MFP RTP Design Documentation

By Craig McCown and Dawson Foushee

MFP RTP Description

Pipelined Design

MFP is a sliding window protocol, and a remaining buffer size is included with every packet sent. That way, each host can let the other know how much space it has to accept packets, and each can slide their window accordingly. The remaining buffer size is specified in the packet header and represents the number of bytes a host has left in its buffer.

Handling Lost Packets

To handle lost packets an acknowledgement message is sent as a response to every packet sent. If an acknowledgement is not received in a certain amount of time, calculated based on previously measured round-trip-times, a timeout occurs and the packet is resent. This is repeated until the packet is successfully acknowledged.

Handling Packet Corruption

Each packet includes an MD5 checksum of the packet's contents (the checksum field is set to 0 during calculation). The receiving host then calculates the same checksum and verifies that it matches the checksum in the packet header. If it does not, the packet is discarded and the sender must decide to resend the packet after a timeout.

Handling Duplicate Packets

When a duplicate packet is received, it is discarded, keeping the original packet. Acknowledgement packets are resent for every duplicate packet received. Each acknowledgement packet includes a number of times that the packet was received, which allows the sender to recalculate it's retransmit timer accordingly.

Handling Out-of-order Packets

Each packet contains a sequence number. That way, if a packet is received out of order, the receiving host can construct the message in the correct order using the sequence number, and send the corresponding acknowledgement to the sender. Sequence numbers are 0-indexed and represent a packet count.

Checksum Algorithms

MFP uses MD5 for calculating checksums. Both the header and body of the packet are hashed. Calculating the checksum on both the header and the body ensures data integrity of the entire packet.

Special Features

MFP has a header field called "frequency". This field indicates the number of times a packet has been sent or acknowledged. This allows for more accurate retransmit timer calculations. For example, if data is sent from a client to a server, and the server's acknowledgement of that packet is dropped, the client will timeout and retransmit the packet. The retransmitted packet will contain an incremented frequency value. Then, the server will acknowledge the

retransmitted packet, also with an incremented frequency value. When the client receives this acknowledgement, it can see that although only one acknowledgement was received, both the original transmission and the retransmission were received by the server. It can use this information to more accurately set the retransmit timer. (Explained in more detail below).

MFP RTP Header Structure/Header Fields

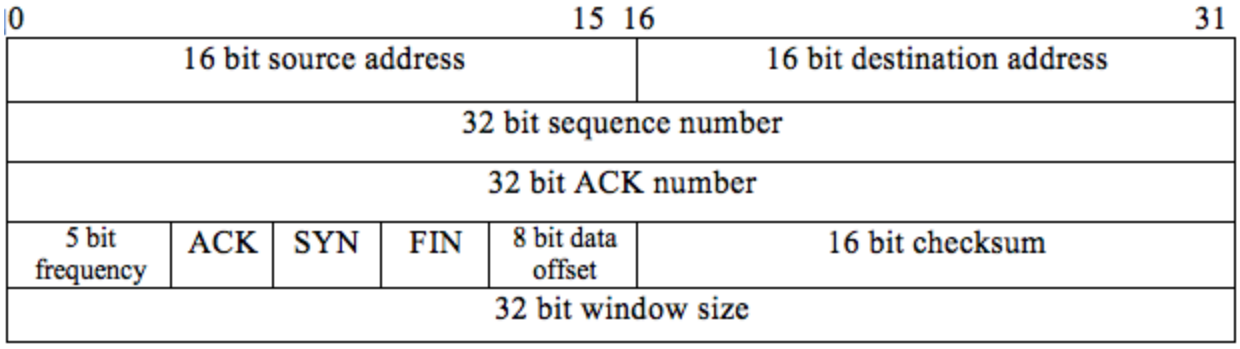

Figure 1. MFP Header structure.

Source Port: The port number of the sending application.
Destination Port: The port number of the receiving application.
Sequence Number: The sender's sequence number.
ACK Number: The sequence number of the next packet that the receiver is expecting.
Frequency: The number of times the packet has been received or acknowledged. The beginning value is 1.
ACK: A bit that signals an acknowledgement.
SYN: A bit that signals that a host is attempting to set up a connection.
FIN: A bit that signals that a host is attempting to close a connection.
Data Offset: The number of 32 bit words that an MFP implementer can add to the end of the header.
Checksum: The MD5 checksum of the packet's header and body. The checksum field is set to 0 during calculation.
Window Size: The number of bytes available in the sender's buffer. Used for flow control.

