TCP Summary

Burton Rosenberg

March 1, 2004

TCP is a Transport Layer (layer 4) protocol for the reliable transmission of an unstructured stream of bytes. It uses IP as the Network Layer (layer 3) carrier. The TCP header includes source and destination port numbers, a sequence number, and acknowledgment sequence number, a set of flags, and a window size.

A TCP connection is distinguished by all of the following: the source IP address, the source port number, the destination IP address, the destination port number. In client-server computing, the destination IP address and port is a service offered by a particular server. The port number and IP address are discovered somehow, but in general the port number will be in the low end of the range of TCP ports. Most services site at *well-known ports*.

The client choses a random port number, usually from the high end of the TCP port range. These are called *ephemeral ports*. Generally, the port choice is made by the TCP implementation, as the client application does not care enough to suggest or force a specific port number.

Each byte of data is numbered with the sequence number, beginning not at zero, but at the *Initial Sequence Number* which is chosen randomly and independently at the establishment of the connection. Each the server and the client chose ISN's independently. For the data sent, the sequence number is set to indicate the last byte sent. Bytes are acknowledged on each TCP packet by setting the acknowledgment sequence number to indicate that all lower numbered sequence numbers have been received. Holes in the data might exist, but TCP does not ack data until all the lower numbered bytes have been received.

Among the flags in the TCP header is the ACK flag, set to indicate that the value in the acknowledgment field is valid. It is set for every packed but the first. TCP always acknowledge, but for the first packet there is nothing to acknowledge. Connections are established by packets with the SYN (synchronization) flag set. The server *passive opens* on a port and is waiting. The client *active opens* the connection by sending a TCP packet with SYN set and an ISP, let us denote by x, to the servers waiting port. The server returns a packet with both SYN and ACK flags set. It selects its own ISP y, and places $x + 1$ in the acknowledgment field. The establishment handshake is completed when the client responds with a pack with only the ACK flag set, and the ack field set to $y + 1$ (and sequence number set to x if there is no additional data in this packet).

There are a variety of closes, since each side can close the connection independently. To close the

side of the connection, the FIN flag is set. The peer acknowledges with the next sequence number. The peer can also set the FIN flag on return, to close its side. Or it can let its side remain open. There is also consideration for simultaneous close, where both side send FIN packets at the same time.

TCP uses a Window to achieve efficient transport of data. The window is *advertised* in the TCP header, and indicates that the advertiser is prepared to take that many bytes beyond the sequence number ack'ed by this same packet. Typically the sender will have multiple packets outstanding, called *segments*, to fill the window. As ack's are received, either the window size remains the same and the window is slid forward, or the window is decreased by the number of bytes ack'ed, thereby closing the window. The window can be closed to zero, if the receiver application is not consuming the data. When the application consumes data, an unsolicited ack can be send with a non-zero window size to open the window again.

That is the basic operation of TCP. For it to work there must be estimates of the round-trip time so that retransmission timers can be set to resend lost data. Also, there must be a method to deal with Internet congestion, so that sender and receiver detect and slow communication on congestion. Finally, there are mechanisms for advancing the window so that efficient, large segments are sent in preference to small segments, but without affecting the interactivity of the communications.

The Round-Trip timers are set by sampling both the mean and variance of round-trip time of segments sent and acknowledged without loss. The timeout is then calculated as a multiplication of the round-trip estimate by β set by the variance and by a method called *Karn's algorithm*. As packets are lost, β is increased exponentially and these lost packets are not considered in the estimation algorithms for Round-trip time When packets are successfully received, β is returned to its base value.

Congestion is controlled using a congestion window, smaller than the advertised window. As packets are lost, the congestion window is narrowed exponentially. As packets are received without lost, the window is opened by increments of one. When a connection is established, the congestion window is set to one and each successful packets opens the congestion window by one until it reaches the advertised window. This is called *slow start*. These two methods control congestion.

To prevent small segments, the window is not moved until this is significant new room in the buffer. Both sender and receiver participate. The sender uses *Nagel's algorithm* which states that data is not sent until the buffer is full or immediately upon getting an ACK packet. Sending when an ACK is received is important for interactive applications. The receiver simply delays opening the window until significant new room is available. It also attempts to delay its ack's, hoping to ack mutliple segments.