Homework4: TCP (Transmission Control Protocol) - Solutions

COM-208: Computer Networks

File transfer time calculation

End-host A sends end-host B a file of size $8 \cdot MSS$ bytes.
A and B are directly connected over a channel with transmission rate R bytes/sec and propagation delay $d_{prop} > \frac{4 \cdot MSS}{R}$ secs.
A and B use a modified version of TCP were:

i Congestion control and flow control are disabled.

ii Sender window size is fixed to $4MSS$ bytes.

iii Retransmission timeout is $T \gg RTT$ secs.

Assume that B stores (does not discard) all out-of-order segments.

How long will it take until B receives the entire file (i.e., B receives the last byte of the file) in each of the following scenarios?
Note: Assume that the packet headers (for all layers) have negligible size and that the connection is already established.

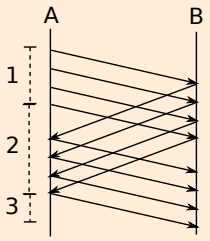
- No segments are lost.

The file transfer consists of three phases:

1 A Transmits 4 segments: $\frac{4 \cdot MSS}{R}$ secs.

2 A Waits for the ACK for the 4th segment: $2 \cdot d_{prop}$ secs.

3 A Transmits the 8th segment, and it arrives at B: $\frac{MSS}{R} + d_{prop}$ secs.



- The $5th$ segment is lost and Fast-Retransmit is disabled.

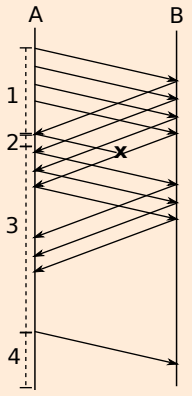
The file transfer consists of four phases:

1 A Transmits the first segment and waits for the ACK: $\frac{MSS}{R} + 2 \cdot d_{prop}$ secs.

2 A transmits the 5th segment: $\frac{MSS}{R}$ secs.

3 A Waits for the 5th segment to timeout: T secs.
(Here, we assume that the timer starts *after* A is done transmitting the 5th segment)

4 A Transmits the 5th segment, and it arrives at B: $\frac{MSS}{R} + d_{prop}$ secs.

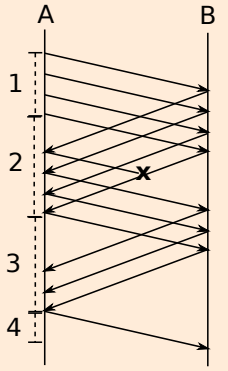


- The $5th$ segment is lost and Fast-Retransmit is enabled.

The file transfer consists of four phases:

1. A Transmits 4 segments: $\frac{4 \cdot MSS}{R}$ secs.
2. A Waits for the ACK for the 4th segment: $2 \cdot d_{prop}$ secs.

3. A Transmits the 8th segments and waits for the ACK: $\frac{MSS}{R} + 2 \cdot d_{prop}$ secs.
4. A Transmits the 5th segment, and it arrives at B: $\frac{MSS}{R} + d_{prop}$ secs.



File transfer over TCP

Consider the topology shown in Figure 1. Host A opens a TCP connection to host B, and starts sending a file of size $F = 10$ bytes, in segments of size $MSS = 1$byte each. As a result of a faulty link, the 5-th packet (without counting the SYN packet in the TCP handshake) transmitted by A is lost.

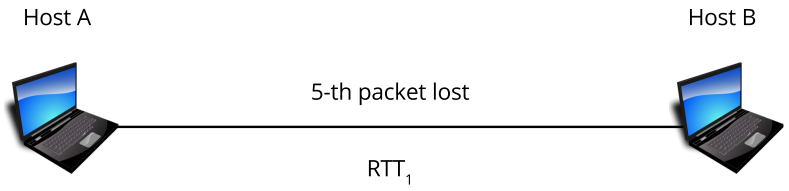


Figure 1: Network topology.

For the following questions, make the following assumptions:

i The transmission delay for packets is negligible.

ii The round-trip time between A and B is RTT_1.

iii The retransmission timer of host A has a fixed duration equal to $2 * RTT$.

iv TCP has Fast Retransmit disabled.

v A TCP receiver sends an ACK for each packet it receives.

vi The first segment that A transmits will have a sequence number of 1.

vii B stores (does not discard) all out-of-order packets.

- Complete the sequence diagram in Figure 3 with all packets exchanged between A and B *(we have completed part of the diagram to help you get started).*

