CS 168 Spring 2025

Transmission Control Protocol

Discussion 6

1 TCP in Action

Consider a sender sending 1000 B of data to a receiver over **TCP**. The sender sends packets of 100B, the window size is 300B, and the ISN is 99 (so **D100** is the first packet sent, then **D200**, and so on).

Remember, TCP uses a sliding window and retransmits the packet containing the next expected byte on a timeout or, if TCP Fast Retransmit is enabled, when three duplicate acks are received. Assume here that TCP Fast Retransmit is not enabled.

The link is **flaky**! The **initial** transmission of packets **D200 and D700** get dropped.

1.1 Fill in the below table with all packets sent by the sender until the receiver has received all packets and the sender knows that. For simplicity, assume that packets (data and ACKs) arrive in order. You may or may not need to fill in all lines.

	Packet Data Offset	**Sent on timeout**	**Dropped?**	**Cumulative ACK**
1	D100			A200
2	D200		X	
3	D300			A200
4	D400			A200
5	D200	X		A500
6	D500			A600
7	D600			A700
8	D700		X	
9	D800			A700
10	D900			A700
11	D700	X		A1000
12	D1000			A1100
13				
14				

$\boxed{1.2}$ If the RTT of the link is 10ms and the timeout is initially 3 seconds, what is the total time needed for the receiver to receive all packets and for the sender to know that?

Assume small packets (negligible transmission delay) and negligible processing time, and that the estimates that go into the Retransmission Timeout (RTO) remain constant during the events below.

2 * RTO +5 * RTT, why the first rtt for 1 counts?????

2 TCP Calculations

In this question, quantities will be measured in bytes unless **explicitly** mentioned otherwise.

2.1 Suppose two hosts are about to open a TCP connection. The TCP headers used in the communication are only 20 bytes long and regular (no-options) IPv4 is being used for Layer 3. If the MTU of the link is 1260 bytes, what is the MSS?

mss = 1260 -
20 - 20 = 1220

2.2 When this connection starts, the sender starts with an ISN 19. The initial window for the sender is set to 10 packets. Given the previously calculated MSS, what ACK does the sender receive as part of the TCP handshake? After that, what is the first and last ACK the sender receives for this initial window? (Assume no packets were lost or reordered).

ack 20. first: 1240, last: 12220

2.3 TCP senders compute their retransmission timeout (Estimated_Timeout, RTO) using exponentially-weighted moving averages of measured round-trip times (Estimated_RTT) and the deviations of those RTT measurements (Estimated_Deviation). Every time a new ACK is received, the sampled RTT of that specific ACK (Sampled_RTT) is used to perform an online update, by making the following three calculations **in order** (where α, β, and κ are fixed TCP parameters):

$$\text{Estimated\_Deviation} = (1-\beta) \times \text{Estimated\_Deviation} + \beta \times |\text{Estimated\_RTT} - \text{Sample\_RTT}|$$

$$\text{Estimated\_RTT} = (1-\alpha) \times \text{Estimated\_RTT} + \alpha \times \text{Sampled\_RTT}$$

$$\text{Estimated\_Timeout (RTO)} = \text{Estimated\_RTT} + \kappa \times \text{Estimated\_Deviation}$$

Now, suppose that we are a sender and have received a new ACK, and want to perform one step of RTO estimation. Let's do each of the three calculations above in the following suparts. Assume that connections have been open and that there are **no** dropped packets.

(a) The sender receives the current ACK and proceeds to update its various estimations. The time from when the packet was sent to when the ACK was received was **10msec**. If the previous **Estimated_Deviation** was **50msec**, and the previous **Estimated_RTT** was **70msec**, what is the new **Estimated_Deviation** (using **β = 0.5**)?

new = 0.5 * 50 + 0.5 * 60 = 55msec

(b) Still using the previous **Estimated_RTT** of **70msec**, What will the new **Estimated_RTT** be (using **α = 0.5**)?

new_rtt = 0.5 * 70 + 0.5 * 10 = 40

(c) Based on the previous two questions, what will the **ETO** be now (using $\kappa = 4$)?

= 40 + 4 * 55 = 260mesc

$\boxed{2.4}$ What is the maximum theoretical rate of data transfer for this window size if the **Estimated_RTT** is what was previously estimated?

rate = 12200 / 40 =
305KBps

$\boxed{2.5}$ Assume **40msec** is the **Estimated_RTT**. If the lowest bandwidth across this connection is **76.25 MBps**, what is the smallest window that optimizes the question?

= 76.25 *10 ^6 * 40msec = 3.05MB

3 Reliable Transport

$\boxed{3.1}$ Bob thinks that using ACKs on every packet is very wasteful and wants to design a transport protocol that is reliable but uses very few ACKs. He comes up with a scheme that he believes provides reliability but only sends at most $\log(n)$ ACKs, where n is the number of packets. The protocol is as follows:

- Let P_i be the $i^{\{th\}}$ packet. When the receiver sees **any** of the packets in the set:

  $\{P_{\{2^i\}} \mid 1 \leq i \leq \lfloor \log_2 n \rfloor\} \cup \{P_n\}$

  it sends back an ACK for that packet.
- The sender will send packets $\left(P_{\{2^{\{i-1\}}\}}, P_{\{2^i\}}\right]$ in order (i starts at 0), and wait for the last packet to be ACKed. If the sender does not hear back after some time, it sends this window of packets again until the last packet is ACKed. Then, i is incremented and the next window of packets is sent. One can think of this as a window that starts at packet 1 with size 1. Whenever the last packet in a window is ACKed, the window size is doubled, and the sender moves to the next set of packets. This process repeats until P_n is ACKed.

(a) Is this transport protocol reliable? Why or why not?

NO, it only ack for the last packet in the window

(b) If you said the protocol was not reliable, what is a modification that you could make in order to fix that? Try to make the smallest change possible.

change individual ack to accmulate ack

(c) Given that the protocol is reliable (or it wasn't and you apply your fix from the last part), does it actually save any bandwidth? Why or why not?

No. the window grows quickly and may congest the network

$\boxed{3.2}$ Alice thinks that Bob is onto something and wants to design a transport protocol that uses fewer ACKs. She decides that we can cut the number of ACKs in half by only acknowledging according to the following rules:

- Let $P_{\{2i\}}$ be the $i^{\{th\}}$ even packet. The receiver sends an ACK **iff** the receiver has received $P_{\{2i\}}$ and $P_{\{2i+1\}}$. Note that we count packet indexing at 0 for the first packet.
- The sender continues to send packets until all even packets have been ACKed.

(a) Is this transport protocol reliable? Why or why not?

No, if the last packet index is even, then the last packet will never be acked

(b) Is there a modification to the protocol that you could make in order to fix it?

let the sender to send a dummy packet when the packets number is odd.

(optional) Does this protocol actually reduce the number of ACKs sent? (Note: you won't be tested on this)

Yes.