Chapter 2 LITERATURE REVIEW

This chapter will introduce TCP technology. In addition, this chapter will also discuss TCP congestion control and its algorithm, and various types of TCP version.

2.1 An introduction to TCP

TCP is a reliable connection-oriented protocol. It needs to establish a virtual path before data transmission. Three-way handshaking is used to set-up the connection. A series of parameters are exchanged while the connection is set-up.

After successfully establishing the connection, the sender sends data to the receiver who then replies with on acknowledgment, and vice versa. It is a duplex transmission, with data transfers in byte-stream format.

When the sender has finished transferring data, connection termination is performed with a four-way handshake.

2.1.1 TCP congestion control

TCP is a closed-loop control. i.e. it relies on feedback information to regulate the sending rate. There are two types of feedback: implicit feedback and explicit feedback. In implicit feedback, the sender uses timeout to determine if congestions have occurred in the network. In explicit feedback, the sender receives an explicit message (duplicate acknowledgement) when congestion has occurred in the network [LEON].

There are four types of congestion control algorithms: Slow Start, Congestion Avoidance, Fast Retransmission and Fast Recovery [RFC2581]. These four algorithms work in pairs. Slow Start and Congestion Avoidance is one pair, whereas Fast Retransmission and Fast Recovery is the other pair.

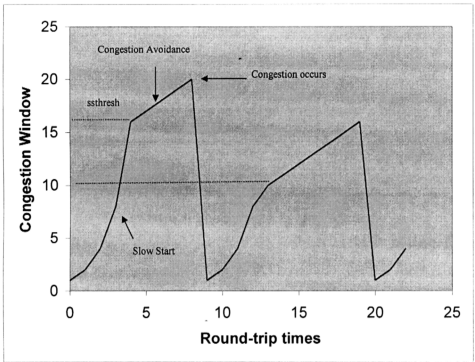

Figure 2.1 TCP congestion window

2.1.1.1 Slow Start and Congestion Avoidance

In TCP flow control, a sliding window mechanism is applied. In the receiver, there is a buffer for receiving incoming data. It is called the Receiver Window. Its acronym is ***rwnd***. rwnd will reply an acknowledgement (ACK) when it successfully receives the data from the sender. rwnd will notify the amount of bytes the sender can send through the ACK as well. This action is to make sure that the rwnd does not overflow. In this way, this TCP window mechanism is used to control traffic congestion.

For older versions of TCP flow control, after making a connection, the sender transmits multiple segments up to the size of the rwnd. When the connection is in the

same LAN, occurrence of severe traffic congestion is not so obvious. However if there are several intermediate routers and either the sender has a faster transmission rate and router has a lower receiving rate; or the preceding router has a faster transmission rate and succeeding router has a lower receiving rate; or the preceding router has a faster transmission rate and receiver has a lower receiving rate, congestion will occur. The receiver's buffer will always be full causing an overflow. The overflow segments will be thus discarded.

In Slow Start mechanisms, another window is added to the sender's TCP, called a congestion window. Its acronym is cwnd. cwnd is the sender's buffer. It is used to buffer the outgoing data. It initializes cwnd with one segment size. The default segment size is typically 512 or 536 bytes. When receiving each ACK, cwnd is increased by one segment. For example, let say a sender sends one segment and then receives one ACK. The cwnd will be increased by one segment. In the next round, the sender will send two segments and then receive two ACKs. On the third round, four segments will be sent and then four ACKs received. On the forth round, eight segments will be sent and so on. The sending size is increasing exponentially. It makes sure that the traffic is fully utilized. Figure 2.1 is a TCP congestion window diagram.

A Slow Start Threshold (*ssthresh*) is used to switch the Slow Start phase to another phase, called Congestion Avoidance. When the ssthresh size is less than the cwnd value, it is in Slow Start, otherwise it is in Congestion Avoidance. If the ssthresh size is equal to the cwnd value either one can be selected.

To avoid an over-load of the traffic, the ssthresh is used. Congestion Avoidance is used to slow down the sender from continually injecting multiple segments into the network. In Congestion Avoidance, the sender sends segment in linear increments. When it receives an ACK, the cwnd is increased by one segment per Round Trip Time (RTT) regardless of how many ACKs are received. One common formula used to update the cwnd during Congestion Avoidance is given below:

cwnd += SMSS * SMSS / cwnd

For a connection in which the receiver acknowledges every data segment, the above equation proves slightly more aggressive than one segment per RTT [RFC2581].

