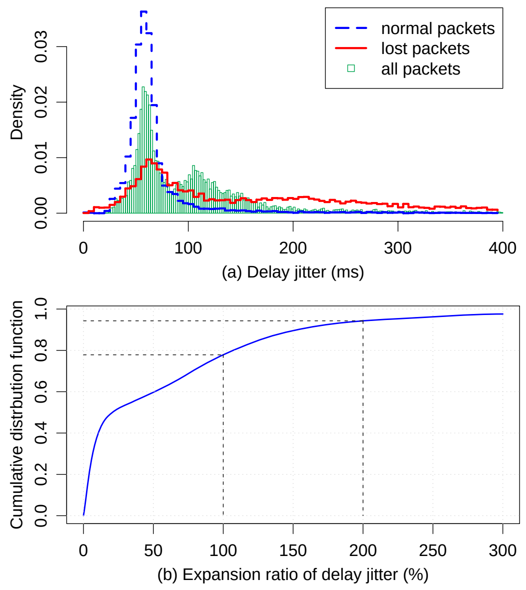

Fig. 4. The impact of packet loss on delay jitters

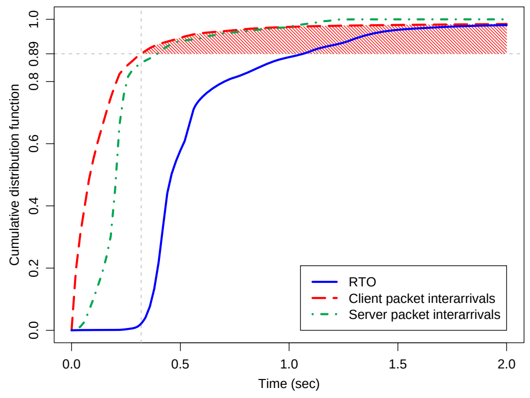

Fig. 5. The comparison of retransmission timeouts (RTO) and packet interarrival times

ms. According to [7], high delay jitter is *less tolerable* than high latency, as a continuously fluctuating game pace severely reduces the smoothness of the game play. Thus, the increase in delay jitter caused by the joint effect of the in-order delivery policy and packet loss could significantly degrade players' gaming experience.

## D. Congestion Control

To avoid congestion collapse in a network and use the network's capacity effectively, TCP uses a congestion window (cwnd) so that each sender can limit the sending rate based on its perceived network condition. Using the additive increase and multiplicative decrease (AIMD) policy [3], the size of the cwnd additively increases when the sender perceives that the end-to-end path is congestion-free, and multiplicatively decreases when it perceives congestion (e.g., packet loss) on the path. Therefore, for efficiency and to avoid congestion avoidance, the cwnd is adjusted to approximate the bandwidth available for legitimate use by a connection.

The AIMD policy is based on the assumption that data transmission is *network-limited*, i.e., the sender always has data to send in the connection. In the case of bulk transfers, the cwnd inflates when a packet is acknowledged, and shrinks when packet loss occurs. Hence, the cwnd oscillates between the minimum size and a value corresponding to the available bandwidth. However, in MMORPGs, data generation is *application-limited*, i.e., an application usually has a smaller amount of data to send compared with the available bandwidth it can use. Thus, packet loss is rare in this kind of game because the traffic load is usually much smaller than the network capacity. As a result, the cwnd *becomes arbitrarily large and cannot faithfully reflect the condition of the network path*. In our traces, the maximum and average window sizes in the server to client direction are 1.7 Mbps and 372 Kbps respectively.

TABLE III
THE REASONS WHY FAST RETRANSMIT FAILED TO TRIGGER

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

Both sizes are too large to reflect the available bandwidth of individual flows. In addition, we found that 36% of connections experienced no packet loss, which would never happen in network-limited applications with sufficient data. A potential problem associated with an inaccurately large window size is that an application with application-limited traffic may occasionally have a large burst of packets to send in a short time. This condition results in the release of an overwhelming traffic load due to the inappropriately large window size; therefore, the network bandwidth will be exhausted and congestion will occur unnecessarily.

Although the congestion window tends to become arbitrarily large in MMORPGs, it could be reset incorrectly because of the *restart after idle periods policy* [3]. In a connection, if the sender has not released packets in an interval longer than one retransmission timeout (RTO), the congestion window should be reset to two packets. The purpose is to prevent an inappropriate burst of packets being transmitted due to an out-of-date cwnd that does not faithfully reflect current network conditions. However, the packet rate in MMORPGs is very low, so the packet interarrival times may be longer than one RTO. In this case, the cwnd is reset unnecessarily before the next packet transmission, so the release of subsequent packets is restricted. Figure 5 shows the distributions of the RTO, client packet interarrival times, and server packet interarrival times in our traces. For the client packet interarrival times, the shaded area in the upper portion indicates that they are probably longer than the RTO. In the traces, 12% of client packets and 18% of server packets encountered a cwnd reset before they were released. The restart-after-idle-periods policy causes additional transmission delay. For example, suppose a player issues a series of three commands, "sneak," "move," and "attack," following a short period of thinking that is longer than RTO. Since the cwnd is reduced to two packets, the third command cannot be released until the first command is acknowledged. As a result, inappropriate resets of the congestion window have a negative effect on the interactivity and responsiveness of MMORPGs.

