Tutorial 2 solution

1 TCP Reliable Data Transfer

Task A

Consider transferring an enormous file of *L* bytes from Host A to Host B. Assume an *MSS (Maximum Segment Size)* of 536 bytes.

a. What is the maximum value of *L* such that TCP sequence numbers are not exhausted? Recall that the TCP sequence number field has 4 bytes.

b. For the *L* you obtain in (a), find how long it takes to transmit the file. Assume that a total of 66 bytes of transport, network, and data-link header are added to each segment before the resulting packet is sent out over a 155 Mbps link. Ignore flow control and congestion control so A can pump out segments back to back and continuously.

Solution

a. Length of TCP sequence number field = 4 bytes = 32 bits.

There are 2^32 possible sequence numbers.

Recall that, in TCP, the sequence number does not increment by one with each segment. Rather, it increments by the number of bytes of data sent.

As such, the maximum size file that can be sent from A to B is simply the number of bytes representable by 2^32 , so *L* = bytes = 4,294,967,296 bytes ≃ 4.29 Gbytes.

b. Maximum size file = *L* = 4,294,967,296 bytes.

Maximum Segment size = *MSS* = 536 bytes.

Number of segments = ceil (*L* / *MSS*) = ceil (2^32/ 536) ≃ 8,012,999 segments.

66 bytes of header get added to each segment, giving a total of (66) (8,012,999) = 528,857,934 bytes of header.

Total number of bytes transmitted = Ltotal= total number of data bytes + total number of header bytes = 4,294,967,296 + 528,857,934 ≃ 4.824 Gbytes.

Total time taken to transmit file = Ltotal/ *R* = (4.824 x10^9x 8) / (155 x10^6) ≃ 249 seconds.

Task B

Host A and B are communicating over a TCP connection, and Host B has already received from A all bytes up through byte 126.

Suppose Host A then sends two segments to Host B back-to-back. The first and second segments contain 70 and 50 bytes of data, respectively. In the first segment, the sequence number is 127, the source port number is 302, and the destination port number is 80. Host B sends an acknowledgment whenever it receives a segment from Host A.

a. In the second segment sent from Host A to B, what are the sequence number, source port number, and destination port number?

b. If the first segment arrives before the second segment, in the acknowledgment of the first arriving segment, what is the ACK number, the source port number, and the destination port number?

c. If the second segment arrives before the first segment, in the acknowledgment of the first arriving segment, what is the ACK number?

d. Suppose the two segments sent by A arrive in order at B. The first acknowledgment is lost and the second acknowledgment arrives after the first timeout interval. Draw a timing diagram, showing these segments and all other segments and acknowledgments sent. (Assume there is no additional packet loss.) For each segment in your figure, provide the sequence number and the number of bytes of data; for each acknowledgment that you add, provide the ACK number.

SOLUTION

a. Recall that, in TCP, the sequence number for a segment is the byte-stream number of the first byte in that segment.

As the first segment sent had 70 bytes and a sequence number of 127, the sequence number of the second segment will be 127 + 70 = 197.

The source port will be the same used in the connection, which is 302. The destination port will be 80.

