Flow control in TCP

The basic protocol of TCP is augmented by an algorithm for flow control. This algorithm, called Congestion Avoidance is invoked in two different situations; it has more than one description, each of them having more than one interpretation.

Congestion avoidance is invoked when one of two events is observed by a TCP receiver:

- A timer expires when a host is waiting for an acknowledgment. In this case it must be assumed that a segment was lost and must be retransmitted. This situation results in a scenario called Slow Start (watch this video).
- The host keeps receiving "negative" acknowledgments[a] hinting that the segments are not getting through. This case is called Fast Retransmit.

[a] In TCP negative acknowledgments take the form of duplicate acknowledgments, a slightly different concept.

Both cases imply the presence of congestion (the main reason for packet loss).

The obvious first step is to slow down the transmission rate (this is an egoistic act: maintaining the sending rate does not serve the sender). This is achieved by reducing the flow to a minimum (resending the lost segment—which is a must) followed by a cautious gradual increase in the transmission rate.

The purpose of Congestion Avoidance is to restart the flow of data while attempting to avoid recreating congestion.

Slow Start

When no feedback comes from the other end of a connection, the recovery timer will eventually expire. Its expiry suggests that one of two several events happened:

1. The other end is dead.
2. The connection is broken.
3. A segment was lost and a deadlock occurred (one side waiting for a segment, the other for an acknowledgment).

The first two cases have no cure; the deadlock in the third can be broken by retransmitting segments.

Fast retransmit

TCP does not have negative acknowledgments, so a duplicate ACK serves as one. When a duplicate acknowledgment arrives, the sender must guess whether it is a result of packets delivered out of order or the result of a lost packet.

The suggested approach is to assume that 3 consecutive duplicate acknowledgments imply a lost packet. Thus, the sender's TCP ignores the first two duplicate ACKs assuming they resulted from an out–of–order delivery; when a third arrives, the sender must act as if its timer timed out and immediately resend the presumably lost segment.

There are various interpretations of what needs to be sent: just the oldest segment or more segments not acknowledged so far. Note the receiver may (does not have to) keep the out–of–order segments in its buffer.

(Maximum) Segment Size

TCP maintains a very important constant called Maximum Segment Size (MSS) for each virtual circuit. The value of MSS represents the maximum size of the payload of a datagram handled by the underlying NL.

For example, in IP a datagram has a default maximum size of 576B, hence an IP–oriented **SS** would be 536 (20B TCP header and 20B IP header). Systems expecting to work over an Ethernet connection often adopt **SS** = 1460 (1460+20+20 = 1500B, the maximum Ethernet frame size).

TCP has the right to send segments no larger than **MSS**; most **TCP** implementations will not send segments smaller than **MSS** unless forced to do so. There are several reasons for that, including the Silly Window Syndrome.

Congestion Avoidance uses two standard TCP–specific variables that are measured in multiples of MSS.

Congestion Window (CongWin or cwnd) is the actual sliding window size (unless it exceeds the window size advertised by the other side). CongWin is initialised to 1 or 2 MSS.

Slow Start Threshold Size ssthresh. It is initialised to 64KB (2^{16} bytes).

Congestion Avoidance

The trick is to sense what is the bandwidth currently available for the circuit used. The basic assumption is that the rate at which new segments should be transmitted should match the rate at which acknowledgments are sent back by the other end[a].

The algorithm limits the number of outstanding segments (i.e. sent but not acknowledged) changing the limit based on observed traffic.

By definition, whenever a TCP output routine is called to send data, it sends $\min(\mathit{data\text{-}available}, CongWin, \mathcal{W})$ bytes of data.

The amount of available data is an external factor, and so is $\mathcal{W}$, the receiver's window size (a bound coming from the header of the latest receiver's segment); CongWin is a variable changing every time the output routine is called.

[a]This assumes that the load is symmetric, an assumption wrong in many applications, e.g.. video–on–demand.

Congestion Avoidance operates in two phases:

- Exponential growth when the limit doubles in every round.
- Linear growth when the limit increases by 1 MSS in every round.

When a virtual circuit is created, CongWin is initialised to MSS (seldom 2 MSS) and Congestion Avoidance enters its exponential phase.

Exponential growth

While in the exponential phase, TCP increases CongWin by MSS for every segment acknowledged.[a]

Thus, if an acknowledgment for k segments arrives in time, TCP may:

- Increase CongWin by k (RFC3390) if CongWin is below a threshold value ssthresh.
- Switch to linear growth phase if CongWin reached the threshold.

[a]If one assumes that segments are sent in rounds, every round in the exponential growth phase will be twice as large as the previous round.

Linear growth

In this phase, CongWin is increased by $MSS \times \frac{MSS}{CongWin}$ each time a segment is acknowledged[a] (RFC 2581).

CongWin is increased only if it is less than 64 KB, the maximum possible segment size.

[a] If one assumes that segments are sent in rounds, then a round of length k (k = CongWin) segments will be followed by a round of length $k + 1$ segments.

Slow Start

When a timeout occurs, ssthresh is set to CongWin/2 and CongWin is set to MSS.

The Congestion Avoidance algorithm starts in its exponential growth phase.

Fast recovery

When 3 duplicate acknowledgments arrive, two conclusions can be reached:

1. A segment was probably lost: the receiver keeps reporting that it did not receive it.
2. The network must be at least partially functional; otherwise the 3 ACKs would not get through.

The second conclusion implies that one should not resort to a full slow start. Hence, both CongWin and ssthresh are set to ssthresh/2.

The Congestion Avoidance algorithm starts in its linear growth phase.