## *E. Loss Recovery*

To detect packet loss, TCP uses one of the following three strategies [3] (depending on the version of TCP): 1) retransmission timeout (RTO), i.e., a packet is deemed to have been dropped if the associated acknowledgment is not received by the sender before RTO occurs; 2) the fast-retransmit mechanism, i.e., a packet is considered lost if four successive and identical ack packets arrive without interruption by other packets; and 3) the selective acknowledgment (SACK) mechanism [16], which enables the receiver to inform the sender about the packets received; then, the sender only needs to retransmit the lost packets. Because the game servers we monitored did not enable the SACK option, we focus on the performance analysis of the first two strategies.

The fast-retransmit algorithm is designed to alleviate the long delays incurred by detecting packet loss via retransmission timeout. However, in our traces, only 0.08% of dropped server packets were detected by the mechanism; that is, 99.92% of the lost packets were not detected by the game servers until RTO occurred. This surprising result indicates that the *fast-retransmit algorithm is ineffective in MMORPGs*. According to our analysis, the mechanism failed for two reasons: 1) insufficient duplicate acks; and 2) the accumulation of duplicate acks was interrupted. As shown in Table III, the occurrence frequency is approximately the same in both cases. To trigger the fast retransmit mechanism, a sender should receive at least three duplicate acks within an $(\text{RTO} - \text{RTT})$ interval following a dropped packet. In other words, within that interval, the sender must release at least three additional packets, each of which will elicit a duplicate ack. However, because the server packet rate is too low, it seems unlikely that four packets could be released within that short period. To examine this conjecture, we compare the distribution of $(\text{RTO} - \text{RTT})$ intervals with that of the server packet interarrival times detailed in Fig. 6. The shaded area indicates that 34% of the server packet interarrival times were probably longer than their corresponding $(\text{RTO} - \text{RTT})$ intervals; that is, there were no subsequent packets within that interval. Hence, fast retransmit was not triggered because insufficient duplicate acks were generated.

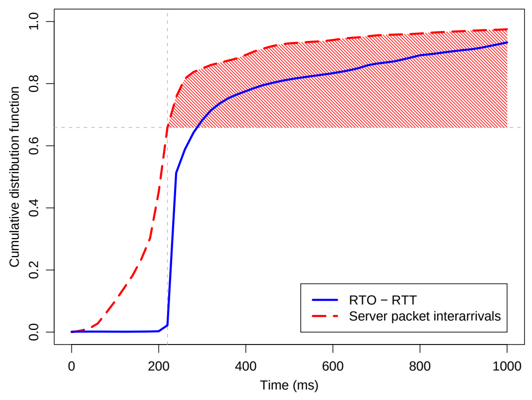

Fig. 6. Server packet interarrival times are comparable to average RTO - average RTT.

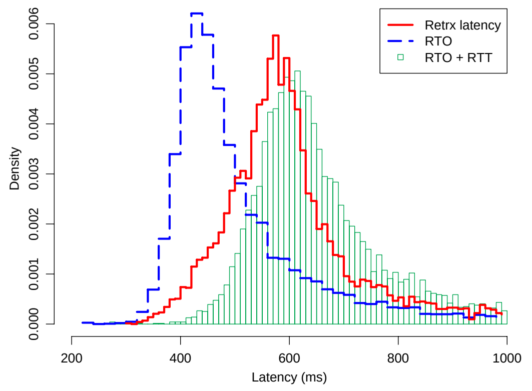

Fig. 7. Average latency of dropped packets

In the second case, the fast retransmit mechanism failed because the counting of duplicate acks was interrupted by other packets. According to [3, 19] and the implementation of 4.4BSD, the definition of a series 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 acknowledgement sequence number. By this definition, if a receiving host sends a data packet to a sender that is waiting for more duplicate acks in that connection, the count of duplicate acks will be reset to zero. Thus, the fast retransmit mechanism will not be triggered unless the counting of duplicate acks is not interrupted, even if sufficient duplicate acks are released. As shown in Table III, most duplicate ack accumulations were interrupted by data packets sent from the game clients. This happens because *game traffic is bi-directional*, so the game clients may have data to release before sufficient duplicate acks are generated. Fig. 2 shows that the client packet rate is generally higher than the server packet rate, which implies that client packets usually arrived at the servers before sufficient duplicate acks were elicited by server packets. Consequently, the fast retransmit mechanism failed to trigger in most cases.

Further evidence that fast retransmit is ineffective in MMORPGs, is provided by the transmission latency of retransmitted packets. Theoretically, the latency of a dropped packet that is detected after retransmission timeout occurs would have a latency equal to $(\text{RTO} + \text{RTT})$. As shown in Fig. 7, the distributions of the latency of retransmitted packets and that of $(\text{RTO} + \text{RTT})$ are approximately equivalent. The result provides strong evidence that most lost server packets were only detected and recovered after retransmission timeout occured.

