Game Transport Protocol: A Reliable Lightweight Transport Protocol for Massively Multiplayer On-line Games (MMPOGs)[1]

Sangheon Pack, Eunsil Hong, Yanghee Choi, Ilkyu Park°, Jong-Sung Kim°, and D. Ko°°
Seoul National University, Seoul, Korea
Electronics and Telecommunications Research Institute, Daejeon, Korea°,
Taff System, Seoul, Korea°°

ABSTRACT

With the advent of high-speed access network technologies such as ADSL, increasing numbers of Internet users are participating in various interactive multimedia applications. Among these, the most popular are the massively interactive on-line games, or MMPOGs. In MMPOGs, a large amount of event data is associated with various control objects. This event data has different characteristics from that which is generally used on the Internet. Namely, events occur very frequently with short inter-arrival times and their size is quite small, because they only contain control information. Most commercial MMPOGs use TCP or UDP as the transport protocol for the event data. However, since TCP is such a heavy protocol, due to its complex congestion control algorithm and byte-oriented window scheme, it is difficult to support many concurrent users. On the other hand, UDP is a relatively lightweight protocol, but there are no functions available which permit reliable transmission and session management. In this paper, we propose a new transport protocol, Game Transport Protocol (GTP), which is designed for the transmission of the event data used by MMPOGs. GTP supports several functions designed to meet the various requirements of MMPOGs. Firstly, GTP uses a *packet-based window scheme* not a byte-based window scheme as in the case of TCP. This scheme is quite simple and suitable for the small size of the event data. Also, GTP performs *session management and retransmission* using GTP control blocks, and supports an a*daptive retransmission scheme* that controls the maximum number of retransmissions according to the real-time priority, in order to meet the time constraints of the event data. Although GTP is a specialized transport protocol, optimized for MMPOGs, it could also be utilized as a transport protocol for other interactive multimedia applications.

Keywords: Massively multiplayer on-line games, Game Transport Protocol, Packet-based window scheme, Adaptive retransmission scheme, Session management

1. INTRODUCTION

There has been a tremendous advance in Internet technology, and the World Wide Web (WWW) in particular has made it easy for many people to access the Internet. The WWW is undoubtedly the dominant application currently in use of the Internet, however, it is destined to be integrated with various multimedia applications in the next-generation Internet. Video on demand (VOD), Web-based Internet telephony, and Internet broadcasting are good examples of multimedia applications. Among these widespread multimedia applications, massively multiplayer on-line games have recently attracted considerable attention as a new Internet multimedia application.

The massively multiplayer on-line game is a type of distributed interactive real-time applications. Over the past years three distinct classes of distributed interactive real-time applications have become prominent: (1) military simulations, (2) networked virtual environments (NVEs), and (3) massively multiplayer on-line games (MMPOGs) [1]. The focus of scientific research has shifted from military simulations (in the 1980s) through NVEs (in the 1990s) and now to MMPOGs. Furthermore, the entertainment industry is investing substantially in MMPOGs, including in the areas of mobile gaming and online gaming in general. Fig. 1 shows the trend being followed by distributed interactive real-time

[1] This work was supported in part by the Brain Korea 21 project of the Ministry of Education, in part by the National Research Laboratory project of the Ministry of Science and Technology, and in part by the Electronics Telecommunication Research Institute, 2001, Korea.

applications with its evolving terminology. For example, until recently NVEs were usually called distributed virtual environments (DVEs), which then gave way to collaborative virtual environments (CVEs).

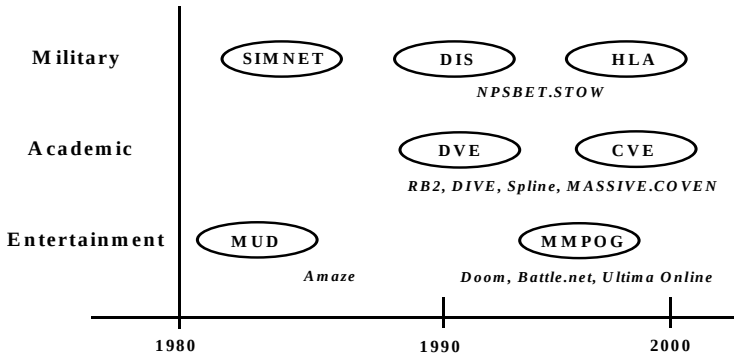

Fig. 1 History of distributed interactive real-time applications

Unlike other multimedia applications, MMPOGs have different characteristics in terms of their underlying network technology. In MMPOGs, there are many objects which need to be controlled by the users. Therefore, a considerable amount of event data containing control information is generated. In spite of the fact that the characteristics of game event data are different from those of general Internet data, most commercial on-line games use Transmission Control Protocol (TCP) or User Datagram Protocol (UDP) as the transport protocol for the game event data.