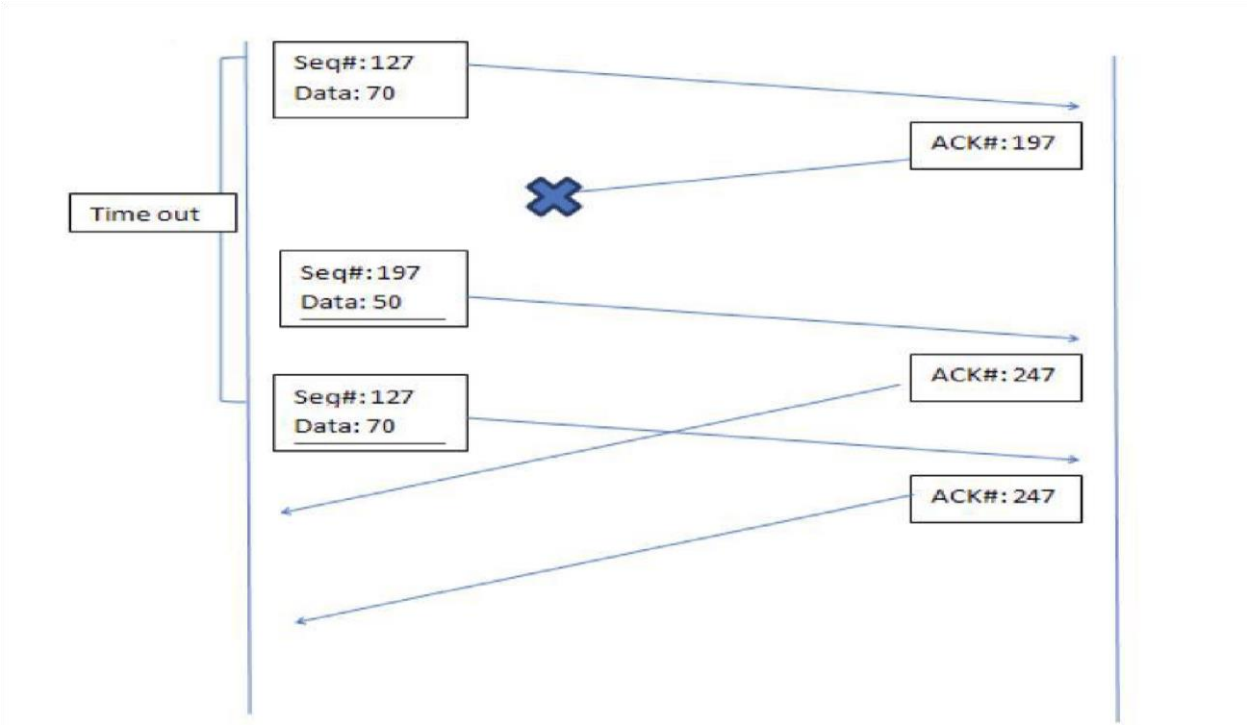
b. In TCP, the ACK number is the next byte expected.

If the first segment arrives before the second, this means that the next byte expected is the first byte of the second segment, so the ACK number will be 197.

Since we are now in the server side, the source and destination ports are swapped; source port is 80 and destination port is 302.

c. If the second segment arrives before the first, this means that the server has received the segment with sequence number 197, but hasn't received the segment with sequence number 127, which is actually the segment that it was expecting to receive. In this case, the server indicates in the ACK number of the response that it has the first 126 bytes, but still expects to receive the one with sequence number 127, so the ACK number would be 127.

d. Although the server retransmits the segment with sequence number 127, the server tells the sender that it has everything OK until byte 246, and that it is expecting the segment with sequence number 197 + 50 = 247 next.



---

2 Round-Trip Time and timeout Estimation in TCP

Q1

- We discussed TCP needs to estimate a value for RTT to know how long it has to wait for an ACK.
- To estimate the RTT, TCP uses SampleRTT values that are the time passed between a segment sending and the arrival of its ACK.
- However, TCP doesn't use the SampleRTT associated to the retransmitted segments.

Why do you think TCP avoids measuring the SampleRTT for retransmitted segments?

SOLUTION

The retransmitted segments contain the same data and the same sequence numbers as the previously sent packets. Therefore, for the retransmitted segments, we would expect the same ACK numbers as the previously sent TCP segments. When we receive these particular ACKs from the receiver, we don't really know or care whether they were sent as responses to the retransmitted segments or to the previously sent TCP segment.