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 follow-up packet is sufficient to make the source host aware of the dropped packets. 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.

## IV. Proposed Methodology

Our analysis of TCP's performance indicates that it is inappropriate for MMORPGs. Another transport protocol, UDP, is widely used in real-time applications, such as video streaming and VoIP, but it cannot be applied to

MMORPGs directly because some game messages require reliable transmission. In this section, we first discuss the design of efficient transport strategies for MMORPGs. Then, we propose three content-based transport strategies that provide an appropriate transmission guarantee for each type of game message in order to opportunistically improve the packet delivery performance.

## *A. Prerequisites of MMORPG Transport Protocol Design*

The transmission requirements of game packets are diverse because of the intrinsic characteristics of messages contained in the packets. For example, some messages require in-order and reliable transmission, while the loss of some messages is tolerable. For MMORPGs, we generally classify game messages generated by players into three types: *move*, *attack*, and *talk* messages.

- Move messages report position updates when an avatar moves or goes to a new area. Since only the latest location in the game play matters, the server simply discards out-of-date move messages.
- Attack messages correspond to an avatar's combat actions when it engages in fights with opponents. Such messages cannot be lost because each action will have some impact on the target. However, if several successive attack messages describe the same combat action against the same target, out-of-order arrivals of these messages can be tolerated, since they can be processed in a different order without affecting the final outcome.
- Talk messages convey the contents of conversations between players. To display the complete contents to players in exactly the same order as they were typed, talk messages must be transmitted in order and reliably.

Intuitively, it is clear that transmitting game messages with short delays and low delay jitters leads to high interactivity and responsiveness in the game play, as game messages can be processed more quickly and smoothly by the receiving hosts. Based on the above descriptions of game messages, we define three levels of transmission requirements for different types of messages.

- Under the strictest level, messages *must be transmitted in order and reliably*. For talk messages, this QoS level must be guaranteed so that the contents of conversations can be displayed correctly.
- For messages that *can be processed out of order*, in-order delivery is not essential. For example, if a player attacks an opponent with the same weapon, a series of repeat messages are generated. Since these messages are semantically and visually equivalent, out-of-order processing will not adversely impact the outcomes of the game play.
- Some types of messages *do not require reliable transmission* because they can be lost without affecting the game's logic. For example, the loss of most move messages is tolerable because they are designed to be *cumulative*; that is, each new message will override the previous ones. However, the last message in a series of move updates must be delivered to the server because it reports the avatar's latest location.

## *B. Opportunistic Content-based Transport Strategies*

To deliver game messages efficiently for different levels of transmission requirement without incurring high overhead, we designed the following transport options.

- **Multi-streaming**: With this option, different types of game messages can be put into separate streams, each of which processes the messages independently. In other words, each stream maintains the message order individually so that a delayed message in one stream will not affect subsequent messages belonging to other streams. Therefore, if different types of messages can be processed independently, we can put them into several separate streams without generating incorrect game semantics.
- **Optional Ordering**: We have shown that in-order delivery incurs long delays and extensive delay jitters due to inevitable packet loss. Optional ordering can reduce this overhead because it allows some types of messages to be processed as soon as they are received without being buffered if their preceding messages have not arrived.
- **Optional Reliability**: With this option, messages that do not require reliable transmission can simply be ignored if they are lost in the network. Therefore, the loss of a packet containing a non-critical message will not prevent the processing of its subsequent messages. In addition, optional reliability can save the network bandwidth used by unnecessary packet retransmission.

Next, we propose content-based transport strategies based on the above options. These strategies assign appropriate levels of transmission guarantee to game messages according to their requirements.

- **MRO** Strategy: MRO only uses *multi-streaming* (M); that is, it guarantees transmission reliability (R) as well as packet ordering (O). Under this strategy, as game messages are classified into three types, namely move, attack, and talk, we can employ separate streams to handle them.
- **MR** Strategy: MR implements both *multi-streaming* and *optional ordering*. This strategy provides two kinds of streams: ordered streams and unordered streams. If messages are tolerant of out-of-order processing, they can be transmitted via an unordered stream; otherwise, they should be put into an ordered stream.
- **M** Strategy: M combines all three options, that is, *multi-streaming*, *optional ordering*, and *optional reliability*. Under this strategy, there are three kinds of streams: ordered and reliable streams, unordered and reliable streams, and unordered and unreliable streams. Messages that must not be lost and must be processed in order must be transmitted via ordered and reliable streams. On the other hand, if messages can be lost without affecting the correctness of the game play, it is better to put them in unordered and unreliable streams. For messages that require reliable transfer without strict order maintenance, unordered but reliable streams should be used.

In the following section, we will use trace-driven experiments to evaluate the performance of the above strategies for transporting game messages, and compare them with a number of existing transport protocols.