Finite State Machine Diagrams For End Points

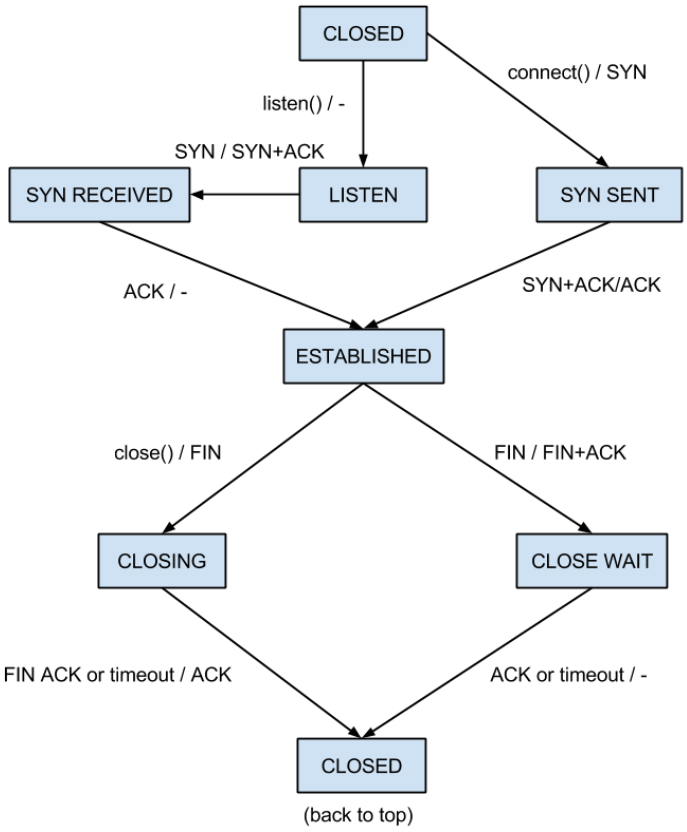

Figure 2. MFP Finite State Machine for Client and Server.

Both client and server begin in the CLOSED state. The server calls mf_listen(), and moves to the LISTEN state. The client calls the mf_connect() method, and sends a SYN to the server. When the server receives a SYN, it responds with a SYN + ACK and moves to the SYN RECEIVED state. When the client receives the SYN + ACK, it responds with an ACK, and moves to the ESTABLISHED state. Once the server receives the ACK it moves to the ESTABLISHED state as well.

To exit the ESTABLISHED state, either the client or server must call the mf_close() method. The closer then sends a FIN to the other host. The other host sends back a FIN + ACK. Then, the closer either receives that FIN + ACK or it times out. Either way, it sends an ACK to the other host. The closer moves back into the CLOSED state. The other host then either receives the ACK from the closer or times out and then moves into the CLOSED state as well.

MFP Programming Interface

mf_socket MFSocket(int window_size): Creates a new socket, which is an an object that exposes an API through which to implement an MFP application. Under the hood, it allocates system resources for possible future MFP communication including a window of corresponding size. All following methods are part of the MFSocket class.

void mf_assign(short port_number): Saves the port number for use later.

void mf_bind(short port_number): Associates (binds) a socket with a port number.

void mf_accept(): Accepts and establishes the next connection by receiving a SYN packet, sending an ACK + SYN packet, and then receiving an ACK packet. This call blocks until a connection is received.

void mf_connect(int ip_address, short port_number): Establishes a connection to a specified address and port by first sending a SYN, receiving a SYN + ACK, and then sending an ACK.

void mf_write(string data): Copies the argument data into the system's write buffer to be transmitted through the socket.

string mf_read(mf_socket socket): This method blocks while waiting for data at the socket, and reads the data that was received through the socket. If no data is received, the call continues to block.

void mf_close(mf_socket socket): Closes the connection by sending FIN packets and closing on both ends of the connection.

Algorithmic Descriptions

Packet Corruption Detection: Before sending any packet, the client takes an MD5 hash of the entire packet, including both the header and the body. During this calculation, all checksum bits in the header are zeroed out. After receiving any packet, the server takes an MD5 hash of the entire packet, again zeroing out the checksum bits. Then, the server compares the resulting hash with the hash in the received packet header. If the checksums match, the packet is received as usual. If they do not, the packet is discarded.

Retransmit Timer Calculation: The duration of time that a client must wait before timing out is calculated based on recorded round trip times and frequency numbers from the packet headers. The general algorithm is timer = threshold * previous RTT + (1 - threshold) * current RTT. In this way, each new RTT measurement affects the retransmit timer is always proportional. The suggested threshold value is .75, and the suggested initial previous RTT value is 6 seconds. Before a packet is sent, a timestamp is recorded. When that packet is acknowledged, the difference is also recorded. That difference is then divided by the frequency number on the acknowledgement packet. The resulting number is the new current RTT.