

1. TITLE OF THE INVENTION

A method for reliable and high speed data transfer

2. DETAIL DESCRIPTION OF THE INVENTION**2.1. Industrial Fields of Application**

In the information technology world, server-client communication systems are common. Server and client are end systems connected by wire or wireless network. The server's role is providing services while the client's is requesting for services. Different services have different requirements for data transfer. Data oriented services require low error/loss rate while voice and multimedia services require low delay and can tolerate limited error/loss rate. The present invention satisfies various requirements of different services by providing high speed, reliable data transfer.

The invention can be used by server-client communication protocols and applications that require high speed data transmission and reliability. It can also be used by networked multimedia systems such as Video on Demand, Videoconference that are sensitive to network delay and delay variation.

This Sheet Has Been Replaced



2.2. Description of Prior Art

2

Prior arts use a transmission-reception method so called PUSH method ("DAVIC 1.0 Specification", by Digital Audio-Visual Council, 1995-1996). Generally, PUSH method works in such a way that client sets up a session with server, after session set up, the server transmits data to the client at a pre-set rate. On the other end, client must receive data as fast as server's transmission rate. With PUSH method, there is no transmission control between client and server except for maintaining a pre-set transmission and reception rate. To perform end-to-end transmission control, prior arts use Transmission Control Protocol (TCP) ("Transmission Control Protocol" by J. Postel, IETF RFC 793). TCP is responsible for breaking up the message into packets, reassembling them at the other end, resending any packet that gets lost, and putting packets back in the right order. It uses a send-acknowledge approach to make sure that packets sent by the sender arrive at the recipient. It uses a slide window to co-ordinate sender's transmission and recipient's reception. A checksum is added in the packet header for checking whether a packet received by the recipient is same as what sender sent. A resending scheme is used to re-transmit error/lost packets. TCP is tuned for the networks relying on Internet Protocol (IP) to provide reliable data transfer across error prone networks.

2.3. Problems that the Invention is to Solve

The PUSH method suffers from asynchronization. Asynchronization refers to those situations in which server's transmission speed is different from client's receiving speed. There are two cases of asynchronization. In the first case, the server sends data faster than client can receive. As a result, buffer on client's network interface card (NIC) is overwritten, which causes data loss. In the second case, the server sends data slower than client requires. In such case, client can not get data in time. For delay sensitive service such as voice, the quality of sound becomes intolerable. In both cases, the server has no way to know how client performs and requires. It's also difficult to implement PUSH method. The difficulties come from two aspects. The first is with the server. It requires a high resolution clock and precise controls to maintain constant transmission speed at every instance. The second problem is on the client side. Usually, data is compressed before transfer. Some portion can be highly compressed while others can be low. After receiving the compressed data, client does decompression. The decompression restores variable amount of data from packet to packet. Even constant transmission and receiving rates are possible, it doesn't mean that the time taken for processing each packet is constant. With PUSH method, there is no way to control server from transmitting data when client needs more time to process the received data and vice versa.

TCP is created for low speed, error prone networks. As technology develops, modern network such as Asynchronous Transfer Mode (ATM) is capable of providing high speed and reliable transmission at lower network layers. TCP is no longer suitable for such networks ("TCP over ATM: Some Performance Results" by Romanow and Allyn, ATM Forum/93-784). Besides that, TCP introduces significant overheads. The overheads decrease the performance of high-speed data transfer. Neither PUSH method nor TCP supports dynamic transmission configuration. For example, packet size and transmission rate are fixed once transmission starts.

This Sheet Has Been Replaced





Network jitter problem refers to delay variation³. The problem can cause data not to arrive at the recipient at the expected instance. As such, it can cause data loss or data deficiency at client end due to buffer overflow or underflow.

The present invention provides methods of solving the above mentioned problems. Prior arts are either difficult to implement or lead to excessive overhead that affects data transfer performance. On the contrary, the invention not only provides reliable and prompt data transfer to meet the requirements of various kinds of services, but also reduces implementation complexity and thus improves data transfer performance. As an added advantage, it supports dynamic transmission configuration.