## V. Performance Evaluation

In this section, we first discuss the transport protocols to be evaluated, namely TCP, UDP, DCCP, SCTP, and the protocols based on our proposed strategies. We then describe the game traces used in the network simulations and the setup of the experiment. In the performance evaluation part, we first assess the network performance of the existing protocols. Then, to demonstrate the efficacy of our opportunistic content-based transport protocols, we compare their performance with that of the existing protocols. Finally, we show how the improvement in network performance affects players' gaming experience.

In our evaluation, the end-to-end delay of a message is the difference between its sending time and its receiving time observed at the application level. For brevity, we denote messages sent by game clients as *client messages* and messages sent by the game server as *server messages*.

### A. Description of Evaluated Protocols

- **TCP**: We have already discussed TCP and analyzed its performance in Section III.
- **UDP**: UDP is a connectionless transport protocol that does not provide in-order delivery, reliable transmission, and congestion control. We expect that UDP will outperform the compared protocols in terms of end-to-end delay and end-to-end delay jitter because it does not stipulate loss recovery and packet ordering, both of which incur additional delays.
- **DCCP**: DCCP is designed for real-time multimedia applications that need a reasonable degree of control in the presence of network congestion [15]. It implements two congestion control mechanisms: a TCP-like congestion control (TCP-like) [10] and a TCP friendly rate control (TFRC) [11]. However, it is not reliable and it does not provide in-order delivery. To the best of our knowledge, no previous studies have evaluated its performance on MMORPGs. We employ DCCP with a TCP-like congestion control mechanism in our simulation.
- **SCTP**: SCTP [20] provides optional in-order transmission and multi-streaming. It can be flexibly configured to satisfy different levels of transmission requirement. We employ a simple SCTP configuration that only provides one ordered and reliable stream, which is the most common setting used by game developers. This configuration is similar to that used in [14].

To evaluate the effect of our three content-based strategies, we use them to develop the following three content-based transport protocols.

- $P_{MRO}$ implements the MRO strategy, which puts move, attack, and talk messages into three separate ordered and reliable streams.
- $P_{MR}$ is based on the MR strategy. It transmits move and attack messages via two unordered and reliable streams individually, while talk messages are put into an ordered and reliable stream.
- $P_{M}$ employs the M strategy, which transmits move messages via an unordered and unreliable stream, attack messages via an unordered and reliable stream, and talk messages via an ordered and reliable stream.

TABLE IV
COMPARISON OF THE PROTOCOLS USED IN THE NETWORK SIMULATIONS

| Protocol | Multi-streaming | Ordered | Reliable |
|---|---|---|---|
| UDP | ✘ | ✘ | ✘ |
| DCCP (TCP-like) | ✘ | ✘ | ✘ |
| TCP | ✘ | ✔ | ✔ |
| SCTP[†] | ✘ | ✔ | ✔ |
| $P_{MRO}$ | ✔ | ✔ | ✔ |
| $P_{MR}$ | ✔ | ▲ | ✔ |
| $P_{M}$ | ✔ | ▲ | ▲ |

Description: ✘ indicates that the transport protocol does not support the feature, ✔ indicates that the transport protocol supports the feature, and ▲ indicates that the transport protocol can support this feature optionally.
†: In our experiment, SCTP transports messages in one ordered and reliable stream.

TABLE V
ANGEL'S LOVE: ACTION TRACES PER USER

| | Game play time | Number of messages |
|---|---|---|
| Total | 22 hr. 7 min. | $7,482,951$ |
| Average | 4 hr. 37 min. | $4,080$ |
| Maximum | 22 hr. 7 min. | $169,116$ |
| Minimum | 1 hr. | 201 |

The main difference between our protocols and existing protocols is that the latter transfer each type of message in the same way, whereas our content-based transport protocols assign an appropriate delivery guarantee to each type of message. The features of the protocols are summarized in Table IV.

## B. Description of Game Trace

*Angel's Love* is a mid-scale, TCP-based MMORPG that is popular in Taiwan. There may be thousands of players online at any one time. The traces used in the experiment were collected by the staff of *Angel's Love*. They recorded the actions of all players during a specific period of time; that is, the traces are the action logs of the players. The action-based traces consist of game messages representing the actions performed by the players as well as the timestamp of each message. Based on the game-level action traces, we can perform network simulations realistically and flexibly. For example, we can investigate the end-to-end delay and end-to-end delay jitter that players experience in various network scenarios, such as different transport protocols, different numbers of clients, and different network configurations. *Such simulations cannot be conducted on network-level traffic traces because we cannot change the environmental conditions and network settings associated with the traces.* As summarized in Table V, the action-based traces were collected in a period longer than $22$ hours. Specifically, they contain more than 7 million action messages from about $1,800$ players.

Recall that game messages are classified into three types: move, attack, and talk. On average, for each player, $84\%$ of the messages are move actions, $12\%$ are related to attack actions, and $4\%$ are talk messages. We believe these results are generalizable to other MMORPGs because players move their avatars most of the time in MMORPGs. In addition, although players may feel that they spend similar amounts of time on fighting and chatting, attack messages easily outnumber talk messages. This is because an attack message is triggered by just a mouse click, whereas a talk message contains several words or letters that require several keystrokes.