TCP is responsible for verifying the correct delivery of data from client to server. When data is lost in the intermediate network, TCP supports mechanisms to detect these errors, or the lost data, and to trigger retransmission until the data is correctly and completely received. On the other hand, UDP is a connectionless protocol that runs on top of the IP layer. Unlike TCP, UDP provides very few error recovery services, offering instead a direct way to send and receive data over the IP layer. It's used primarily for broadcasting messages over a network. Because these two transport protocols are implemented in almost all network devices, it is easy to deploy new applications using them. However, since TCP is such a heavy protocol, due to its complex congestion control algorithms and byte-oriented window scheme, it is difficult to support many concurrent users with this protocol. Furthermore, in TCP, even moderate congestion makes the game data unusable, so it is a particularly inefficient protocol for MMPOGs. On the other hand, UDP is a relatively lightweight protocol, but there are no functions which enable for reliable transmission and session management. Therefore, neither TCP nor UDP is the best solutions for the transmission of game event data. The description of the characteristics of game traffic, presented in the next section, demonstrates that neither of these two dominant protocols is suitable as the transport protocol for game event data.

To support the efficient delivery of game traffic over IP networks, many solutions have recently been proposed [2] [3]. However, since these proposals focused on system architectures rather than on transport protocols, they do not represent optimized solutions for game traffic. In this paper, we propose a new transport protocol, Game Transport Protocol (GTP), which is designed for the transmission of game event data in MMPOGs. This protocol supports several functions designed to meet the various requirements of MMPOGs, i.e. a packet-based window scheme, an adaptive retransmission scheme, priority based marking, etc.

The remainder of this paper is organized as follows. Section 2 shows the characteristics of event data in MMPOGs, based on the results of traffic measurement during real massively multiplayer on-line games. Section 3 describes the specification of GTP. Section 4 summarizes the main features of GTP. Section 5 introduces the result of GTP implementations. Finally, Section 6 concludes this paper.

2. CHARACTERISTICS OF GAME TRAFFICS

The development of GTP was motivated by the fact that game traffic in MMPOGs has characteristics which are different from those of other Internet applications such as the WWW, File Transfer Protocol (FTP), e-mail services, etc. Several previous reports have documented the characteristics of game traffic [4] [5]. However, since game traffic is highly dependent on the features of the game itself, it is difficult to generalize these characteristics. Therefore, in order to characterize game traffic, we selected certain representative network games which are currently very popular. For the traffic analysis, we measured the inter-arrival time of packets and their packet size. Fig. 2 and Fig. 3 give the probability density functions (pdf) of the inter-arrival time and the packet size, respectively. The representative MMPOG chosen is "StarCraft", which is currently the most popular networked on-line game worldwide. In this game, TCP is used for session and user management, while UDP is used for delivery of game event data.

In terms of packet size, almost all event data consists of less than 64 bytes, with 60% of this data being made up of just 50 bytes. This means that the packet size of game traffic is much smaller than that of general data traffic and that the variance is also small. According to the measured results, the inter-arrival time is very small, generally less than 100 msec. This shows that event data tend to occur in the form of bursts.