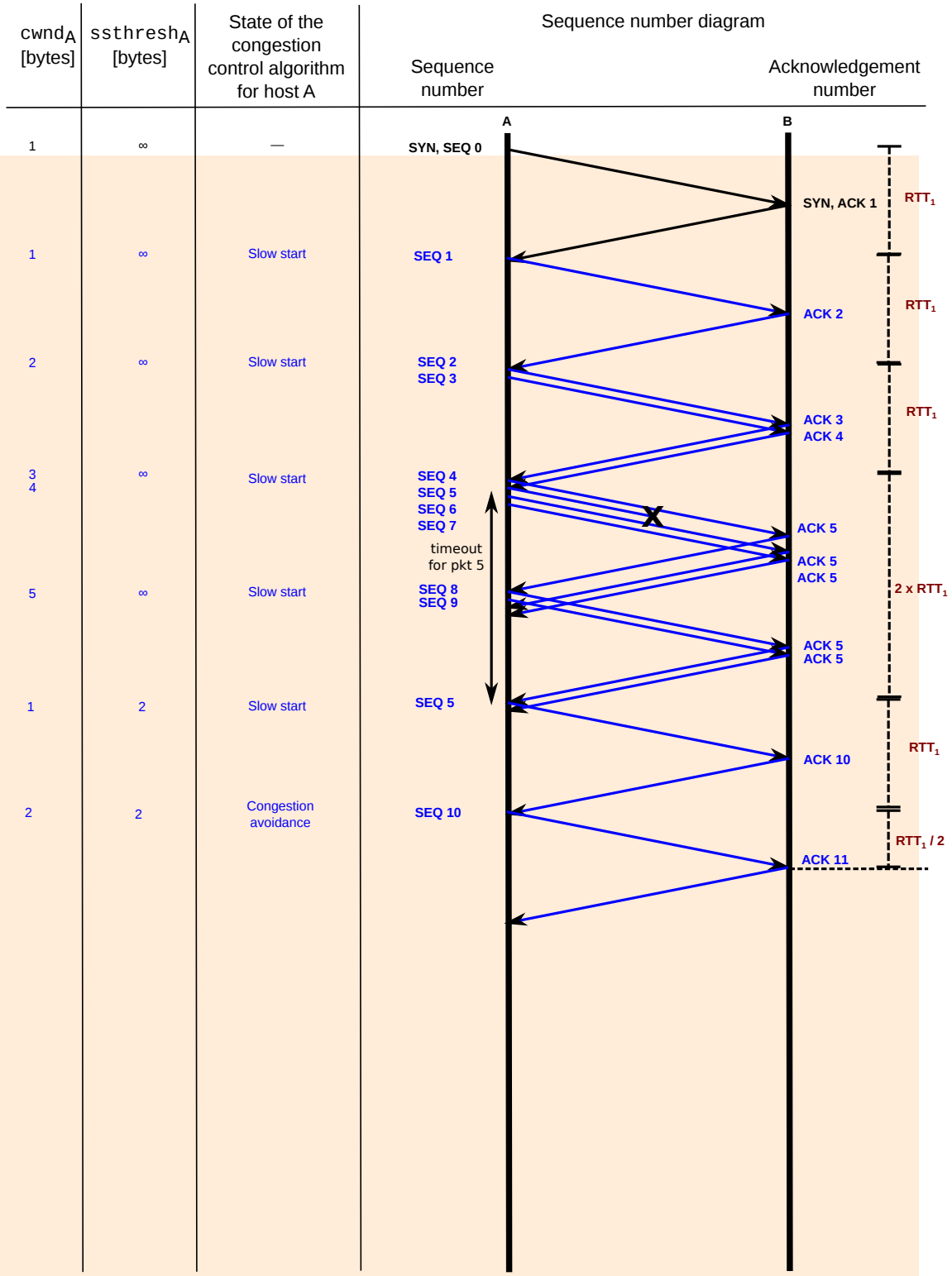


Figure 3: Sequence diagram of packets exchanged.

- Calculate how much time it takes for B to finish receiving the file.
  (Note: The one-way propagation delay from A to B is $\frac{RTT_1}{2}$)

> You can view the time duration marked down on the sequence diagram in Figure 3. With the connection setup time included, it takes $6.5 \times RTT_1$ time to complete the file transfer, from start to finish.

Now, assume A uses node P, which runs an application-layer proxy to transmit the file to B, as shown in Figure 4.

When P receives a connection request from A, it connects a TCP socket with B. After that, the proxy application receives data from the TCP socket connected to A (the input socket), and writes data out to the TCP socket connected to B (the output socket). P forwards these packets to the output socket, the moment it can read them from the input socket. The proxy's operations do not incur any processing delay.

P is located exactly in the middle of the path between A and B, such that the round-trip times between A and P, and between P and B are both equal to $RTT_2 = \frac{RTT_1}{2}$.

The faulty link described previously is now located on the part of the path between P and B (the second half of the path). As a result, the 5-th packet transmitted on that part of the path is lost. No packet loss occurs on the part of the path between A and P.

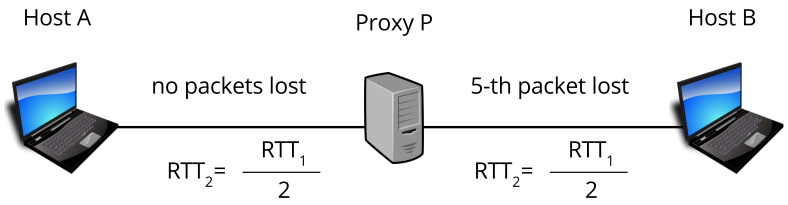


Figure 4: Network topology after adding a proxy.

- Calculate the time it takes for the file transfer to be completed in this new setting.
  (Note: Do not forget to adjust the timeout interval for the two TCP flows; from A to P, and from P to B. The timeout interval for the two flows is equal to $2 \times RTT_2 = RTT_1$)

> In this new setting, the transfer time can be computed as follow:
>
> First A will connect to P and send a TCP SYN. This will take $\frac{RTT_2}{2}$. As soon as P receives this SYN from A, it will initiate the handshake with B. The rest of the communication between A and P will be carried out in parallel with the communication between P and B. Hence, for the first half of the path, we only need to consider

the time it will take to propagate the first (connection) packet from A to P.