2.1.1.2 Fast Retransmit and Fast Recovery

In traditional TCP implementations, implicit feedback was used. In such an implementation, when a loss segment is lost, the sender will be in an idle state until the retransmission timer expires. Only after the timer timeouts will the sender retransmit the lost segment. This is because there is no feedback or acknowledgement to tell the sender that the segment is lost. Thus, the sender could not retransmit the lost segment. It creates an inefficient traffic utilization environment if a few segments are lost. In order to overcome this problem, fast retransmit and fast recovery [RFC2581] are used to quickly recover the lost segments.

In current TCP implementations, when the sender receives three duplicate acknowledgments in a row from receiver and notices that there is a lost segment, it will retransmit without requiring the retransmission timer to expire. The **ssthresh** will then be set to a maximum of half of the current cwnd or two segments. (Figure 2.1 is congestion window diagram.) This is to make sure that Slow Start is not performed in a large scale. ssthresh is the switch between Slow Start and Congestion Avoidance. The missing segment will be retransmitted and the cwnd updated to ssthresh plus three segments size. This inflates the congestion window by the number of segments that have left the network and which the other end has cached. [RFC2581].

There is a chance that the new data (retransmit packet after receiving the third duplicate acknowledgement) [RFC2581] is lost or the duplicate acknowledgement (dupACK) is sent before the new data arrives at the receiver. Therefore another dupACK will arrive. Each time another dupACK arrives; the cwnd will increase by one segment size. This inflates the congestion window for an additional segment that has left the network. A segment will be transmitted if allowed by the new value of cwnd [RFC2581].

When the receiver succeeds in receiving new data, it will send an ACK. When the next ACK arrives at the sender, the cwnd will update to the ssthresh. This ACK should acknowledge all the intermediate segments sent between the lost segment and the receipt of the first dupACK [RFC2581].

Fast Retransmit allows retransmission without retransmission timer timeout. It applies Congestion Avoidance algorithm, whereas Fast Recovery allows high throughput under moderate congestion especially for large windows.

2.1.2 TCP Terminology

2.1.2.1 Lost Segment

A lost segment is defined as a segment, which is discarded, caused by full buffers either in the intermediate routers or the receiver. Normally, an error segment is also considered as a lost segment. However, in current wired network, the possibility of an error segment is less than one percent in a series of transmitting segment.

2.1.2.2 Duplicate Acknowledgement

When a receiver receives an out-of-order segment, it will notify the sender. The receiver will send a duplicate acknowledgement to the sender. The receiver will also send a duplicate acknowledgement when there is a lost segment.

A duplicate acknowledgement packet consists of the segment number sequence the receiver expects.

2.1.2.3 Slow Start Threshold

There is a threshold called the Slow Start Threshold. Its acronym is ssthresh. It is a variable. Initially, its size is the minimum of cwnd and rwnd as shown by the equation below [RFC2001]:

ssthresh = min (cwnd, rwnd)

It is to make sure that the TCP output routine never sends more than this value.

When the TCP sender detects segment loss using the retransmission timer or when the third duplicate ACK is received, the value of ssthresh must not be set more than the maximum of Flight size /2 and two SMSS as per the equation below [RFC2581]:

ssthresh = max (FlightSize /2, 2 * SMSS)

FlightSize is the amount of outstanding data in the network.
It is to make sure that when the retransmission timer expires, ssthresh is always at least two SMSS.

The Slow Start Threshold is used to the switch Slow Start phase to another phase, called Congestion Avoidance. When the ssthresh size is less than the cwnd value, it is in Slow Start, otherwise it is in Congestion Avoidance. If the ssthresh size is equal to the cwnd value either one can be select.

2.1.2.4 Retransmission Time-out (RTO)

There are two methods for estimating RTO for TCP.

(i) First method
This is the old method for estimating the time-out RTO as a fixed multiple of Round Trip Time (*RTT*). The equation as below:

$$RTO = \beta \, RTT$$

where $\beta = 2$ (for example)

. β is a constant value and does not take into account the delay variance [LEON].

(ii) Second method

Jacobson [LEON], in [JACOBSON98] proposed an efficient implementation that keeps track of the delay variance. It is an algorithm that estimates a standard deviation using a mean deviation. The equation is as below:

$$DEV \leftarrow \alpha \, DEV + (1 - \alpha)|RTT - M|$$

Where DEV is smoothed mean deviation and

α is typically set to 3/4

When the mean deviation of the round-trip time is found, RTO is set as per the below equation:

$$RTO \leftarrow RTT + 4 \, DEV$$

2.1.3 TCP version of Congestion Control Algorithms

There are four types of TCP congestion control algorithm. For example, Tahoe, Reno and New Reno are types of TCP version. They apply their individual congestion control algorithm.

