RFC 5681 – Summary

**Introduction:**

RFC 5681 specifies about four of the congestion control algorithms. Those are fast recovery, fast retransmit, congestion avoidance and slow start. Apart from explaining about the above-mentioned algorithms, this RFC also specifies about how the TCP connection should react after a long ideal time along with the clarification of the issues related to TCP ACK generation.

**Congestion Control Algorithms:**

There are scenarios where the TCP sender tends to be conservative than required. This can be beneficial considering the scenario, but the TCP sender should never be aggressive than the allowable limit by the congestion control algorithms. This is obvious because when the TCP sender tends to be conservative, though it might not make use of the total available capacity, it still does the job without any data loss. But when it tries to be more aggressive, there are very good changes of data loss since the TCP sender is trying to send more data than the available bandwidth for the data transfer. Also, the congestion in network is identified through the loss of data in the below mentioned algorithms.

**Slow Start and Congestion Avoidance:**

The TCP sender uses these two congestion control algorithms to curb the data which is sent to network. This is implemented by adding two variables to the TCP for ever connection state. The amount of data which will be sent through the network will be governed by the value which is the minimum of rwnd and cwnd. This is the first variable. Sender side limit is Congestion Window (cwnd) and receiver side limit is Receivers Advertised Window (rwnd). The second variable – Slow Start Threshold (ssthresh) is used to choose between one of these two algorithms.

When a new transmission is beginning, TCP uses a slow start algorithm to release the data into the network. The reason behind this is that the network conditions are unknown, and the TCP doesn’t want to send huge chunks of data and congest the network. Hence this algorithm can be used to calculate the available capacity of the network. This algorithm is also used to start the “ACK clock”. The upper bound of the cwnd`s Initial Window (IW) must be set based on the value of SMSS. The ssthresh`s initial value is set to the maximum possible announced window. But this value should reduce according to the congestion response from the data connection. If cwnd < ssthresh then slow start algorithm is implemented and if cwnd > ssthresh then congestion avoidance algorithm is implemented. If both the values are same, then the sender can choose any one of the two algorithms.

If the slow start algorithm is used, then the value of cwnd is incremented by TCP by atmost by SMSS bytes. This increment of cwnd happens whenever an acknowledgment is received. When ssthresh >= cwnd or when a congestion is occurred in the network, then this algorithm comes to an end. The recommendation here is that the TCP should increment cwnd based on the minimum value of N and SMSS. This also helps in handling the “ACK Division” which tries to inflate the cwnd by SMSS for multiple ACKs for single segment of TCP data. If the congestion avoidance is used, then the value of cwnd is incremented by one full sized segment per RTT. This algorithm runs till any congestion is observed in the network. The recommendation here to increment cwnd based on a number which is equal to the count of the number of bytes that have been acknowledged by ACKs for the new data. It is important to note that cwnd should never increase by more than SMSS bytes per RTT. Also, TCP may update the cwnd using the following: cwnd += SMSS\*SMSS/cwnd. This is applied whenever it receives an incoming ACK for the new data. Furthermore, when there is a loss detected using the transmission timer by the TCP sender and if it has not been resent, then the ssthresh at max could have a value of max (Flight Size / 2, 2\*SMSS). Also, if it has been resent, then the ssthresh is held constant. Whenever a timeout is occurred, cwnd value should be set in such a way that it is less than the loss window (LW). Hence, after resending the dropped data, TCP sender usually starts with the slow start – increases the window till it reaches ssthresh – then followed by congestion avoidance algorithm. Also, we should note that when the slow start algorithm is used after a timeout, there are a lot of duplicate acknowledgements.

**Fast Retransmit/Fast Recovery**

The TCP Sender sends a duplicate ACK when an out of order segment is received to inform the sender. The reason for the duplicate ACK are as follows. Firstly, they can be caused due to dropped segments. Secondly, it can be because of the re ordering of the data segment. And lastly, it can be due to the replication of the ACK or the data segment. Once TCP receiver receives the missing data, it should send an ACK immediately to the sender which helps the sender to recover quickly. The TCP sender makes use of this fast-retransmit algorithm to detect and repair the loss based on the duplicate incoming ACKs. This works in such a way that if it receives three duplicate ACKs, then TCP retransmits the missing data segment. The fast recovery algorithm takes over the fast-retransmit algorithm once the fast retransmit, transmits the missing data segment.

**Additional Considerations:**

**Restarting Idle Connections:**

After a long period of idle time, TCP congestion control algorithms tend to send inappropriate chunk of traffic through the network. This is because it cannot use ACK clock as all the ACKs would have drained. The recommendation here is that the TCP should use the slow start algorithm to restart the transmission after a long idle time. Also, this starts the ACK clock.

**Generating Acknowledgements:**

The following are the recommendation: Delayed ACKs should be used by the TCP receiver. When doing so, the TCP receiver should not delay excessively. Meaning, for every second full sized segment, there should be an ACK sent and the ACK must not be delayed for more than 500 ms waiting for the arrival of second full sized segment. Apart from these, acknowledgments should be sent immediately when the received segments are out of the order.

**Loss Recovery Mechanisms:**

The recommendation here is that the TCP implementors should employ some form of advanced loss recovery to handle the multiple loss in single window of data.

**Security Considerations:**

Currently on receiving an ACK, the size of the congestion window is increased based on a constant number. The recommendation here is that the size of the window should be increased based on the number of bytes newly acknowledged by the arriving ACK. On successful implementation of these recommendations, we could avoid the congestion collapse and preserve the network stability.