Q2:

Assume we have the following measurements for the SampleRTT in ms each at a time respectively such that:

SampleRTT1= 12 ms
Sample RTT2=22ms
Sample RTT3=23ms
Sample RTT4=35ms

Use the exponential weighted moving average to compute the estimated timeout , given the following weights $\alpha=0.15$, $\beta=0.30$

SOLUTION

To calculate the estimated time out first calculate the estimated RTT and the RTT variation DevRTT (estimate how SampleRTT deviated from EstimattedRTT) as follows:

Estiamted RTT i = (1- α) * EstimattedRTT (i – 1) + α * SampleRTT i

DevRTT i = (1- β) * DevRTT (i – 1) + β * |SampleRTT i – EstimatedRTT i|

Estimatted RTT1 = SampleRTT1=12ms
DevRTT1=|SampleRTT1-EstimatedRTT1|=|12-12|=0
(1st estimation , there was no previous samples so all the weight is for the 1st current sample)

Estiamted RTT2=(1- α)EstimattedRTT1+ αSampleRTT2=(1-0.15)12+(0.15)22=13.5ms
DevRTT2=(1- β) DevRTT1+ β |SampleRTT2-EstimatedRTT2|=(1-0.30)(0)+(0.30)(22-13.5)=2.55

Estiamted RTT3=(1- α)EstimattedRTT2+ αSampleRTT3=(1-0.15)13.5+(0.15)23 = 14.925ms
DevRTT3=(1- β) DevRTT2+ β |SampleRTT3-EstimatedRTT3| = (1-0.30)(2.55)+(0.30)(2314.925)=4.2075

Estiamted RTT4=(1- α)EstimattedRTT3+ αSampleRTT4=(1-0.15)14.925+(0.15)35=16.725ms
DevRTT4=(1- β) DevRTT3+ β |SampleRTT4-EstimatedRTT4|=(1-0.30)4.2075+(0.30)(35-16.725)=8.42775

TimeoutInterval = EstimatedRTT + 4*DevRTT

So the Timeout Interval given these 4 SampleRTTs= EstimatedRTT4 + 4*DevRTT4=16.725+(4*8.42775)=50.436 ms

---

3 PORT NUMBERS AND IP ADDRESSES

TASK A:

Suppose Client A initiates a Telnet session with Server S. At about the same time, Client B also initiates a Telnet session with Server S. Provide source and destination port numbers for

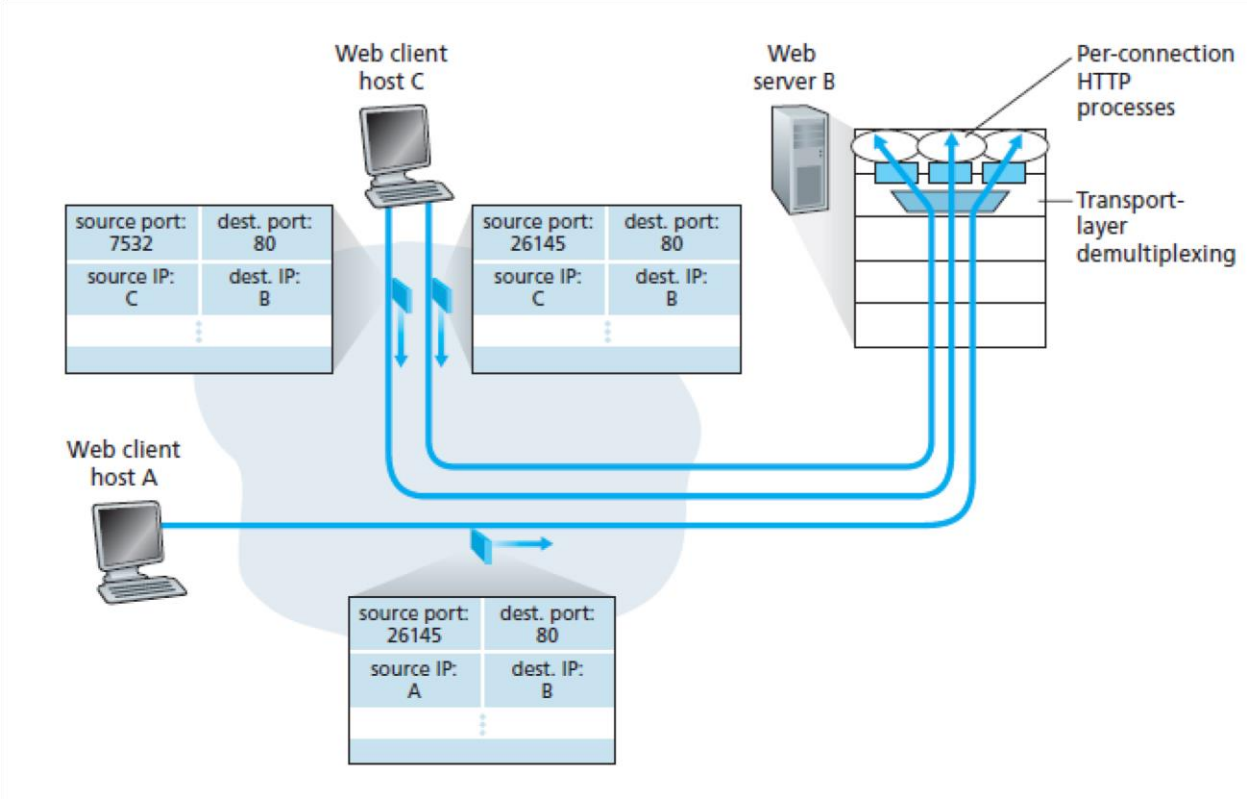
a) The segments sent from A to S.
b) The segments send from B to S.
c) The segments sent from S to A.
d) The segments sent from S to B.
e) If A and B are different hosts, is it possible that the source port number in the segments from A to S is the same as that from B to S?
f) How about if they are the same host?