Now for the communication between P and B, the sequence diagram will be an exact replica of the one we created in figure 3. The only difference now is that the round-trip time between P and B is half the round-trip time between A and B. Hence, it will take $6.5 \times RTT_2$.

Thus, the total file transfer delay will be:

$$6.5 \times RTT_2 + \frac{RTT_2}{2} = 7 \times RTT_2 = \frac{7 \cdot RTT_1}{2}$$

- Does the introduction of the application-layer proxy in the previous part improve or worsen the file transfer? Which features of TCP are responsible for this?

  The main reason why the file transfer will take shorter to complete is the reduction of the end-to-end RTT for each flow. This reduction in RTT will have the following effects on TCP:

  - The congestion window will converge faster to an ideal value (assuming that link capacity is limited). In our example, where the capacity is unlimited, slow start will benefit even more.
  - This will make TCP more responsive in detecting and correcting channel error (e.g., the 5-th packet which was lost). This is because the timeout interval adjusts to the RTT estimate; a smaller RTT means that a packet is retransmitted faster. We would observe the same behavior in the case where fast-retransmit was also activated.

Now with routing

Consider the topology shown in Figure 5. End-host A sends end-host B a large file. This is the only active flow in the network and all the traffic traverses links L_1, L_2, L_3 and L_4. A sends application data into its TCP socket at a rate of 80 Mbps. B can read data from its TCP socket at a rate of 30 Mbps.

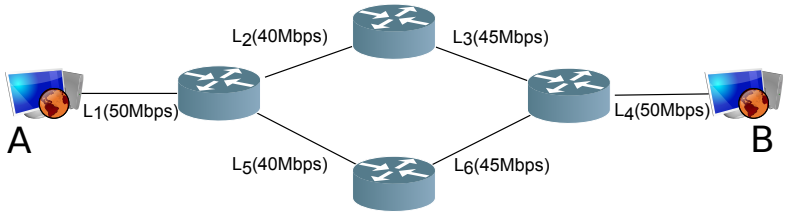


Figure 5: Network Topology with Links' Bandwidth Capacity.

- What is the max transfer rate for the TCP flow in the following scenarios? Which aspect of TCP limits it?
  - The TCP receive buffer at B can hold only a small portion of the file.

    Flow control will limit the transfer rate to 30 Mbps, because buffer space is limited. A will not send data faster than the rate at which B's application layer frees up TCP buffer space by reading the data.

  - The TCP receive buffer at B can hold the entire file.

    Congestion control will limit the transfer rate to 40 Mbps, because B has sufficiently large buffer space, which makes flow control irrelevant. The transfer rate is, thus, determined by link with the least capacity (L_2, in this case).

Now, Assume that the TCP receive buffer at B can hold only a small portion of the file. During the file transfer Link L_2 fails and the traffic between A and B is rerouted via links L_1, L_5, L_6 and L_4. The RTT between A and B increases from RTT_{orig} to $RTT_{new} = 5 \cdot RTT_{orig}$.

- Do A and B need to establish a new TCP connection? If yes, describe the message exchange.

There will be no handshake involved; the connection will be preserved. Routing will take care of the path change and nothing but the increased RTT will be visible at the transport layer. This is because it is only the end-hosts that maintain TCP connection state.

- Will the path change affect the TCP flow control?

  Flow control, again, only involves A and B. Thus, flow control will not be affected by the path change.

- What effect will the path change have on the TCP traffic between A and B?

  Initially, the retransmission timeout at A will be too small, given that the RTT has increased. This may cause premature timeouts and, as a result, unnecessary retransmissions. Nevertheless, A will continue collecting RTT samples for ACKed packets and, thus, it will readjust the retransmission timeout to a suitable value.

TCP fairness

Consider the topology shown in Figure 6. Links A-R, B-R and C-R have identical characteristics, and have a higher transmission rate than link R-D. Transmitting hosts A, B and C all have to share the same link to send data to host D.
(Note: Assume that TCP is fair, in the sense that all TCP connections sharing the same bottleneck link will equally share the link's bandwidth.)

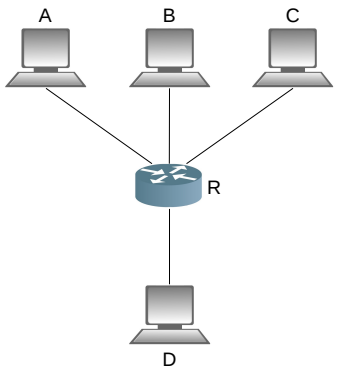


Figure 6: Network Topology

Calculate the share of the link capacity of R-D that each host gets (a long time after the connections have been established) in each of the following cases.
You should justify your answers.

- A, B and C are transmitting data. Each host uses a single TCP flow.

  After the TCP flows stabilize, they use the entire available transmission rate on the path from source to destination.

  The common link R-D is the bottleneck for each flow (its transmission rate is smaller than the transmission rate of any other link on the path). So all three TCP flows share the same bottleneck.

  Since TCP flows are fair to each other, and the paths they traverse have the same characteristics, this means that all flows will have equal throughput: $T_A = T_B = T_C = \frac{rate_{R-D}}{3}$.