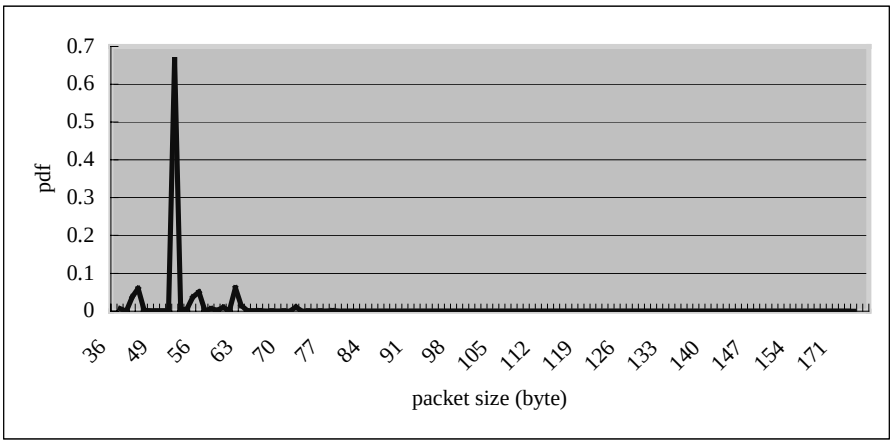

Fig. 2 Probability density function of packet length

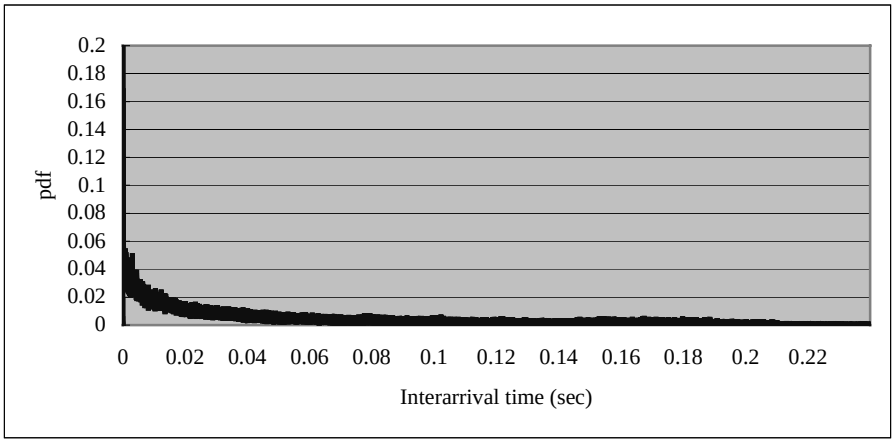

Fig. 3 Probability density function of inter-arrival time

As a result of the traffic analysis, we defined several requirements for a new transport protocol for MMPOGs.

Low delivery latency: The most important factor in MMPOGs is to deliver game event data within the specified time boundary. For example, if the event data for the movement of an object in the game is delivered after the specified time limit, the data will become useless and, as a result, user satisfaction will decrease dramatically. Therefore it is essential to guarantee low delivery latency. However, it is impossible to provide on-time delivery over the current IP networks, because IP networks support only best-effort services. Of course, there are many proposals for providing Quality of Service (QoS) over IP networks, however, for a variety of reasons these solutions have not been widely deployed. Therefore, we propose a flexible method in GTP for on-time delivery. Although this method does not guarantee absolutely on-time delivery, it can be selectively inter-worked with various existing network technologies such as DiffServ, MPLS, etc.

Adaptive reliable transmission: In MMPOGs, it is necessary to deliver event data reliably in order for the games to be able to progress correctly. Therefore, many commercial games use TCP as a transport protocol in spite of its overhead. However, not all event data has to be delivered reliably. For example, in on-line shooting games, when a user pushes the spacebar hundreds of times to shoot the gun, each key pressed generates event data. However, although a portion of the event may be lost, this has no effect on the progress of the game. Therefore, this kind of data does not require guaranteed reliable transmission. On the other hand, user commands that make an object move in a certain direction should be delivered without any data loss and retransmission in case of packet loss should be supported in this case. To take into account these particular characteristics, we propose an adaptive reliable transmission scheme.

Low protocol overhead: MMPOGs based on the client-server model are designed to support millions of concurrent users. Therefore, the transport protocol used should be a light-weighted protocol. The currently used transport protocol, TCP, is not a lightweight protocol, because of its complex flow control mechanisms and control blocks. Therefore, it cannot meet the scalability requirements of MMPOGs. Furthermore, since TCP uses a byte-based window scheme, it is difficult to support retransmission. Therefore, in developing GTP, we focused on reducing the protocol overhead for the sake of scalability.

Connection-oriented service: In terms of low protocol overhead, UDP may be a suitable choice as a transport protocol in MMPOGs. However, it does not provide any retransmission mechanism or flow control scheme. In addition, it transmits data using a connection-less service model. However, in MMPOGs, it is essential to manage the game states for each user, so that a connection-oriented service model is required. Therefore, GTP was designed as connection-oriented protocol, unlike UDP.

3. GAME TRANSPORT PROTOCOL (GTP)

3. 1 GTP HEADER STRUCTURE

Since GTP needs to provide reliable transmission, its header structure is similar to that of TCP. However, several fields are modified to support the unique characteristics of GTP. Fig. 4 shows the structure of the GTP packet header. Detailed descriptions for each field are as follows.

