CS 168 Fall 2024

Introduction to the Internet: Architecture and Protocols
Sylvia Ratnasamy, Rob Shakir, Peyrin Kao

Discussion 8: TCP

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. 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				

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.

   Solution: 2*RTO + 5 RTT = 6.05 s
   1*RTT for 1, 1*RTO before 5, 2*RTT for 5-7, 1*RTO before 11, 2*RTT for 11-12
   Note that we have 2*RTT for 5-7 – not 3*RTT. Because D500 and D600 are sent at practically the same time, it takes 1 RTT from when we send D500 until we receive A700.

2 TCP Calculations

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

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?

   Solution: 1220 bytes = 1260 bytes - 20 bytes (TCP) - 20 bytes (IP)

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).

   Solution: The ACK as part of the TCP handshake will be 20, because the receiver is ACKing having received the ISN. After that, the first ACK for real data sent by the sender will be 1240 which is the MSS (1220, calculated in the last problem) + the current sequence number, which is 20. The last last ACK will be 12220, which is the next sequence number after all data is sent, 12200 + ISN = 12200 + 20 = 12220.

3. In this part of the question we will determine the estimated RTT using some parameters of TCP. The following equations *may* be useful. Assume that connections have been open and that there are **no** dropped packets.

$$\begin{aligned} Estimated\_Timeout(ETO) &= Estimated\_RTT + 4 \cdot Estimated\_Deviation \\ Estimated\_Deviation &= \beta \cdot Estimated\_Deviation + (1-\beta) \cdot |Estimated\_RTT - Sample\_RTT| \\ Estimated\_RTT &= \alpha \cdot Estimated\_RTT + (1-\alpha) \cdot Sampled\_RTT \end{aligned}$$

   (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 $Estimated\_RTT$ was $70msec$, what will the new $Estimated\_RTT$ be (using $\alpha = 0.5$)?

   Solution: $Estimated\_RTT = 0.5 \cdot 70msec + 0.5 + 10msec = 40msec$

   (b) If the previous $Estimated\_Deviation$ was $50msec$, what is the new $Estimated\_Deviation$ (using $\beta = 0.5$)?

   Solution:

$$\begin{aligned} \text{Estimated Deviation} &= 0.5 \cdot \text{Estimated Deviation} + 0.5 \cdot |\text{Estimated RTT - Sample RTT}| \\ &= 0.5 \cdot 50msec + 0.5 \cdot (40msec - 10msec) \\ &= 40msec \end{aligned}$$

   (c) Based on the previous 2 questions, what will the ETO be now?

   Solution: $ETO = 40msec + 4 \cdot 40msec = 200msec$

4. What is the maximum theoretical rate of data transfer for this window size if the $Estimated\_RTT$ is what was previously estimated?

Solution:

$$Window\_Size(\text{bytes}) = Window\_Size(Packets) \cdot 1220\text{bytes} = 12200\text{bytes}$$
$$\text{Bandwidth} = \frac{12200\text{bytes}}{40msec} = 30500\frac{\text{bytes}}{sec} = 305KBps$$

5. Assume $40msec$ is the $Estimated\_RTT$. If the lowest bandwidth across this connection is $76.25MBps$, what is the smallest window that optimizes the question?

   Solution: $\frac{76.25\times10^6\text{bytes}}{sec} \times 40msec = 3.05MB$

3 Reliable Transport

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

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

  it sends back an ACK for that packet.

- The sender will send packets $(P_{2^{i-1}}, P_{2^i}]$ 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 sends the next window of packets. 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.

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

Solution: No, it is not reliable. If only the packets that are designated to be ACKed make it through, the receiver will ACK them and the sender will happily accept that all the data was received, even though only $log(n)$ packets actually made it through.

(2) 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.

Solution: Simply change the ACKs from individual ACKs to cumulative ACKs that are only sent if the target packet and all previous packets have been received.

(3) 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?

Solution: Not at all. In fact, this protocol uses much more bandwidth. As the window size increases (exponentially!), the odds that a packet is dropped is larger and larger, since there are more packets (more changes for drops) and the large number of packets sent at once will almost guarantee to congest the network if a large file is being sent. This means that the entire windows will be resent constantly, wasting lots of bandwidth.

As an example, consider the transfer of a 1 GB file. The last 500 MB of the file will be sent as a single window. Assume about 1 KB of data is sent per packet. Then the last window is 500,000 packets wide! If even one of these packets are lost, then all 500,000 will be resent until every packet has been received.

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 packets indexing at 0 for the first packet.
- The sender continues to send packets until all even packets have been ACKed.

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

Solution: No, it is not reliable. Consider sending an odd number of packets. The receiver will never acknowledge the last packet since the sender won't send another even packet.

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

Solution: Have the sender keep a count of the number of packets. If the count is odd send a dummy packet with no data.

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

Solution: Yes in expectation the number of transmissions for each paper in the even/odd scheme is always less than the number of transmissions in the 1 ACK per packet scheme, meaning the total number of ACKs sent is also always less.

Proof: Let X the number of transmissions by ACKing every packet.

Let Y the number of transmissions by ACKing every even packet.

Let p the probability that 1 packet is dropped

We assume geometric distributions to model the probability of a given number of drops before a packet is successfully transmitted.

When ACKing every packet, $P(\text{retry for packet } i) = p$. Therefore, X_i, the number of retries for packet i, is a geometric distribution.

When ACKing only even packets, $P(\text{retry for packet } i) = 1-(1-p)^2$ (the probability that either or both of the packets in the pair are dropped). Therefore, Y_i, the number of retries for packet i, is a geometric distribution.

Then $X = \sum_{i=0}^{n} X_i$ and $Y = \sum_{i=0}^{n} Y_i$.

$E[X] = \sum_{i=0}^{n} E[X_i] = n\frac{1}{p}$

$E[Y] = \sum_{i=0}^{n} E[Y_i] = \frac{n}{2}\frac{1}{1-(1-p)^2} = \frac{n}{2}\frac{1}{p(2-p)}$

Then, for all $p \leq 1$, $E[Y] < E[X]$