- Only A and B are transmitting data. A uses only one TCP flow, and B uses two parallel TCP flows.

Given that TCP fairness works on a per-flow basis (and not on a per-host basis), this question is similar to the previous one. After the TCP flows stabilize they use up the entire available capacity on the bottleneck link, and share the capacity equally among them. Since there are three flows, the throughput of each flow becomes $\frac{rate_{R-D}}{3}$.

Finally, given that B uses two flows, the throughput for each host becomes $T_A = \frac{rate_{R-D}}{3}$ and $T_B = \frac{2 \times rate_{R-D}}{3}$. This means that $T_B = 2T_A$.

- A, B and C are transmitting data. A and B each use one TCP flow. C uses a UDP-based application which transmits data at a constant rate of 40% of the transmission rate of link R-D.

UDP does not adjust its transmission rate, even if there exists any other source of traffic in the network. Thus the throughput of host C is $T_C = 0.4 \times rate_{R-D}$.

Similar to the previous questions, the TCP flows are fair to one another, and each will get an equal portion of the remaining (60%) capacity of the bottleneck link. Thus, $T_A = T_B = 0.3 \times rate_{R-D}$.

Therefore, we have that $T_A = T_B = \frac{3}{4}T_C$.

Multiple TCP flows

Two flows, $flow_1$ (from A to C) and $flow_2$ (from B to C) traverse the same bottleneck link, R-C, as shown in Figure 7. The end-hosts use a variant of TCP where the congestion window is fixed to 60 packets (i.e., the window does not adapt to network conditions). You know that the capacity of link R-C is 3100 packets/sec.

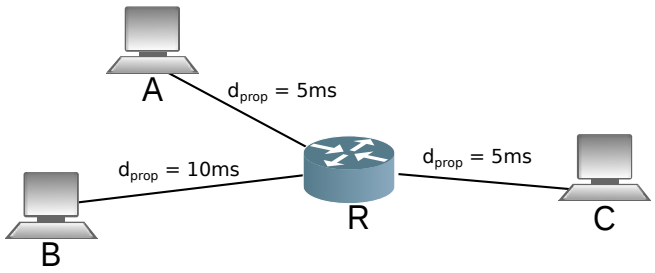


Figure 7: Network Topology.

- Suppose that the transmission delay is negligible. Compute the throughput that $flow_1$ ($Tput_1$) and $flow_2$ ($Tput_2$) would achieve if only one flow is active at a time, and no packet is ever dropped. Compute ratio $\frac{Tput_1}{Tput_2}$.

  Since the window size is fixed, each sender will transmit 60 packets per RTT seconds (where RTT is the two-way propagation delay) and, thus, $throughput = \frac{60}{RTT} = \frac{60}{d_{X->R}+d_{R->C}+d_{C->R}+d_{R->X}}$ (where X is A or B). According to this scheme we have that $Tput_1 = \frac{60}{0.020} = 3000$ pkts/sec and $Tput_2 = \frac{60}{0.030} = 2000$ pkts/sec. Both throughput values are smaller than the bottleneck link capacity and, thus, they are feasible. The ratio of the throughput values is $\frac{Tput_1}{Tput_2} = 1.5$.

- You run an `ns-2` simulation where both flows are active at the same time, and you get that $\frac{Tput_1}{Tput_2} = 1.28$. In your simulation, you have set the buffer size of router R to 1000 packets, so that no packet is ever dropped.

  Explain why the ratio between the throughput of the two flows is different from what you computed in the previous sub-question.

  The capacity of the bottleneck link (3100) is much smaller than $Tput_1 + Tput_2 = 5000$ pkts. Thus, (given that the router drops no packets) all packets forwarded will experience queueing delay. We can argue that both flows will experience equal queueing delay since both flows use the same window size. In effect the RTT

of each flow will increase by Q (the queueing delay), which means that $throughput = \frac{60}{Q+RTT}$. For a large enough value of Q, link propagation delay becomes insignificant, and $\frac{Tput_1}{Tput_2}$ goes closer to 1.

Thinking creatively about TCP

Suppose that each router in the network has infinite buffer space which can hold all packets the router has to forward, so that no packet is ever dropped. Given this setting:

- What happens when a network link becomes congested.

  Since no packet will ever get dropped, congestion will manifest in the form of queuing delays, which will increase the RTT between communicating hosts.

- Describe how individual TCP flows will behave.

  There are two mechanisms in TCP which are relevant here; congestion control and RTT estimation. The outcome will depend on how these two interact with each other. In the worst case, the sender's estimate of RTT will keep up with the increasing queuing delay and no timeout will occur (transmission timeout depends on RTT). This will cause the congestion window size (and queuing delay) to grow indefinitely.

  At first glance, this will not affect TCP throughput that much. However, new TCP flows will take longer to establish (the time duration of the TCP handshake is proportional to RTT), and will not be able to catch up with pre-existing flows, since nobody will back off (this will affect fairness). Finally, network congestion will affect latency-sensitive applications (e.g. video calls, DNS lookups) and will render the network pretty much unusable.

- Propose a modification for TCP, which will improve its behavior.

  The sender should base its congestion control algorithm on packet delay, as opposed to packet loss. This is something similar to what TCP Vegas (an existing flavor of TCP) does. Note that this modification might have some unwanted interference with the way that TCP estimates the RTT for a connection.