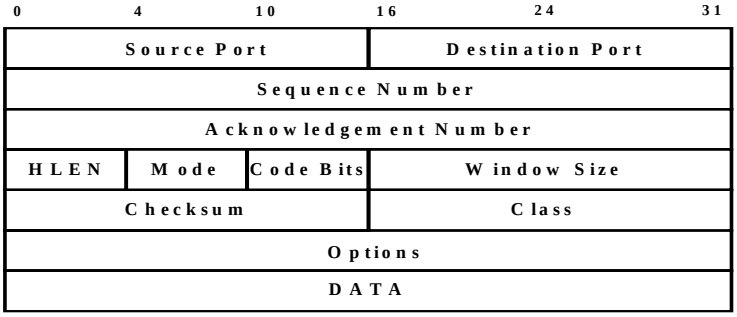

Fig. 4 GTP Packet Header

• **SOURCE PORT** & **DESTINATION PORT**: They contain the GTP port numbers that identify the application programs at each end of the GTP connection. They perform the same functions as the source and destination ports in TCP.

• **SEQUENCE NUMBER** & **ACKNOWLEDMENT NUMBER**: In TCP, the SEQUENCE NUMBER field is used to identify the position of the data in the packet in the sender's byte stream. However, GTP uses the packet-based window scheme and not the byte-based window scheme. Therefore, in GTP, this field identifies the position in the sender's packet stream. The ACKNOWLEDMENT NUMBER field identifies the number of the packet that the source expects to receive next.

• **HLEN**: This field contains an integer that specifies the length of the packet header measured in multiples of 32-bits. It is needed because the OPTIONS field varies in length, depending on which options have been included. Thus, the size of the GTP header varies according to the options selected.

• **MODE**: The characteristics of game traffic in MMPOGs vary according to the categorization and contents of the games. Therefore, it is necessary to provide diverse flow control options within the application layer. By offering multiple options, applications can selectively choose a suitable flow control mechanism. The MODE field is therefore designed to support the multiple options available in the transport layer. More detailed descriptions are given in section 4.3.

• **CLASS**: This field is used to separate GTP packets into several classes. This classification information is used in the adaptive retransmission scheme and the inter-working with DiffServ. More detailed explanations are presented in section 4.3.

• **CODE BITS**: GTP uses the 4-bit field labeled CODE BITS to determine the purpose and contents of the GTP packet. These four bits allow the interpretation of other fields in the header according to Table 1.

Bit (left to right)	Meaning if bit set to 1
SYN	*Synchronize sequence numbers*
FIN	*Sender has reached end of its data stream*
RST	*Reset the connection*
ACK	*Acknowledgement field is valid*

Table 1 Bits of the CODE BITS field in the GTP header

• **WINDOW SIZE**: The WINDOW SIZE field advertises how much data a GTP session is willing to accept, by specifying its buffer size every time a packet is sent.

• **OPTIONS**: The OPTIONS field is defined to provide additional functions in a GTP session. It is not currently used.

3.2 CONNECTION SETUP & RELEASE

In MMPOGs, there are continuous interactions between game servers and user clients. Therefore, connection-oriented data transmission should be supported in MMPOGs. For the purpose of establishing connections, GTP uses a three-way handshake method that is similar to that used by TCP. To support full-duplex communication, GTP uses only three frames instead of four packets.

Fig. 5 shows the connection establishment procedure. Connection setup begins by the client setting a bit, known as the SYN bit, within the GTP header, indicating a request for synchronization with the destination GTP process. The receiving host's GTP session must acknowledge the receipt of this SYN request and send its own SYN request. The SYN request should also contain the ACK bit. Namely, the packet has both the SYN and ACK bits set. The SYN

request must be acknowledged by the sending client. By means of these message flows, a full-duplex connection is established.

If the client does not need to maintain the established connection with the server any longer, it can tear down the connection. The GTP session teardown process utilizes the same three-way handshake method as was used for session setup. It utilizes a three-frame exchange to close the session. The client requesting session closure sets the FIN bit within the GTP header, indicating that the request is to close the session; it then sends the FIN packet to the other side. The recipient must in turn acknowledge receipt of the initial FIN packet and send its own FIN packet, which is then acknowledged by the original host requesting to close the session. At this point both sides release the allocated resources and make them available for other processes. Fig. 6 shows the message flows in the connection release process.