2.1.3.1 Introduction to Tahoe

Tahoe performs the Slow Start and Congestion Avoidance algorithm as well as the Fast Retransmit algorithm. In Fast Retransmit, after receiving three duplicate acknowledgements for the same segment, the sender assumes the segment is lost and performs retransmission for that particular lost segment without waiting for the retransmission timer to expire. The count of three for the duplicate acknowledgement is the threshold for duplicate acknowledgement. It means that Tahoe performs congestion control more effectively only when there is one loss segment occurred.

2.1.3.2 Introduction to Reno

Reno [RFC2581] performs the Slow Start and Congestion Avoidance algorithm as well as the Fast Retransmit and Fast Recovery algorithm. Compared to Tahoe, Reno has maintained the Slow Start and Congestion Avoidance algorithm but has modified the Fast Retransmit operation and includes the Fast Recovery algorithm. In Fast Retransmit and Fast Recovery, after receiving three duplicate acknowledgements, it retransmits the particular lost segment. In the meantime its congestion window is set to one half of the current congestion window and the sender transmit one segment. If it receives an additional duplicate acknowledgement, the sender transmits one segment for each additional duplicate acknowledgment received until the traffic is fully utilized.

In Reno, there is a number for duplicate acknowledgement or **ndup** used. ndup maintains at 0 until the sender receives three duplicate acknowledgements the ndup will then track the number of duplicate acknowledgements. These duplicate acknowledgements indicate that the particular lost segment has not yet been received by the receiver. After the receiver receives the particular lost segment and the sender receives the acknowledgement, the sender exits Fast Recovery and resets ndup to 0.

As compared to Tahoe, Reno will still performs congestion control effectively when there are three consecutive loss segments occurred.

2.1.3.3 Introduction to New Reno

NewReno performs the Slow Start and Congestion Avoidance algorithm as well as the Fast Retransmit and Fast Recovery algorithm. Whatever is performed in Reno for Fast Retransmit and Fast Recovery it also performed in New Reno but New Reno further applies a "recover" variable.

For "recover" variable, we use this variable-name as send_high. This variable is used to capture the last segment loss sequence number. When receiving the third duplicate acknowledgement, send_high equals the highest sequence number or, when retransmission timeout occurs, send_high is set to the highest sequence number.

When receiving an acknowledgement for new data (acknowledgement for a retransmitted packet after receiving the third duplicate acknowledgement), New Reno will proceed differently in two different scenarios.

In the first scenario, it will acknowledge all of the data up to and including "recover" which means that all the lost segments have been safely received by the receiver and the acknowledgements have also been safely received by the sender. The sender would either update the cwnd to equal the minimum of ssthresh and Flightsize plus MSS, or update the cwnd to equal the ssthresh, then exit the Fast Recovery procedure.

In the second scenario, it will not acknowledge all of the data up to and including "recover" (partial acknowledgement). Only a portion of the lost segment, not all, will be acknowledged. This is called partial acknowledgement. The sender will increase the cwnd by one MSS and reset the retransmit timer when the first partial acknowledgement arrives and every subsequent partial acknowledgement.

The sender will then exit the Fast Recovery procedure either acknowledging all of the data up to and including "recover" or a retransmit timeout.

During Fast Recovery, the sender sends two segments for each duplicate acknowledgement received. This is to fully utilize the traffic.

It means that New Reno performs better congestion control as compared to Reno when there are three consecutive loss segment occurred.

2.1.3.4 Introduction of Selective Acknowledgement

Fast Retransmit and Fast Recovery works well when lost segments are isolated events. However, when there are several losses occuring within a short period of time, the performance of Fast Retransmit and Fast Recovery degrades. Selective Acknowledgement (SACK) [S.FLOYD] works better in cases of multiple segment losses. An acknowledgement from a SACK TCP contains additional information about the segments received at the destination. When duplicate SACKs are received, the sender can then reconstruct information about the segments that were not received.

When three duplicate SACK are received by the sender, SACK enters Fast Recovery, where the sender will retransmit the first lost segment and reduces the cwnd by half. For each additional duplicate acknowledgement received the cwnd increases. From that point onward, whenever the sender is allowed to send another segment, it will use the SACK information to retransmit any lost segments before sending any new segments. Therefore, the sender can recover from multiple segment losses in one Round Trip Time.

The sender exits Fast Recovery when all the outstanding new data, which were sent when Fast Recovery was entered into, are acknowledged.

2.1.3.5 Summary of TCP version

NewReno has handled well in Fast Retransmit and Fast Recovery compared to Tahoe and Reno. It means that NewReno intends to have a better thoughput than Tahoe and Reno.

SACK has better performance if the lost segments are continuous and not isolated as it has better control when the system enters Fast Retransmit and Recovery.