A variant of TCP

Consider a variant of TCP were flow control is disabled. Is the congestion control in this case sufficient to control the transmission rate of a sender if it is overwhelming a

receiver (i.e., when the receiver has no more buffer space)?
Justify your answers.

Congestion control will throttle the sender when the receiver becomes overwhelmed; the receiver will start dropping packets, which the sender will perceive as a congestion event. In particular, the sender will experience a transmission timeout and remain in the Slow Start state for as long as the receiver has no buffer space left.

Although congestion control can deal with the issue of an overwhelmed receiver to some degree, it will not be as efficient as flow control for the following reasons:

1. This will waste network bandwidth; packets traverse the entire network path before an overwhelmed receiver is able to drop them.
2. It will take some time for the sender's congestion window (and, possibly, their RTT estimate) to re-adapt to the network conditions. The congestion window will have to grow back to a size which reflects the true capacity of the network (as opposed to the capacity of the receiver). This can reduce performance if a receiver only processes requests in bursts.

To sum up, TCP will still work if we disabled flow control. However, this deprives the sender of one source of information which it can use to optimize its sending strategy. This makes receiver-related events (flow control) pollute the state that the sender maintains about network capacity (congestion control). As a result, the sender will overreact when a flow-control-related event occurs.

Reading congestion window plot

Consider the graph shown in Figure 8, which plots the window size of a TCP sender as a function of time.

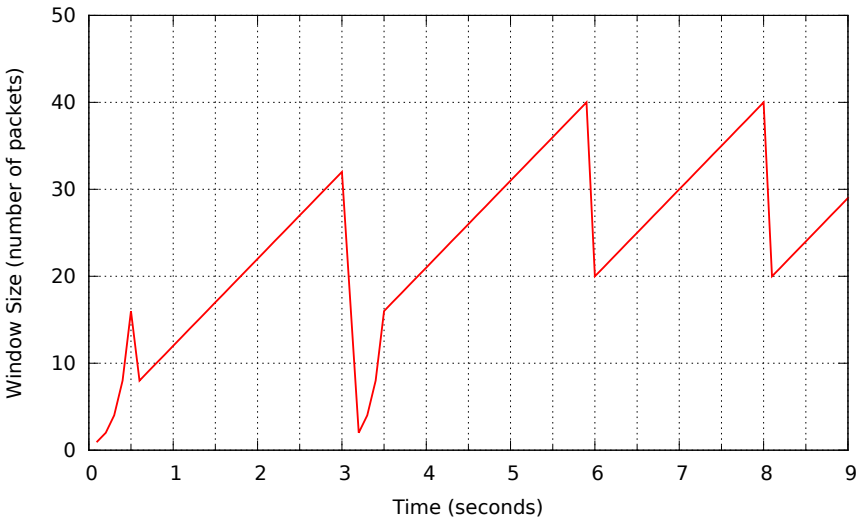


Figure 8: Congestion window size over time.

- Identify what happens to the congestion window at the following times: (i) t = 0.5 secs, (ii) t = 3 secs, (iii) t = 3.5 secs, and (iv) t = 8 secs.

  For each case, you should: 1. describe the state transition (previous state and next state), 2. identify the event that caused it, and 3. explain how we can conclude that from the graph.

  Example: *at $t = 2$ secs the sender transitions from state u to state v because event x occurred. We can see that event x has occurred there, because the window size changes from y to z.*

  (i) at t = 0.5 secs, the sender transitions from Slow Start to Fast Recovery. We can see that Triple duplicate ACKs has occurred, because the sender reduces the congestion window to half its original size. We can already tell that the previous state was Slow Start because of the exponential increase.

  (ii) at t = 3 secs, the sender transitions from Congestion Avoidance to Slow Start. We can see that Packet Timeout because the congestion window decreases to 1, and a slow start state follows

this event. We know that the previous state was Congestion Avoidance because the window size was increasing linearly (+1 packet per RTT)

(iii) at t = 3.5 secs, the sender transitions from Slow start to Congestion Avoidance. This is because the congestion window becomes larger than ssthresh. We can see this transition from the difference in the curves. (Slow Start and Congestion Avoidance, as explained before)

(iv) at t = 8 secs, the sender transitions from Congestion avoidance to Fast recovery. We can see that Triple duplicate ACKs has occurred, and not a timeout, because the window size decreases to half, and a Congestion Avoidance state follows.

- Calculate the number of packets that the TCP sender transmits between t = 6 secs and t = 8 secs.

The sender is in the Congestion Avoidance state between t = 6 secs and t = 8 secs. We know that when a sender is in the Congestion Avoidance state, the congestion window of the sender increases by 1 with each transmission round.

The window size is 20 at t = 6 secs and 40 at t = 8 secs. Thus, (if we include last transmission at exactly t = 8 secs, when the sender transmits 40 packets), there are 21 transmission rounds (from 20 packets to 40 packets). Therefore, the total number of packets transmitted is:

$$\sum_{i=20}^{40} i = \frac{20+40}{2} \cdot 21 = 630$$

- Calculate the RTT of the TCP flow.

We know that the window size changes by 1 packet per RTT. Following up on the analysis done in the previous question, the window size changes 20 times from t = 6 secs till t = 8 secs (from 20 packets to 40 packets). Thus, $RTT = \frac{8-6}{40-20} = 0.1$ secs.