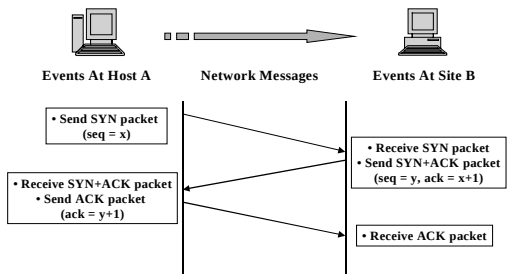

Fig. 5 Connection setup

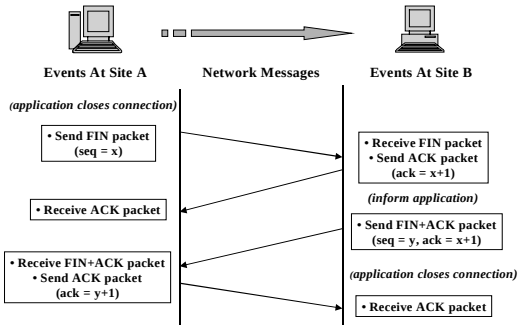

Fig. 6 Connection release

4. CHARACTERISTICS

4.1 PACKET-BASED WINDOW SCHEME[2]

TCP views the data stream as a sequence of bytes that it divides into segments for transmission. Usually, each segment travels across the Internet as a single IP packet. TCP uses a sliding window mechanism for efficient transmission and flow control. The TCP window mechanism makes it possible to send multiple segments before an acknowledgement arrives. Doing so increases total throughput because it keeps the network busy. The TCP sliding window protocol also solves the end-to-end flow control problem, by allowing the receiver to restrict transmission until it has sufficient buffer space to accommodate more data [6].

The TCP sliding window mechanism operates at the byte level, not at the segment or packet level. Bytes of the data stream are numbered sequentially, and a sender keeps three pointers associated with every connection. These pointers define the sliding window. The first pointer marks the left of the sliding window, separating bytes that have been sent and acknowledged from bytes yet to be acknowledged. A second pointer marks the right of the sliding window and defines the highest byte in the sequence that can be sent before more acknowledgements are received. The third pointer marks the boundary inside the window that separates those bytes that have already been sent from those octets that have not yet been sent. The protocol software sends all bytes in the window without delay, so the boundary inside the window usually moves quickly from left to right.

However, the byte level sliding window mechanism induces more overhead than the packet level sliding window mechanism. In terms of retransmissions, a TCP sender has to record bytes acknowledged as well as bytes to be

[2] In this section, the term “packet” means a data segment delivered from the application layer to the transport layer.

acknowledged. Furthermore, it has to compute how many octets should be transmitted in the next time slot and what the first and last bytes to be transmitted are. If more complex flow control schemes such as the selective ACK scheme are used, even more computing overhead may be required. As mentioned above, since the size of game event data is much less than that of general data, a lightweight transport protocol is more appropriate for its transmission. To reduce this overhead, we used the packet level sliding window scheme in GTP. Of course, the byte-based window scheme is more efficient in case of a variable size data stream, however, as mentioned above, the variance of the game event data length is quite small. This means that most event data have similar packet sizes. Therefore, the packet-based window scheme is more suitable for the transmission of game event data.

In this scheme, a client (or a game server) sends data to a server (or a client) in units of packets rather than bytes. In TCP, which uses the byte-based window scheme, much overhead is required to divide application-level data into byte streams and to reassemble them at the receivers. Furthermore, it causes flow control and retransmission to be quite complex operations. However, since GTP uses the packet-based window scheme, flow control schemes can be implemented more easily. Fig. 7 compares two schemes: the byte-based window scheme and the packet-based window scheme.

In the byte-based window scheme, the data stream at the application level is divided into several packets containing a certain number of bytes, and each fragment of data is transmitted as a packet. The packet size is determined by the flow control mechanism being used. In the case of short sized data segments, many portions of the packet are filled with useless data (padding). Furthermore, since long sized data segments are fragmented into several packets, additional packet handling is required at the receivers. On the other hand, in the packet-based window scheme, since each data segment at the application level forms a packet to be transmitted, there is no fragmentation or padding. Of course, when the data segment size varies greatly, the packet-based window scheme may lead to network congestions. In TCP, when network congestions occurs, the sender decreases the number of bytes to be sent in the next time. Doing so prevents severe network collapses. On the other hand, in the packet-based window scheme, when the congestions are occurred, the sender decreases the number of packets to be sent. However, since a packet may be quite large, the fact that the number of packets is small doesn't mean that the number of bytes to be sent is any less than before. However, as described in section 2, game event data involves small sized packets and most packets have similar sizes. Therefore, it is more efficient to utilize the packet-based window scheme for the transmission of game event data.