2.4. Means of Solving the Problems

To resolve the problems outlined in section 3.3, the invention applies a request-respond approach. In essence, it makes use of a master clock at the client end to synchronize with a slave clock at the server end. Following the master clock, the server is able to determine when it shall send data to the client. The invention works through a Request module and a Response module embedded into client and server respectively. A backward channel is provided to link the Request and the Response module. A request (REQ) message is sent between two modules to convey synchronisation signal. The message also carries information of client's state. Server's transmission is governed by the REQ message. Based on the information in the REQ message, server can configure packet size, transmission rate and adjust transmission dynamically.

In order to solve the network jitter problem, the invention applies an elimination method. The impact of jitter problem is to cause either overflow or underflow to client's buffer. Being granted the control of delivery of data, client can choose not to request for new packets until overflow trend disappears or to request for more packets with faster transmission rate when underflow happens. Client uses a buffer monitor to keep track of its buffer state. With the feedback information in the REQ message, server knows the impact of jitter on the previous packet. It can then compute the transmission rate and data amount according to client's current state to eliminate the jitter impact. Such computations are provided in this invention.

2.5. Operation of the Invention

The operation of the invention is as follows. The client requests for data from server by sending a REQ message through Request module. Server must wait for a REQ message before sending out a packet. The client, upon receiving a packet, retrieves it from NIC buffer and transfers it into an application provided buffer. When the current buffer level drops below a specified threshold, the Request module is triggered to proceed the master clock. When the master clock is changed, the Request module sends a REQ message through the backward channel to server. REQ message carries information of client buffer's state. It is used to synchronise the server's slave clock and by means it also notifies server that the client has received previous packet and requests for new packet. Server waits and receives the REQ message through Response module. After receiving the REQ message, server adjusts the slave clock to synchronise with the master clock. Based on the information carried by the REQ message, server is able to



estimate how big and how fast the next packet should be sent. After assembling the packet and adjust transmission rate, server sends the packet to the client. Because server follows client's request, it is able to send out data only when it is needed.

2.6. Embodiments

The first embodiment of this invention is described with reference to Figure 1. In this figure, the numeral 1.1 is server. It is a back end system that maintains information resource and provides service by sending the requested data to a client that originates a request. The numeral 1.2 is client. It is a front end system that requests service from server, receives data sent by server and presents it to user. Server and client system may be connected by a wire or wireless network. The numeral 1.3 is forward channel. It is a unidirectional network connection established for transmitting data from server to client. The numeral 1.4 is backward channel. It is another unidirectional network connection established specifically for the invented method.

The embodiment of Figure 1 operates as follows. When the client requests for data, the master clock is proceeded. The request module sends a REQ message to the server. Server waits for a REQ message from client. Upon receiving a REQ message, server adjusts the slave clock to be synchronized with the master clock at the client end. In response to the REQ message, the server sends out an amount of data in a packet based on some computation results given out in embodiment presented later. The term "packet" here refers to application defined data structure. It may consist of multiple network packets. Client receives the incoming packet.

The second embodiment of this invention is described below with reference to Figure 2, 3 and 4. It describes how the invented method works in the server system. In Figure 2, the numeral 2.1 is a Sending module. 2.2 is a timer. 2.3 is the Response module. 2.4 is a slave clock. 2.5 is a network interface card (NIC) plugged into server machine.

The function of the Sending module 2.1 is retrieving data from storage media, assembling packets and sending packets to designated client. The operations of the Sending module 2.1 and Timer 2.2 are shown in Figure 3 and described as follows. The Sending module listens to a command to be issued by the Response module 2.3. When it receives the command, it configures the packet size and adjusts transmission rate based on the parameters included in the command. The maximum size of the packet is defined by application. It then passes the packet to NIC 2.5. Timer 2.2 counts the time elapsed since the last packet was sent to the client. If the elapsed time exceeds an application specifying threshold, it notifies application. Application can either stop the Response module 2.3 from waiting or resend the packet to the client. Timer 2.2 is reset when Sending module 2.1 sends out a new packet.

The operation of the Response module 2.3 is shown in Figure 4 and described as follows. Response module 2.3 polls for a REQ message. When it receives such a message, it adjusts the slave clock 2.4 as "Synchronized" to the client. It further retrieves the information from that message. Basing on that information, it optimizes the size and transmission rate of the next packet. Methods to compute the size and the transmission rate are presented in Embodiment 4. The Response module 2.3 then issues a command including these parameters to the Sending module 2.1. The command



enables the Sending module 2.1 to configure a new packet and adjust the transmission rate with the two specified parameters. When the Sending module 2.1 sends out the requested packet, the slave clock is set to "Not Synchronized".