## C. Experiment Setup

We conduct experiments with the ns-2 simulator, which is useful and therefore widely used in networking researches. As shown in Figure 8, we deploy $m + n + 3$ nodes: a game server and two intermediate nodes that serve as network routers; $n$ nodes are the game clients, and the remaining $m$ nodes are traffic nodes. The game clients and traffic nodes are randomly connected to one of the routers. For the links between the routers and the server, we set the bandwidth at $600$ Kbps, and the propagation delay at $70$ ms. For the links between the other

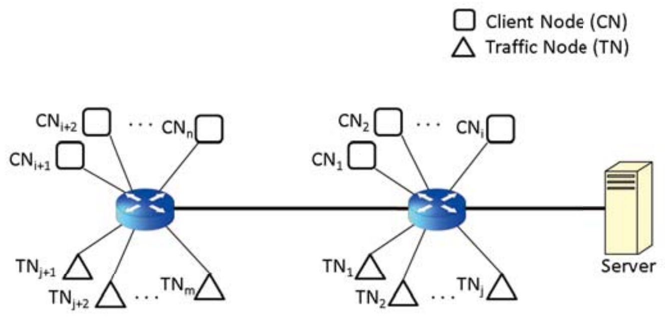

Fig. 8. Experiment network topology

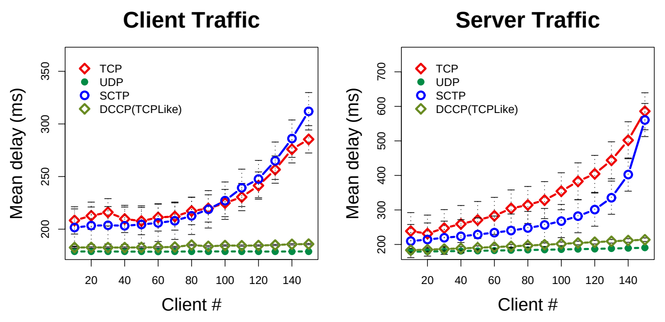

Fig. 9. The mean end-to-end delays of existing transport protocols

nodes and the routers, the bandwidth varies between $64$ Kbps and $128$ Kbps, and the propagation delay is set at $70$ ms. To simulate cross traffic on the Internet, $11$ pairs of traffic nodes are configured to generate UDP traffic in an exponential distribution with a sending rate of $500$ Kbps. The cross traffic causes a $4\%$ packet loss rate in the network approximately.

We implement the game server and the game client modules to simulate communications between the server and clients in MMORPGs. The game client module is *trace-driven*, so it can replay the action-based traces in the simulated network. In other words, the client module regenerates and sends game messages based on the times and the types of messages recorded in the traces. Meanwhile, the game server module tracks the locations of avatars by the move messages reported by game clients. When a client sends the game server a message about the avatar's next action, the server responds to the client and, depending on the type of the action, notifies the target clients related to the action (e.g., two players who are chatting) or other clients whose avatars are in the same zone (e.g., location updates).

We apply each of the discussed transport protocols to transport game messages. The number of clients is set between $10$ and $150$. For each protocol, we run each setting for $1000$ iterations, each of which simulates a different client arrival pattern. Specifically, the client interarrival time is a uniform distribution with an interval between $0$ and $1$ second.

## *D. Comparison of Existing Protocols*

We begin by investigating the mean end-to-end delays of transmitting game messages using the existing transport protocols. As the number of game clients in the simulations varies from $10$ to $150$, we can examine the performance under different network loads. Figure 9 shows that the mean end-to-end delay of server messages is more sensitive to the number of clients than that of client messages. The reason is that after a new player joins the game, the server needs to send out two additional messages: 1) messages for the new game client about the status of the other avatars; and 2) messages related to the new avatar for existing clients. Therefore, the number of server messages increases faster than linear growth as the number of clients increases. This effect is especially noticeable in the case of TCP and SCTP because both protocols provide reliable and ordered transmission; therefore, the packet loss caused by excessive messages causes long delays.

Next, we compare the mean end-to-end delay values under different protocols. As shown in Fig. 9, UDP always outperforms the other three protocols, and TCP yields the worst performance in nearly all scenarios. The result