SOLUTION

a) Source port: 467 (any other port number between 1024 and 65535 is also accepted)
   Destination port: 23
b) Source port: 513 (any other port number between 1024 and 65535 is also accepted)
   Destination port: 23
c) Source port: 23 Destination port: 467
d) Source port: 23 Destination port: 513
e) Yes.
f) No.

TASK B: Consider the following figure.

What are the source and destination port values in the segments flowing from the server back to the clients' processes? What are the IP addresses in the network-layer datagrams carrying the transport-layer segments?



SOLUTION

Suppose the IP addresses of the hosts A, B and C are a, b, c respectively. (Note that a, b, c are distinct).
To host A:
Source port = 80
Source IP address = b
Destination port = 26145
Destination IP address = a To host C (left process): Source port = 80
Source IP address = b
Destination port = 7532 Destination IP address = c To host C (right process):
Source port = 80
Source IP address = b
Destination port = 26145 Destination IP address = c

---

4 MULTIPLEXING AND DEMULTIPLEXING

TASK A: CONNECTIONLESS MULTIPLEXING AND DEMULTIPLEXING

Suppose a process in Host C has a UDP socket with port number 6789. Suppose both Host A and Host B each sends a UDP segment to Host C with destination port number 6789.

Will both of these segments be directed to the same socket at Host C?
If so, how will the process at Host C know that these two segments originated from two different hosts?

SOLUTION

Yes, both segments will be directed to the same socket. This is because each UDP socket is fully identified with a two-tuple: (destination IP address, destination port number). In addition, both segments have the same destination IP address and destination port number.
For each received segment, at the socket interface, the operating system will provide the process with the source IP addresses of Hosts A and B to determine the origins of the individual segments.

TASK B: CONNECTION-ORIENTED MULTIPLEXING AND DEMULTIPLEXING

Suppose that a Web server runs in Host C on port 80. Suppose this Web server uses persistent connections, and is currently receiving requests from two different Hosts, A and B.
Are all of the requests being sent through the same socket at Host C?
If they are being passed through different sockets, do both of the sockets have port 80? Discuss and explain.

SOLUTION

For each persistent connection, the Web server creates a separate "connection socket". Each connection socket is fully identified with a four-tuple: (source IP address, source port number, destination IP address, destination port number).
When host C receives an IP datagram, it examines these four fields in the TCP segment contained within the datagram to determine to which socket it should pass the payload of the TCP segment. Thus, the requests from A and B pass through different sockets.
The identifier for both of these sockets has 80 for the destination port; however, the identifiers for these sockets have different values for source IP addresses. Unlike UDP, when the transport layer passes a TCP segment's payload to the application process, it does not specify the source IP address, as this is implicitly specified by the socket identifier.