ANC4 Tutorial 4: Sketched Solutions

1. Suppose the transmission time is *T* and the RTT, *R*.

   Suggestion: draw a time sequence diagram to illustrate the following argument.

   At *t*=0, *cwnd* is set to 1MSS. At *t=T*, after a segment is sent, 1MSS of data is unacknowledged and so transmission must stop (this is what *cwnd*=1 means).

   At *t=R+T*, the first ACK appears and so cwnd is increased to 2. Two segments can now be sent by *t= R+3T*. If all goes well they will be ACKed by *t=2R+3T*.

   As each ACK arrives *cwnd* is increased by 1 so at *t=2R+3T*, *cwnd* =4, at *t=3R+7T, cwnd*=8.

   In general at $t=nR+(n+2^{n-1})T$, *cwnd* is 2^n. However if *T<<R*, then $nR+(n+2^{n-1})T \approx nR$ for small values of *n* (remember that this process stops at sstresh). Thus at $t \approx nR$, $cwnd=2^n$, which establishes the result

2. Fast retransmit requires that three additional copies of an ACK be received before the mechanism is triggered. Each copy is sent by the receiver when another segment arrives from the sender which is not the required one. If the window size is less than 4 the sender will not be able to end out sufficient segments after the missing one to trigger the 3 ACK copies.

3. This is best seen from the time sequence diagram on the next page. In the scenario shown,

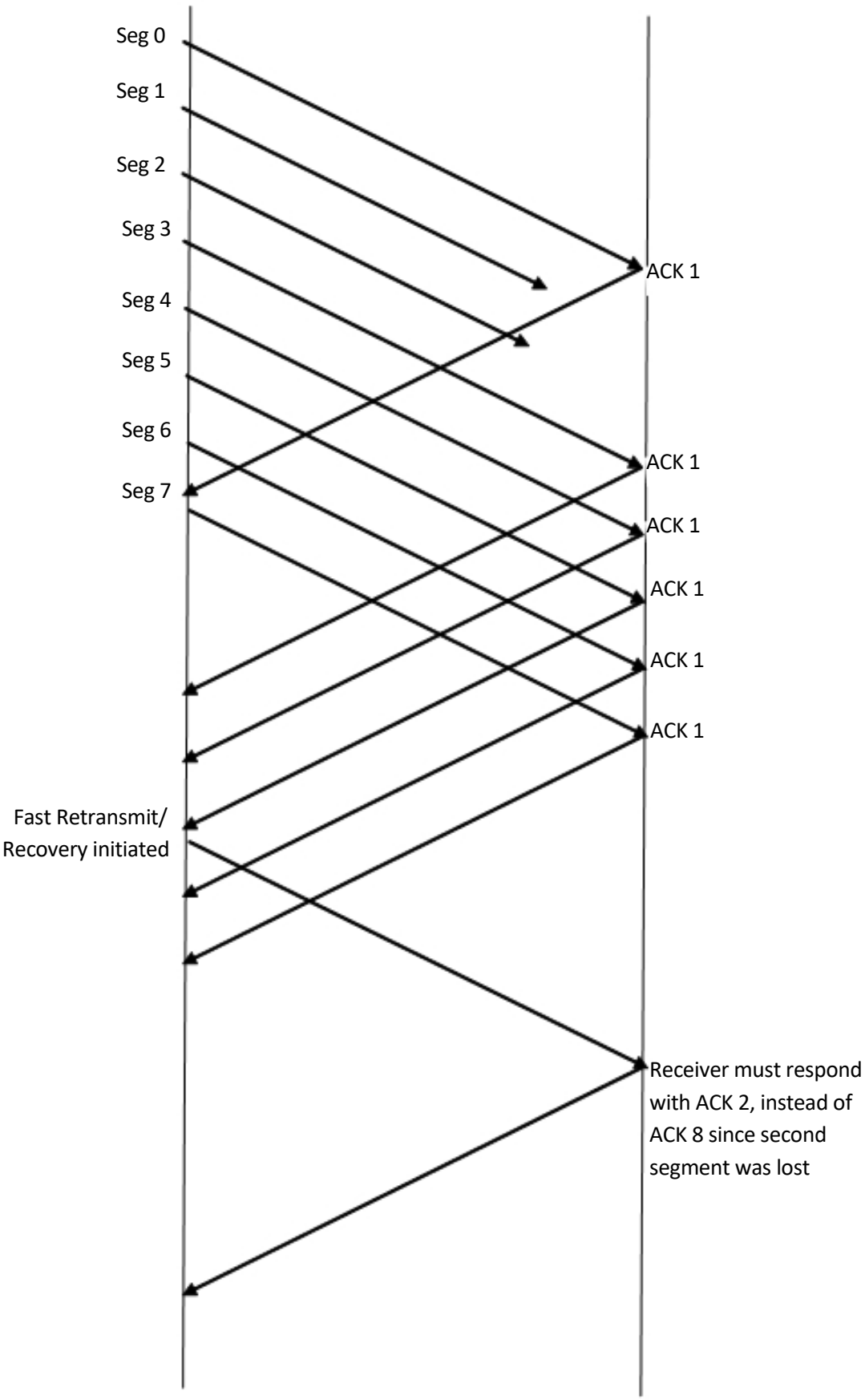
   Segs 1 and 2 are lost. The receiver continues to ACK Seg 0, requesting Seg 1 until finally after 4 such ACKs are received, Fast Retransmit and Fast Recovery are initiated. Normally this phase ends after the resent Seg 1 is ACKed (if all else was OK, in this case with ACK8) and that ACK received. However as Seg 2 is also missing, the receiver now starts ACKing with ACK2. However the sender is now out of windows and is unable to send enough further frames to trigger a fast-retransmit and (fast recovery). As a result the pipe empties until a timeout occurs and the whole connection resets to slow start.