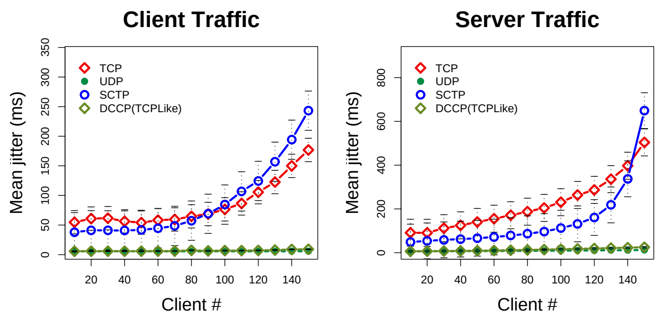

Fig. 10. The mean end-to-end delay jitters of existing transport protocols

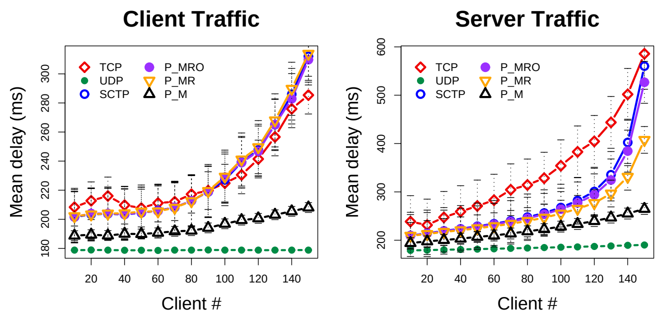

Fig. 11. The mean end-to-end delays of TCP, UDP, SCTP, and our transport protocols

confirms our intuition that TCP's guarantee of data delivery is too inflexible, so long delays are inevitable. On the other hand, since UDP does not guarantee data reliability and in-order delivery, no additional delay will occur in the transport level. In addition, DCCP(TCP-like) always achieves the second best performance, and SCTP yields the third best. The performance of SCTP is similar to that of TCP because we employ an ordered and reliable stream, which provides a transmission guarantee similar to that in TCP. DCCP(TCP-like) can be viewed as UDP plus a congestion control mechanism, so its mean delays are very close to those of UDP. Our observation supports the conjecture that the more functionalities a protocol provides, the worse the end-to-end delay experienced by packets.

We also consider the mean end-to-end delay jitter incurred by different protocols, as shown in Fig. 10. The trends in the results and the differences between the results of these protocols are very similar to those in Fig. 9, since a high average end-to-end delay usually implies that the end-to-end delay may vary over a wide range.

## *E. Evaluation of Content-based Transport Protocols*

We now compare the mean end-to-end delays of our proposed content-based transport protocols with those of TCP, SCTP, and UDP, as shown in Figure 11. Overall, the trends of our protocols as the number of game clients increases are similar to those of the compared protocols. We observe that the more flexible the strategies employed by our protocols, the better the performance they achieve. $P_{MRO}$ uses three TCP streams to transmit different types of messages so that the sequencing delays between different types of messages can be eliminated. Therefore, in most of the scenarios, the mean delays under $P_{MRO}$ are lower than those in TCP. For $P_{MR}$, move and attack messages are transmitted via two separate streams with optional ordering, so its performance is better than that of $P_{MRO}$. $P_M$ achieves the best performance among our protocols because approximately $80\%$ of move messages do not have to be transmitted reliably and in order. This avoids a large number of unnecessary retransmissions and reduces sequencing delays for non-critical move updates. As a result, the mean delay of $P_M$ is relatively close to that of UDP. We also analyze the mean end-to-end delay jitter, as shown in Fig. 12. Again, the results are similar to those for end-to-end delays, i.e., a high average end-to-end delay implies that the end-to-end delay jitter may be large. The results indicate that our strategies reduce end-to-end delay as well as end-to-end delay jitter.

To emphasize the performance of our protocols, we normalize the mean end-to-end delays and the mean end-to-end delay jitters of our transport protocols based on those of SCTP (upper bound) and UDP (lower bound). We

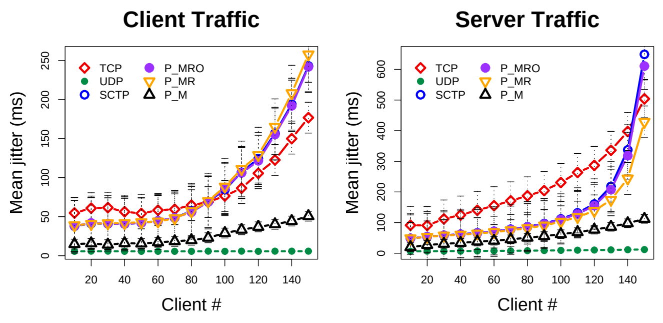

Fig. 12. The mean end-to-end delay jitters of TCP, UDP, SCTP, and our transport protocols

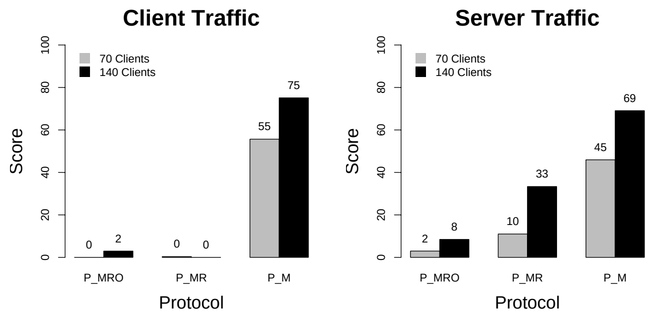

Fig. 13. Scores of mean end-to-end delays of our transport protocols

calculate the normalized score of the mean delay for each protocol by

$$\frac{\text{(mean delays of SCTP)} - \text{(mean delays of the protocol)}}{\text{(mean delays of SCTP)} - \text{(mean delays of UDP)}} \times 100.$$