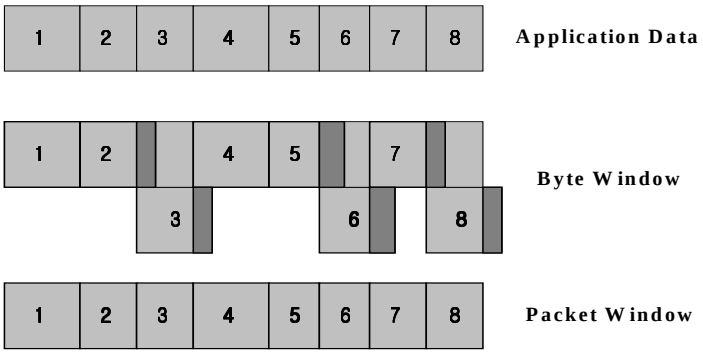

Fig. 7 Comparison of two window schemes: Byte-based window vs. Packet-based window

4.2 ADAPTIVE RETRANSMISSION SCHEME

GTP supports a retransmission scheme for the reliable data transmission. The retransmission scheme is implemented using the packet level window scheme described in the previous section. Unlike general data traffic, game traffic should be delivered within a pre-defined time boundary. Furthermore, considerable real-time traffic is generated by MMPOGs

during short time intervals. If some packets, having a specific time boundary, are delivered outside of this time boundary, the packets may no longer be useful for the receiver. In this case, it would be more efficient not to retransmit these packets to the receiver.

To support this selective retransmission, we used an adaptive retransmission scheme in GTP. In GTP, all packets are categorized into several classes using the **CLASS** field. The **CLASS** field is also used for interworking with the DiffServ model. This is described in the next sub-section. Up until now, we have defined just four retransmission types based on the **CLASS** field. Following packet classification, each packet is treated according to a different retransmission policy. Table 2 shows the different retransmission schemes. In the case of adaptive retransmission, we adopted several adaptive retransmission schemes which are presented in [7].

CLASS field	Category	Retransmission Scheme
000	**Scheme 1**	*Retransmit all non-delivered packets*
001	**Scheme 2**	*Retransmit packets up to N times*
010	**Scheme 3**	*Retransmit packets up to M times*
011	**Scheme 4**	*No retransmit*
Others	**None**	*Currently Unused*

Table 2 The **CLASS** field and retransmission schemes

In scheme 1, a sender retransmits all non-delivered packets until the receiver eventually receives them. On the other hand, in schemes 2 and 3, the sender retransmits non-delivered packets up to N or M times, respectively. N and M represent the maximum number of retransmission and N is set to a larger value than M. The only difference between schemes 2 and 3 is the maximum number of retransmissions. The following pseudo-code describes the retransmission procedures in the case of scheme 2:

when the receiver detects the loss of packet A:

if(C > N)

give up the packet's delivery, namely ignore the packet;

end;

if(Tc + RTT + Ds < Td(A))

send the request for retransmission of packet A to the sender;

increase the value of C;

where ***C*** is the number of retransmission up to the current time, ***N*** is the maximum number of retransmission in the scheme 2, ***Tc*** is the current time, ***RTT*** is the estimated round trip time, ***Ds*** is a slack term, and ***Td(A)*** is the time when packet A is scheduled for playing. The slack term, ***Ds***, could include the error tolerance in estimating ***RTT***, the sender's response time to a request, and/or the receiver's processing delay. These schemes are aimed at minimizing the number of requests for retransmissions that will not arrive in a timely manner for playing. It is clear that if ***Tc+RTT+Ds < Td(A)***, the retransmitted packet is expected to arrive in a timely manner for playing. The timing diagram for receiver-based control is shown in Fig. 8.

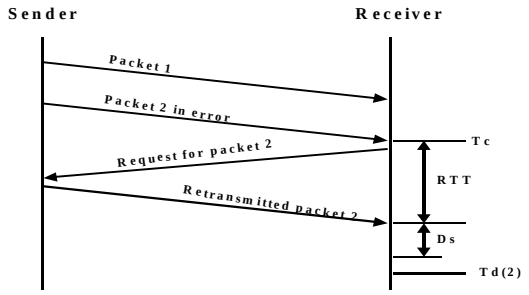

Fig. 8 Timing Diagram

4.3 PRIORITY-BASED PACKET MARKING

As mentioned above, on-time delivery and low latency are the most important issues in MMPOGs. However, it is impossible to provide on-time delivery in transport layers without appropriate support from the lower layers. To meet these constraints, the Internet Engineering Task Force (IETF) has proposed several models: Integrated Service (IntServ) [8], Differentiated Service (DiffServ) [9], etc. Although IntServ can provide per-flow guaranteed services, it is not a scalable solution. Therefore, DiffServ may be a more suitable solution for wide area networks.