4. The flows will receive an equal share if both want at least half. If, however, one flow requires less than half the bandwidth, the other is free to use more than half, should it wish to.

5. As soon as a segment is lost a Reno flow will cut its send rate in half. Once the link becomes saturated, segments will be discarded and (most likely) both flow send rates will be cut in half. If there is a difference, D, between the rates, this difference will also be halved. Assuming both senders are equally fast, the new difference will be maintained until the next congestion event at which point it will be halved again (D/4). Over several congestion occurrences D will eventually become close to zero and the two flows will be sharing the link equally.

6. In this scenario Base RTT will not be a proper reflection of the idle network RTT but will instead be a measure of a congested network. If the current throughput is a little worse than the initial congested throughput, the new Vegas connection will increase its congestion window, thus congesting the network further. If it worsens RTT enough (Diff > β) it will eventually back off but will continue to act to prevent the RTT falling below its initial congested value.

7. A TCP variant using loss-based congestion control will continue to ramp up send rate until loss occurs. Vegas on the other hand will start throttling back as soon as the buffer occupancy on the path increases beyond a minimal level. As a result, the loss-based TCP variant will gradually seize more and more of the available bandwidth.


Seg 0
Seg 1
Seg 2
Seg 3
Seg 4
Seg 5
Seg 6
Seg 7
ACK 1
ACK 1
ACK 1
ACK 1
ACK 1
ACK 1
Fast Retransmit/
Recovery initiated
Receiver must respond with ACK 2, instead of ACK 8 since second segment was lost

8. If RTT suddenly worsens as a new path is established around the failed router, Vegas (to which the router failure is invisible) will retain the old value of BaseRTT and will interpret all new values as well in excess of the upper β limit. As a result, it will gradually throttle its cwnd and slow its send rate down to a minimal level. However, it will never be able to converge the CurrentT to the ExpectedT and so the behaviour will continue.

9. The formula can be derived by finding the equation of the line relating the drop probability P_{DROP} to the average queue length, x, passing through (*MinTresh*, 0) and (*MaxTresh, MaxP*). This gives:

$$\begin{aligned} P_{DROP} &= 0 && x < MinTresh \\ P_{DROP} &= MaxP \times \frac{x - MinTresh}{MaxTresh - MinTresh} && MinTresh \leq x < MaxTresh \\ P_{DROP} &= 1 && x \geq MaxTresh \end{aligned}$$

10. In the ECN nonce system, when there is no congestion, the receiver is asked to echo the ECT pattern received (either 01 or 10). Suppose a receiver gets a message with CE set and decides to cheat. It must guess which ECT pattern was sent originally and has a 50% chance of being wrong (assuming the patterns are sent with equal probability). It is therefore unlikely to get away with cheating many times before being caught.

11. UDP conducts an optional checksum on the entire segment (+pseudo-header). However, as UDP is often used in circumstances where no action is feasible if an error in data is found, such comprehensive checking may be a waste of resources. Nonetheless a checksum on critical control information (e.g. the segment header) may still be useful. UDP offers no way of providing such a "lite" checksum without error checking all the data.

12. Main problem with Int Serv is that routers are required to store (soft) state for all flows passing through them. This would place a very large burden on an Internet router on, say, a backbone network and there are doubts about the practicality of this.