Finding a security loophole

In Figure 9, Alice is sending a large file to Bob using TCP. Denis tries to disrupt their communication by sending traffic to Céline. No other hosts send any traffic.

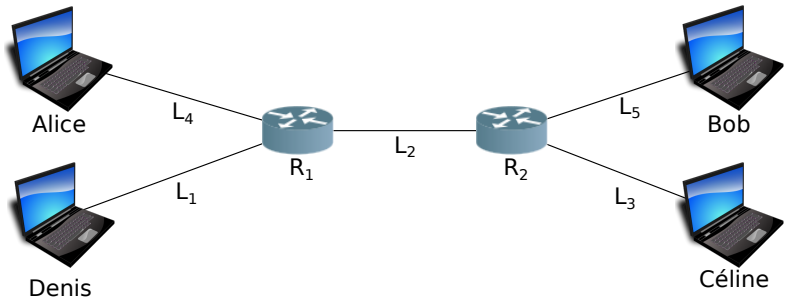


Figure 9: Network topology.

- Describe the simplest attack strategy that achieves Denis's goal. What condition needs to hold for the transfer rates of the links such that this strategy works?

  Denis prevent Alice and Bob from communicating with each other by flooding link L_2. To do so, Denis could send a constant stream of high-volume UDP traffic to Céline.

  This attack will succeed if and only iff the capacity of L_1 is greater or equal to the capacity of L_2.

  If the capacity of L_1 is greater or equal to the capacity of L_2, Denis can send traffic at a rate equal to the capacity of L_2. Since L_1 can carry Denis' attack traffic, all of it will reach L_2 at the desired rate. Thus, Denis' attack will succeed.

  Otherwise, if the capacity of L_1 is lower than the capacity of L_2, the attack will fail. This is because, even if Denis sends attack traffic at a higher rate, L_1 cannot carry traffic at a higher rate than the capacity of L_1. Since the capacity of L_1 is lower than the capacity of L_2, L_2 will have some spare capacity to accommodate Alice's traffic.

  Thus, we have proven that the condition we described is both necessary and sufficient.

- How will the TCP connection between Alice and Bob be affected by this attack? Draw a simple diagram that shows how Alice's congestion window, *cwnd*, evolves over time during the attack. You do not need to provide specific time values on the x-axis, just show the trend (e.g., does *cwnd* increase monotonically?)

We can see an example diagram at Figure 10. You can see that after the attack has commenced (sometime between 6 seconds and 8 seconds), Alice is no longer able to reliably reach Bob.

Thus, Alice's transmissions will almost always timeout.

(In the figure it is not shown how the timeout estimation algorithm of TCP will make Alice's timeout achieve increasingly bigger values, as Alice tries to determine the RTT between her and Bob.)

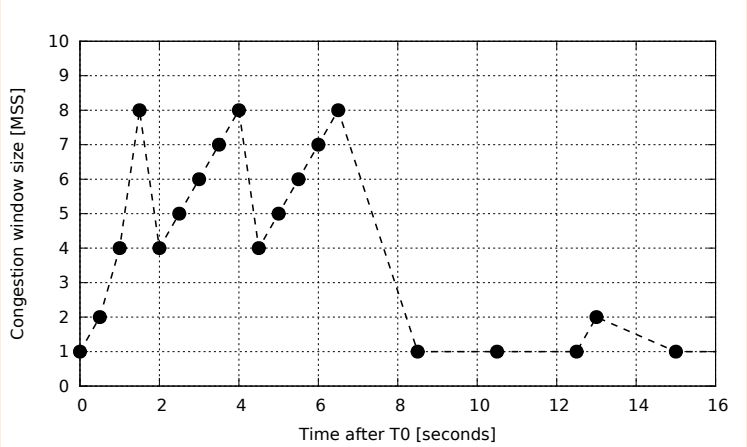


Figure 10: Congestion window of Alice over time

- Describe the attack strategy that achieves Denis's goal while minimizing the amount of traffic that Denis sends to Céline.
  Hint: Denis does not need to send traffic at a constant rate.

The TCP algorithm makes the sender transmits its messages in bursts. As long as there is no serious congestion issues on the channel, segments will flow from the sender to the receiver in a relatively steady rate. In that state, the congestion window oscillates between $\frac{w}{2}$ and w, where w is the maximum achievable value for the congestion window.

However, when a channel becomes heavily congested (e.g., due to an ongoing attack), the sender will be experiencing constant timeouts. Thus, all traffic transmitted by the sender will be done in bursts. Moreover, these bursts will transmitted at the precise moment when the sender's timeout event triggers.

Thus, it is possible for Denis to attack the TCP flow by targeting

those retransmission events. Every time that Denis expects Alice to retransmit her packets, he sends a burst of traffic to saturate the buffers of R_1. Thus, when Alice's retransmitted packet arrives at the congested router, R_1 will drop the segment of Alice, and Alice will have to wait for her timeout again. Naturally, Denis can stop transmitting flood traffic for as long as Alice is not about to transmit a new segment.

An added benefit of targeting timeout retransmissions is that such failed retransmissions cause TCP to exponentially increase its timeout interval. Thus, Alice will be sending progressively less frequently. This means that Denis will have to send even less traffic as time goes on.