The score of the mean delay jitter is calculated similarly. If a protocol has a high mean delay score, its performance is relatively close to that of UDP in terms of the mean delay, while a low score indicates the performance is close to that of SCTP. Figure 13 plots the scores of the mean delay for the scenarios with $70$ clients and $140$ clients. Obviously, $P_M$ outperforms the other two protocols. Furthermore, we observe that our transport protocols achieve a bigger improvement as the number of clients increases. The normalized scores of the mean delay jitter are shown in Fig. 14. The results are similar to those in Fig. 13 because reducing the end-to-end delay further reduces the end-to-end delay jitter effectively.

Finally, to examine the effect of providing different levels of transmission guarantee to different types of messages by using $P_M$, we plot the mean end-to-end delay of each type of message in Fig. 15. For the three types of messages, the magnitudes of the delays they encountered correspond to the transmission guarantee assigned to them. Thus, providing an appropriate guarantee for each game packet based on its content effectively minimizes end-to-end delay and delay jitter.

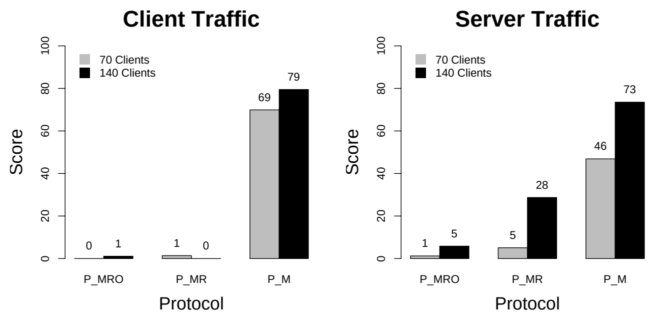

Fig. 14. Scores of mean end-to-end delay jitters of our transport protocols

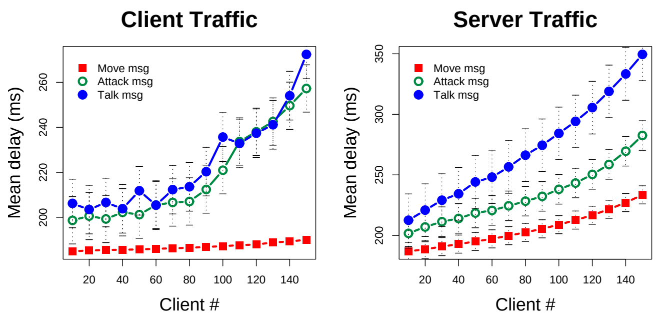

Fig. 15. The mean end-to-end delay of each type of messages in $\mathrm{P_M}$

## F. *Improvement in User Satisfaction*

Here, we consider how the improvement in network performance derived by our content-based strategies enhances users' game play satisfaction. In [6, 7], the authors provide an in-depth analysis of the relationship between network quality and users' game-playing time in MMORPGs. Degraded network QoS in terms of end-to-end delay, delay jitter, and the packet loss rate significantly affect the gaming experience, so players quit games because they are not satisfied. Based on the survival model used in [6, 7], we define the Game Satisfaction Index (GSI) as the level of player satisfaction calculated by the following equation:

$$\text{GSI} \propto \exp((-1.6) \times \text{delay} + (-9.2) \times \text{delay jitter} + (-0.2) \times \log(\text{loss rate})).$$

GSI indicates that the level of player satisfaction is proportional to the exponent of the weighted sum of certain network metrics, where the weights reflect the effect of network impairment. With this equation, we compare the improvement in user satisfaction with $\mathrm{P_M}$ over TCP. In the simulations, when the number of clients is $150$, the mean delay and mean delay jitter of client traffic are, respectively, $208$ ms and $52$ ms in $\mathrm{P_M}$, and $285$ ms and $177$ ms in TCP. The packet loss rate in $\mathrm{P_M}$ is similar to that in TCP. The ratio of the player satisfaction levels of these two transport protocols can be computed by $\exp((-1.6) \times (0.208 - 0.285) + (-9.2) \times (0.052 - 0.177)) \approx 3.57$, where $-1.6$ and $-9.2$ are the coefficients of delay and delay jitter respectively. In other words, the reduction in delay and delay jitter achieved by $\mathrm{P_M}$ raises the GSI significantly. Specifically, it is $3.57$ times higher than in the GSI for TCP. That is, players have much better gaming experiences with $\mathrm{P_M}$ as the underlying protocol, which implies that they tend to spend much longer time in the game. The figure manifests that the proposed content-based transport strategies can effectively raise player satisfaction levels in MMORPGs.