The NIC 2.5 consists of hardware device plus device driver and transport application programming interface (API) software. The NIC accepts packets from the Sending module 2.1, segments and packs them into multiple network packets. It then puts them on the wire, transmits to the client through forward channel 1.3. NIC 2.5 also serves to receive packets from backward channel 1.4, reassemble a REQ message and pass the message to Response module 2.3. Both forward and backward channels are set up, maintained and managed by NIC 2.5.

The third embodiment of this invention is described below with reference to Figure 5 and 6. In Figure 5, the numeral 5.1 is a NIC. Its function is same as that of NIC 2.5 in embodiment 2. The numeral 5.2 is a receiving module. 5.3 is a buffer. 5.4 is a generic data processing entity. The numeral 5.5 is a Threshold pointer. 5.6 is Current level pointer. These are the pointers associating to the buffer 5.3. The numeral 5.7 is the Request module. 5.8 is a buffer monitor located in the Request module 5.7. 5.9 is a master clock.

The operation of this embodiment is shown in Figure 6 and described as follows. NIC 5.1 receives incoming data packets through forward channel 1.3 and sends out REQ messages through backward channel 1.4. The receiving module 5.2 interrogates NIC 5.1 to find out incoming packets. If a packet is available, it retrieves and puts it into buffer 5.3. Buffer 5.3 is designed to hold received data and serves as a cache to data processing entity 5.4. Another functionality of the buffer 5.3 is to absorb jitter. Jitter tolerance is one of the factors that determine the size of buffer 5.3. The following formula decides the size of buffer.

$$\text{Buffer size} \geq \text{data_processing_rate} * (\text{jitter_tolerance} + \text{round_trip_delay}) \quad (1)$$

data_processing_rate is the maximum processing speed of module 5.4. Module 5.4 can be a video/audio decoder or other data consuming unit. Jitter is a characteristic of packet switching networks. *Jitter_tolerance* refers to how much jitter that can be absorbed by the client. It is an application requirement. *Round_trip_delay* is the time taken by a packet travelling from the client to the server and returning back. It is determined by the distance between server and client. A method is provided in embodiment 4 to compute the *Round_trip_delay*.

Threshold pointer 5.5 is pre-set by application. It is determined by the following formula.

$$\text{Threshold} = \text{round_trip_delay} * \text{data_processing_rate} \quad (2)$$

The data processing entity 5.4 transfers data out of buffer 5.3. Every time data is transferred out, the current level pointer 5.6 gets updated. Similarly, every time the receiving module 5.2 adds in a newly received packet, current level pointer gets updated.



Buffer monitor 5.8 checks current level pointer 5.6 against threshold 5.5 regularly. If current level pointer 5.6 falls below the threshold pointer 5.5, buffer monitor 5.8 triggers the Request module 5.7 to proceed the master clock 5.9. When the master clock 5.9 is proceeded, the Request module generates a REQ message. Information such as client Identifier, current buffer level, current time stamp and user data are inserted into the REQ message. The Request module 5.7 then sends it through backward channel 1.4 to the Response module 2.3 in embodiment 2.

The fourth embodiment of this invention is in connection to the REQ message used in the invented method with reference to Figure 7. REQ message originates from the Request module 5.7 in Figure 5 and destined for the Response module 2.3 in Figure 2. It serves as a synchronization signal to make the slave clock 2.4 at server end synchronize with the master clock 5.9 at client end. It consists of 5 fields. Each of the first 4 fields is 4 byte in length. The first field is client Id. It is for identifying the client who sent the REQ message. The second field contains information of unoccupied space in client's buffer 5.3 at the time when this REQ message is generated. The Request module 5.7 uses the following formula to set this field.

$$\text{unoccupied space size} = \text{buffer size} - \text{current level} \quad (3)$$

The first REQ message indicates the size of buffer 5.3. Knowing the buffer size, server can foresee current buffer level when next packet reaches client.

$$\begin{aligned} \text{Current level (when next packet reaches client)} \\ = \text{buffer size} - \text{unoccupied space size} - \text{round_trip_delay} * \text{data_processing_rate} \end{aligned} \quad (4)$$