What we have described so far is an attack strategy that works. In order to argue that this strategy is optimal, we need to examine whether an attack where we transmit a fewer number of messages is possible.

Note that the goal of Alice is to be able to send traffic to Bob at some predictable rate, even if this rate is significantly lower than before Denis' intervention. The goal of Denis is to make it so that Alice has no guarantee about how long it will take for Alice's segment to reach Bob.

Denis only transmits attack messages while Alice is retransmitting timed-out packets. If Denis fails to transmit enough messages to flood L_2, the segment of Alice will not get dropped, and Alice will be able to transmit a new piece of information to Bob. This will make Denis' attack ineffective. Thus, for Denis' strategy to be effective, Denis should always transmit at least as many messages are necessary to block Alice's retransmissions. Since Denis is not transmitting any attack traffic at any other point in time, it is impossible that there exists a strategy more optimal than the one we described.

A recap of everything

Alice has opened a persistent TCP connection to Bob. At time T_0, Alice starts sending to Bob, over this connection, a file of size 12 bytes in segments of $MSS = 1$ byte.

Figure 11 shows how the congestion window of Alice, *cwnd*, changes over time after T_0 and until the file transfer completes. Each of the $five$ points in the graph shows the time a change in *cwnd* took place and *cwnd*'s value after the change.

Make the following assumptions:

- Transmission delay is negligible.
- Bob sends an ACK for each segment it receives.
- The first segment that Alice transmits after T_0 has sequence number 10.
- Fast-retransmit is disabled.
- Only one segment gets lost after T_0, and it is a segment sent by Alice.
- B stores (does not discard) all out-of-order packets.

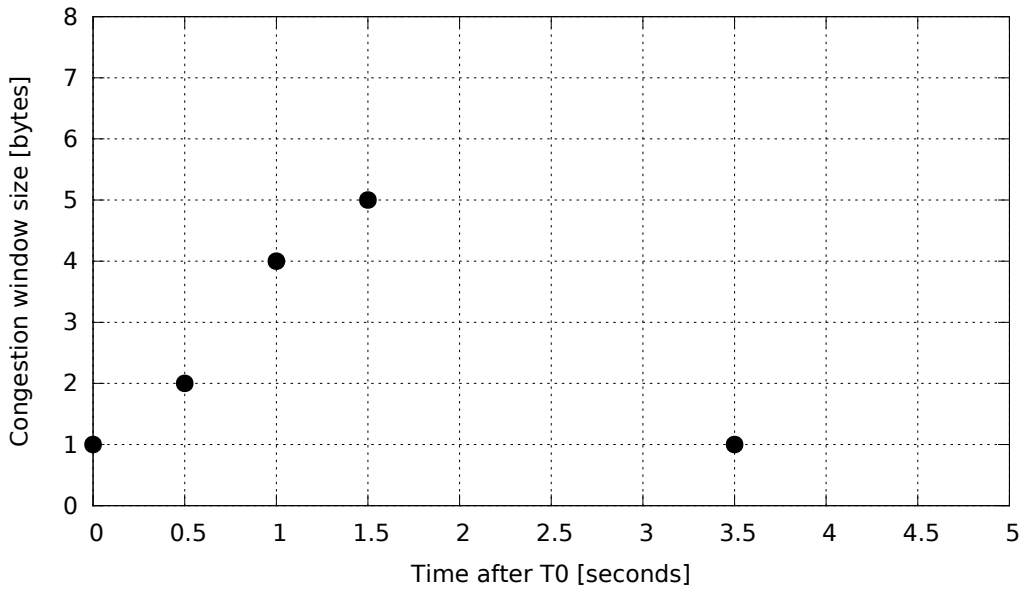


Figure 11: Congestion window of Alice over time

- What is the RTT between Alice and Bob?

The RTT between Alice and Bob is 0.5 seconds.

We can extract this piece of information from what happens at time 1 seconds after T_0 and 1.5 seconds after T_0. TCP is in Congestion avoidance phase during that time, and the size of the congestion window increases by 1 MSS (=1 byte) per RTT during that phase.

Since the congestion window increases from 4 to 5, between 1 secs and 1.5 secs, RTT is equal to:

$\frac{1.5-1}{5-4} = 0.5$ secs

- What is the retransmission timeout used by Alice?

The transmission timeout of Alice is equal to 2 seconds.

According to the information we are given, there is only one packet loss (and possible cause for a timeout). This timeout, obviously, happens at time 3.5 secs, since the window drops to size 1 MSS. According to the sequence diagram we have drawn, the segment that was lost was transmitted at time 1.5. Thus, the timeout is equal to $3.5 - 1.5 = 2$ secs.

- What was the size of Alice's congestion window, *cwnd*, the last time a packet was lost before T_0?

The size of *cwnd* the last time a packet was lost before T_0 was 8 (or 9) bytes.

The value of ssthresh updated if and only if there is a packet loss, and it is set to $\frac{cwnd}{2}$. Since, in our sequence diagram, the current value of ssthresh is 4, the last there was a packet loss (and ssthresh's value was updated), *cwnd* was $2 \cdot 4 = 8$ bytes.

If we assume that ssthresh has any non-integral values rounded down, another possible value could have been 9, since 9 divided by 2 also yields 4.

- Complete the diagram in Figure 13 that shows what happens after T_0 and until the file transfer completes:
  - All segments exchanged between Alice and Bob.
  - The sequence numbers sent by Alice and the acknowledgment numbers sent by Bob.