## VI. Conclusion

We have analyzed the performance of TCP in MMORPGs based on real-life traces. The evaluation results indicate that while TCP is a reliable transport protocol, using it to transmit all game packets degrades game's performance. Specifically, we have shown that TCP's congestion control mechanism and the fast retransmit algorithm are ineffective for MMORPGs. The performance problems are due to the following characteristics of game traffic: 1) tiny packets, 2) low packet rate, 3) application-limited traffic generation, and 4) bi-directional traffic. In addition, since not every game message requires reliable transmission and strict in-order processing by the destination host, both the loss recovery mechanism and the in-order delivery policy cause unnecessary transmission delays.

Having shown that TCP is unsuitable for MMORPGs, we propose three content-based transport strategies for this genre of games. The purpose of our strategies is to provide appropriate levels of transmission guarantee that exactly match the requirements of the game packets' contents. Based on realistic game traces, we conducted network simulations to evaluate the performance of our strategies and that of several existing protocols. The results indicate that, the proposed strategies reduce end-to-end delay and end-to-end delay jitter significantly. We also show that the improvement in network performance derived by our strategies can effectively raise the level of gaming satisfaction among players. To implement our strategies, it is necessary to identify game packets according to the types of contents. However, as these strategies are more efficient than non-content-based protocols, this additional task for game developers is worthwhile because it will bring much better gaming experiences to players.

## ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their helpful comments. This work was supported in part by Taiwan Information Security Center (TWISC), National Science Council under the grants NSC97-2219-E-001-001 and NSC97-2219-E-011-006. It was also supported in part by the National Science Council of Taiwan under the grants NSC96-2628-E-001-027-MY3 and NSC97-2221-E-001-009.

## REFERENCES

[1] "ENet: An UDP networking layer for the multiplayer first person shooter cube," http://enet.cubik.org/.

[2] "FAQ - Multiplayer and Network Programming," GameDev.Net, 2004. [Online]. Available: http://www.gamedev.net/

[3] M. Allman, V. Paxson, and W. Stevens, "TCP congestion control," RFC 2581, Apr. 1999.

[4] R. Braden, "Requirements for internet hosts - communication layers," RFC 1122, Oct. 1989.

[5] K.-T. Chen, P. Huang, and C.-L. Lei, "Game traffic analysis: An MMORPG perspective," *Computer Networks*, vol. 50, no. 16, pp. 3002–3023, Nov 2006.

[6] ——, "How sensitive are online gamers to network quality?" *Communications of the ACM*, vol. 49, no. 11, pp. 34–38, Nov 2006.

[7] ——, "Effect of network quality on player departure behavior in online games," *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, no. 5, pp. 593–606, May 2009.

[8] M. Claypool, "The effect of latency on user performance in real-time strategy games," *Elsevier Computer Networks, special issue on Networking Issues in Entertainment Computing*, vol. 49, no. 1, Sep 2005.

[9] M. Claypool and K. Claypool, "Latency and player actions in online games," *Communications of the ACM*, vol. 49, no. 11, Nov 2006.

[10] S. Floyd and E. Kohler, "Profile for datagram congestion control protocol (DCCP) congestion control ID 2: TCP-like congestion control," RFC 4341, March 2006.

[11] S. Floyd, E. Kohler, and J. Padhye, "Profile for datagram congestion control protocol (DCCP) congestion control ID 3: Tcp-friendly rate control (TFRC)," RFC 4342, March 2006.

[12] "OpenTNL: The Torque network library," GarageGames, April 2004, http://www.opentnl.org/.

[13] C. Griwodz and P. Halvorsen, "The fun of using TCP for an MMORPG," in *NOSSDAV '06: Proceedings of the 2006 international workshop on Network and operating systems support for digital audio and video*, 2006, pp. 1–7.

[14] S. Harcsik, A. Petlund, C. Griwodz, and P. Halvorsen, "Latency evaluation of networking mechanisms for game traffic," in *NetGames '07: Proceedings of the 6th ACM SIGCOMM Workshop on Network and System Support for Games*, 2007, pp. 129–134.

[15] E. Kohler, M. Handley, and S. Floyd, "Datagram congestion control protocol (DCCP)," RFC 4340, March 2006.

[16] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP selective acknowledgement options," RFC 2018, Oct. 1996.

[17] S. Pack, E. Hong, Y. Choi, llkyu 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*, vol. 4861, 2002, pp. 83–94.

[18] S. Shirmohammadi and N. D. Georganas, "An end-to-end communication architecture for collaborative virtual environments," *Computer Networks*, vol. 35, no. 2-3, pp. 351–367, 2001.

[19] R. W. Stevens, *TCP/IP Illustrated, Volume 2: The Implementation*. Addison-Wesley, 1995.

[20] R. Stewart, "Stream control transmission protocol," RFC 4960, September 2007.

[21] "ShenZhou Online," UserJoy Technology Co., Ltd., http://www.ewsoft.com.tw/.

[22] B. S. Woodcock, "An analysis of MMOG subscription growth – version 23.0." [Online]. Available: http://www.mmogchart.com/