So the amount of data, i.e. size of the packet to be sent is

$$\text{size of packet} \leq \text{buffer size} - \text{current level (when next packet reaches client)} \quad (5)$$

It can further estimate the remaining time before the client buffer becomes empty. Based on that time, the transmission rate is derived.

$$\text{transmission_rate} = \frac{\text{size_of_packet}}{\frac{\text{buffer_size} - \text{unoccupied_space}}{\text{data_processing_rate}} - \text{round_trip_delay}} \quad (6)$$

The third field is Time stamp. Assuming server and client follow same clock, the time stamp tells the time when the Request module 5.7 sent out this REQ.

round_trip_delay in formula (1), (2), (5) and (6) can be computed using the following formula

$$\text{round_trip_delay} = \frac{\text{time stamp(when REQ is received)} - \text{time stamp(in REQ)}}{\text{clock_frequency}} * 2 \quad (7)$$



The fourth field is User Data Length. It indicates the amount of user data attached after it.

The fifth field is User Data. Depending on field 4, it could be none. This is where client application can put their specific information to feed back to server.

2.7. Effects of the invention

The effect of this invention is to enhance data transfer reliability, optimise data transmission control and simplify client and server's synchronisation. Without this method, server has no knowledge on client's requirements and can not adjust packet size and transmission rate to the optimised values. It has to control data transfer precisely at a fixed rate. Even though, it can not guarantee that client can adapt to its rate. This invention introduces a request-respond method to allow client feed back information to sever so that client and server are synchronised with client's clock. Besides that, server can take advantage of those information to dynamically configure packet size and transmission rate to archive efficiency and eliminate network jitter impact. The request-respond approach guarantees that packets are sent and received in sequential order. The overhead of the invented method is negligible compared to conventional method such as TCP.

3. DESCRIPTION OF THE DRAWINGS

Figure 1 is a view of a client-server communication system described in the embodiment 1. The difference from its conventional counterpart is that it adds a backward channel for transferring REQ messages back to server.

Figure 2 is a view of the server system in the second embodiment. It depicts the construction of a server system in which the invented method is applied. It also shows the relationships among various components.

Figure 3 is a view of the Sending module's design. It depicts Sending module's flow of operations in embodiment 2.

Figure 4 is a view of the Response module's design. It depicts the operations of the Response module in embodiment 2.

Figure 5 is a view of the client system in the third embodiment. It depicts the construction of a client system in which the invented method is applied. It also shows the relationships among various components.

Figure 6 is a view of the Request module's design. It depicts Request module's flow of operations in embodiment 3.

Figure 7 is a view of the REQ message structure. REQ message carries information that enables server to configure and control data transmission dynamically. The message itself serves as a synchronization signal.



4. CLAIMS

- (1) A method for transferring data reliably at high speed and controlling data transfer in a client-server system comprising:

a software based Request module embedded in the client system for requesting data and acting as a master clock;

a software based Response module embedded in the server system for responding to client's request and acting as a slave clock;

a means to synchronise said master clock and said slave clock;

a unidirectional backward communication channel connecting Request module and Response module;

a request message being generated by Request module and transmission of said generated message to Response module

- (2) A method according to claim 1, wherein said Request module comprises:

a buffer monitor checking the current buffer level;

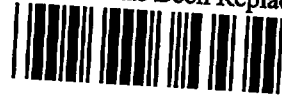
a master clock used to synchronise the client's reception with the server's transmission;

a means to proceed the master clock and to trigger Request module to request for data from server;

a means to generate said request message

- (3) A method according to claim 1, wherein said Response module comprises:

A slave clock used to keep the server synchronised with the client;



a means to adjust said slave clock in order to synchronise to said master clock

a means to interpret said request message in claim 1;

a means to compute size of a packet to be sent;

a means to compute transmission rate of packet;

a means to control and configure data transmission dynamically

(4) A method according to claim 1, wherein said request message comprises:

a means to provide information necessary for identifying client and configuring data transmission;

a means to provide synchronization signal to server

(5) A sending module provided under the control of said Response module comprises:

a means to poll for a command issued by said Response module;

a means to configure packet size and transmission rate dynamically and control data transmission;

a timer to count the time elapsed since the transmission of last packet to prevent server from halting due to long waiting for said request message