In this paper, we utilized the CLASS field to interact with the DiffServ model. In the DiffServ model, since all packets are processed according to Per-Hop Behaviors (PHBs) at each node, PHBs for the game event data should be defined. Table 2 shows an example of a mapping table between game traffic and DiffServ PHBs. In this example, we used Assured Forwarding (AF) as a PHB for game event traffic. Defined in [10], AF is actually a PHB for edge-to-edge services specified in terms of relative bandwidth availability and multi-tiered packet drop characteristics. Whereas Expedited Forwarding (EF) supports services with “hard” bandwidth and jitter characteristics, AF allows for more flexible and dynamic sharing of network resources supporting the “soft” bandwidth and loss guarantees appropriate for bursty traffic. [10] currently defines four service classes and three levels of drop precedence. A shorthand notion exists for referring to a particular AF PHB. AFxy refers to AF service class x with drop precedence y. Specific drop probabilities for each precedence level are assigned by the network operator to meet the desired packet loss characteristics of each class. The only requirements are that drop precedence 3 have a more aggressive drop probability than precedence 2, which in turn should have a more aggressive drop probability than precedence 1. In addition, the probability functions may vary from one class to another, for example packets marked AF12 (service class 1, drop precedence 2) may be subject to an entirely different drop probability function than packets marked AF22 (service class 2, drop precedence 2).

Fig. 9 shows the marking procedure for the systems at each end of the network connection. In the application layer, data segments are classified into several classes according to their priorities. The transport layer, where GTP is used, marks the **CLASS** field in the classified segments and delivers them to the IP layer. In the IP layer, GTP packets are mapped to correspondent PHBs using a mapping table similar to the table in Table 3.

CLASS field	DiffServ PHB	Description
000	**AF11**	*Highest importance level*
001	**AF12**	*Medium importance level*
010	**AF13**	*Lowest importance level*
Others	**None**	*Currently Unused*

Table 3 A mapping table between **CLASS** field and DiffServ PHB

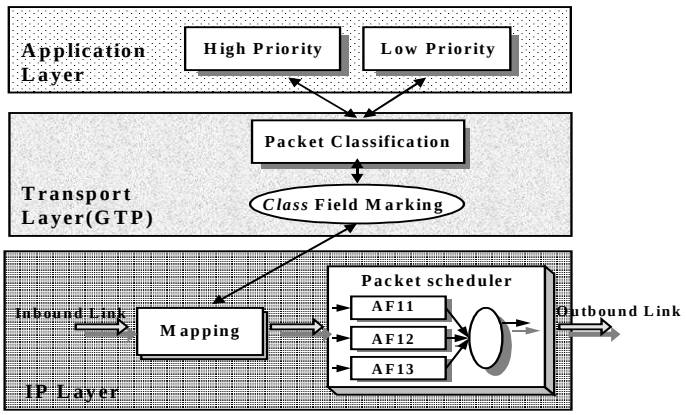

Fig. 9 Marking procedure in end hosts

4.4 MULTIPLE MODES SUPPORT

The motivation behind GTP is to provide MMPOGs with a more efficient transport layer protocol. However, it is difficult to define the common requirements of MMPOGs. In some cases, reliable transmission is more important than on-time delivery (e.g. On-line chess games). On the other hand, in other MMPOGs, such as shooting games, on-time delivery is the most critical factor. In order to provide for these varying characteristics, GTP provides multiple mode options. The developers can choose a suitable options using upper layer interfaces, namely socket APIs, according to the requirements of the applications that GTP is being used for. In the current version of GTP, we defined the four options described below. Since the **MODE** field in the GTP header is a 4-bit field, it is possible to support more options.

In mode 1, retransmission and flow control are not supported. Namely, mode 1 supports similar functions to those of UDP. Mode 2 and 3 support only the retransmission or only flow control, respectively. On the other hand, in mode 4, GTP supports both retransmission and flow control.

- **MODE 1**: No retransmission & No flow control
- **MODE 2**: Retransmission & No flow control
- **MODE 3**: No retransmission & Flow control
- **MODE 4**: Retransmission & Flow control

5. IMPLEMENTATION

We implemented and tested GTP in the Microsoft Windows operating system environment. Although GTP is a transport layer protocol, we implemented it over the UDP layer due to difficulties encountered in decoupling the network layer and the transport layer in the Windows operating system. Therefore, several redundant functions are implemented, e.g. those dealing with port numbers.