- The state of Alice's congestion-control algorithm.
- The size of Alice's congestion window, `cwnd`, in bytes.
- The value of Alice's slow-start threshold, `ssthresh`, in bytes.

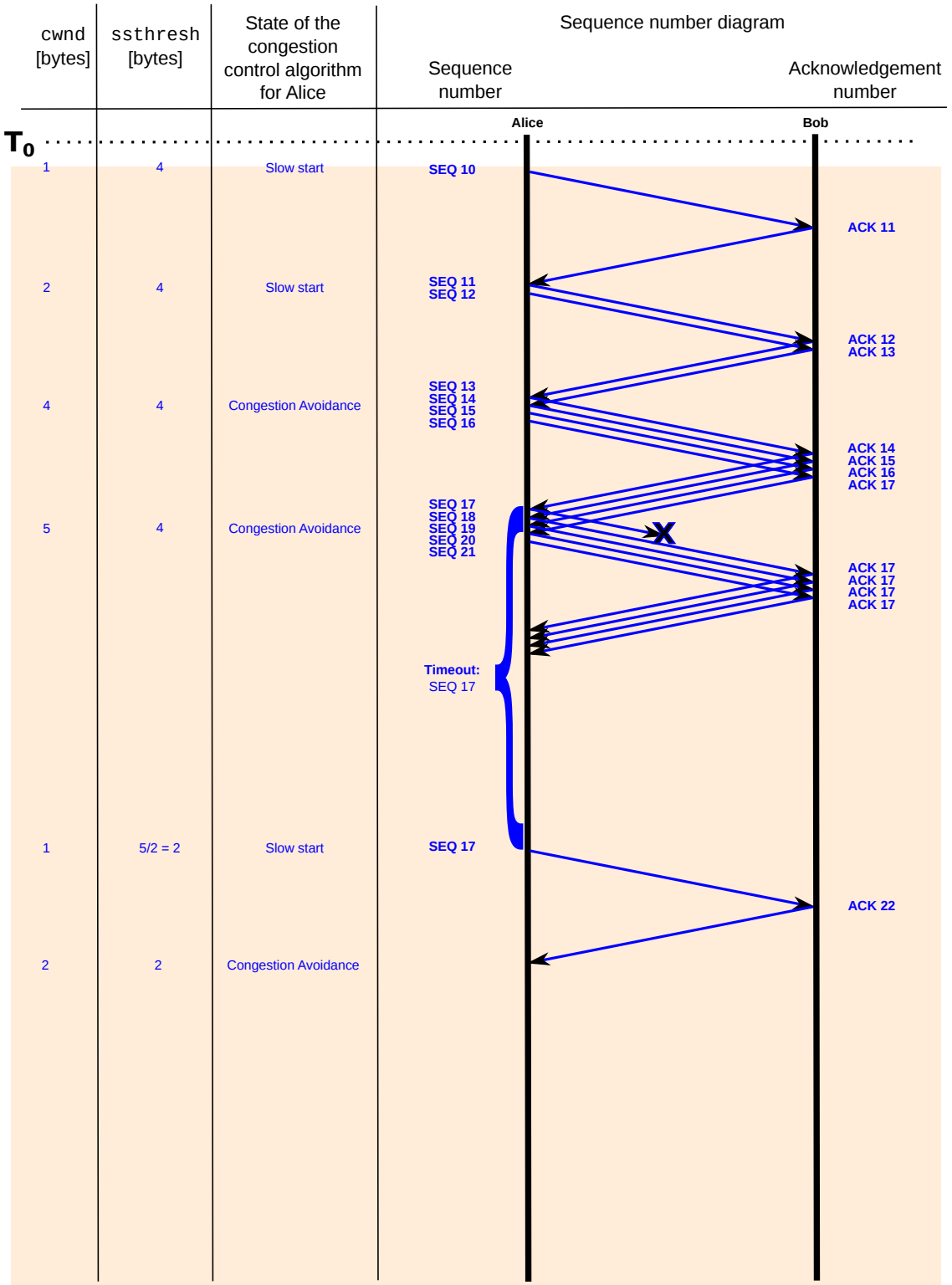


Figure 13: Sequence diagram.

- How long does the file transfer take? Assume that the file transfer completes once Alice has received the final ACK for file data.

  From the sequence diagram, we can see that the complete file transfer takes:

  - $3 \times RTT$ (to receive the ACKs for packets 13 to 16)
  - A timeout
  - $1 \times RTT$ (to receive the ACK for packet 17, which has been retransmitted)

  Thus, in total we have $4 \cdot RTT + timeout = 4 \cdot 0.5 + 2 = 4$ seconds.

- Now assume (just for this part) that fast-retransmit is enabled. Does this change the duration of the file transfer and how/why?

  If Fast-Retransmit were enabled, the file transfer would take 2.5 seconds instead.

  This is because at time 2 seconds, Alice would have finished receiving 3 triple duplicate ACKs for SEQ 17. Thus, Alice would have retransmitted the lost packet at time 2 seconds, as opposed to time 3.5 seconds (which is the case in the sequence diagram).

  As a result, this would decrease the file transfer time by 1.5 seconds. Thus, the total transfer time would be $4 - 1.5 = 2.5$ seconds.