GTP was originally designed as a new transport protocol in the middleware system developed by the Electronics and Telecommunications Research Institute (ETRI) in Korea [11]. The overall system architecture is presented in Fig. 10. The GTP packet queue is used for sending GTP packets to the network buffer in the OS layer and receiving IP packets from the network buffer. The GTP runtime module provides several functions such as retransmission, flow control, etc. The GTP socket is an interface between the network layer and the upper layer. It notifies the game engine about events which have occurred.

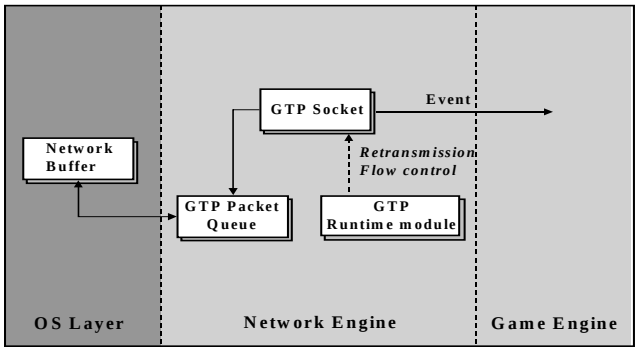

Fig. 10 Overall System Architecture

6. CONCLUSION & FUTURE WORK

In this paper, we proposed a new transport protocol for massively multiplayer on-line games (MMPOGs), named Game Transport Protocol (GTP). In order to meet certain requirements of MMPOGs, GTP supports several functions: the packet-based window scheme, the adaptive retransmission scheme, the priority-based marking scheme, as well as allowing for multiple options. From traffic measurement results, we found that the game events generally consist of bursts of small amounts of data. Consequently, GTP was developed on the basis of this observation, in order to provide for more efficient transmission of game traffic. GTP is currently implemented in the Windows operating system environment and is utilized in the ETRI 3D game engine.

In the future, we plan to implement TCP friendly rate control (TFRC) mechanisms [12]. These are proposed as a means of overcoming the unfairness which exists between TCP flows and non-TCP flows. Since many Internet applications use TCP as a transport protocol, GTP should be able to support the TFRC mechanism, in order to coexist with other flows.

Since GTP is a specialized transport protocol, designed for MMPOGs, it cannot entirely replace the existing transport layer protocols such as TCP and UDP. However, when we consider the tremendous growths in the game markets, the development of GTP represents a meaningful approach to providing MMPOGs with efficient network functions, and it is hoped that it will be widely deployed in client-server programs.

7. REFERENCES

1. Jouni Smed, Timo Kaukoranta, and Harri Hakonen, “A review on networking and multiplayer Computer Games,” Turku Centre for Computer Science (TUCS) Technical Report No. 554, April 2002.
2. Daniel Bauer, Sean Rooney, Paolo Scotton, “ Network Infrastructure for Massively Distributed Games,” ACM NetGames 2002, April 2002.
3. Stefan Fiedler, Michael Wallner, and Michael Weber, “A communication architecture for massive multiplayer games,” ACM NetGames 2002, April 2002.
4. Michael S. Borella, “Source Models of Network Game Traffic,” Computer Communications, February 2000.
5. Johannes Färber, “Network Game Traffic Modeling,” ACM NetGames 2002, April 2002.
6. Douglas E. Comer, “ Interworking with TCP/IP – Principles, protocols, and architectures,” 4th Edition, Prentice Hall, 2000.
7. Dapeng Wu, Yiwei T. Hou, and Ya-Qin Zhang, “Scalable Video Coding and Transport over Broad-Band Wireless Networks,” Proceeding of IEEE, January 2001.
8. R. Braden, D. Clark, and S. Shenker, "Integrated Services in the Internet Architecture: an Overview," IETF RFC 1633, June 1994.
9. S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An Architecture for Differentiated Services," IETF RFC 2475, Dec. 1998.

10. Heinanen, J., F. Baker, W. Weiss, and J. Wroclawski, “Assured Forwarding PHB Group,” IETF RFC 2597, June 1999.
11. Technical Report, “A Study on the Large Scale 3D Information Manipulation and Collaboration Middleware Technology for Internet Virtual Space”, ETRI, Korea, December 2001.
12. The TCP Friendly Website, http://www.psc.edu/networking/tcp_friendly.html.