Ονοματεπώνυμο : Στυλιανός Αδάμου
AT : 1038399

R7

Yes, both segments will be directed to the same socket. For each received segment, at the socket interface, the operating system will examine source IP addresses to determine to which process the segment will be transmitted.

R8

For each persistent connection, the Web server creates a separate "connection socket". By using source IP address, source port number, destination IP address, destination port number demultiplexs the segment/datagram in host C side and find in which socket each segment will be transmitted. Therefore A and B pass through different sockets. The identifier for both of these sockets has 80 for the destination port but different source IP addresses.

R9

Used for sequential numbering of packets of data flowing from sender to receiver. Gaps in the sequence numbers of received packets allow the receiver to detect a lost packet. Sequence numbers are required for a receiver to find out whether an arriving packet contains new data or is a retransmission (duplicate packet will be discard by checking sequence number)

R10

Used to timeout/retransmit a packet, possibly because the packet (or its ACK) was lost within the channel -> Used to handle losses in the channel. Because timeouts/retransimissions can occur when a packet is delayed but not lost (premature timeout), or when a packet has been received by the receiver but the receiver-to-sender ACK/NAK has been lost.

R15

a) 20 bytes
b) ack number = 90

P2

Server -> client A
Source port =80, source IP address = C, dest port = 9157, dest IP address = A
Server -> client B left process:
Source port =80, source IP address = C, dest port = 5775, dest IP address = B
Server -> client B, right process:
Source port =80, source IP address = C, dest port = 9157, dest IP address = B

P6

Όταν ο sender βρίσκεται στην κατάσταση "Wait for call 1 from above" και ο receiver βρίσκεται στην κατάσταση "Wait for 1 from below". Ο sender στέλνει πακέτα με sequence number 1 και μεταβαίνει στο "Wait for ACK or NAK 1" όπου περιμένει για ACK ή NAK. Αν ο receiver λάβει το πακέτο με sequence number 1, στέλνει ένα ACK και μεταβαίνει στην κατάσταση "Wait for 0 from below", περιμένοντας για το πακέτο με sequence number 0. Το πακέτο αυτό είναι corrupted και ο rdt2.1 sender λάβει το corrupted ACK

Ονοματεπώνυμο : Στυλιανός Αδάμου
AT : 1038399

θα ξαναστείλει το πακέτο με sequence number 1. Όμως, ο receiver περιμένει να λάβει το πακέτο με sequence number 0 και θα στείλει NAK όταν δεν λάβει αυτό το πακέτο.

P15

L/R = 1500 * 8 bits /10^9 = 12msec
N = Window size
RTT =30msec
util =0.98 = N * (L/R) / RTT+ L/R = N * 0.012 / 30.012
N=2451 packets -> more than 2451 packets must be the window size
Window size = 2450

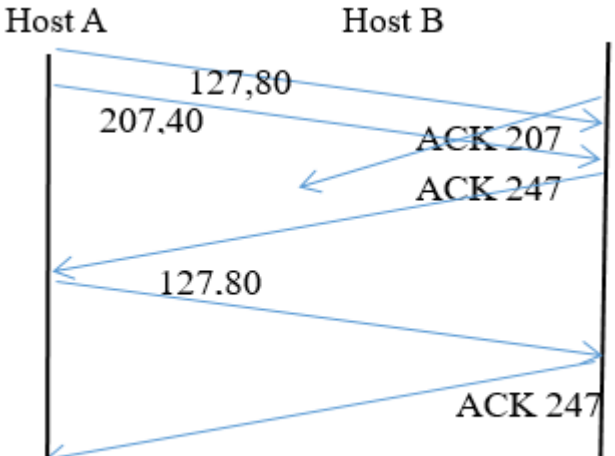
P22

a) window size = N=3. if the receiver has received packet k-1, and has ACKed all k-1 packets and before and all of those ACK's has been receiver by the sender then sender's window is [k, k+N-1] as the window will slide to the next packets that it have to forward. But if none of the ACKs have been received at the sender, the sender's window contains k-1 and the N packets up to and including k-1. The sender's window is thus [k-N,k-1].

b) If the receiver is waiting for packet k, then it has received (and ACKed) packet k-1 and the N-1 packets before that. If none of those N packets ACKs have been yet received by the sender, then ACK messages with values of [k-N,k-1] may still be propagating back. Because the sender has sent packets [k-N, k-1], it must be the case that the sender has already received an ACK for k-N-1. Once the receiver has sent an ACK for k-N-1 it will never send an ACK that is less that k-N-1. Thus the range of in-flight ACK values can range from k-N-1 to k-1.

P27

a) sequence number is 207, source port number is 302 and destination port number is 80
b) acknowledgement number is 207, the source port number is 80 and the destination port number is 302.
c) acknowledgement number is still 127 as it waits for bytes 127 and after.
d)



Ονοματεπώνυμο : Στυλιανός Αδάμου
ΑΤ : 1038399

P31

EstimatedRTT = (1- a) * EstimatedRTT + a * SampleRTT
DevRTT = (1- b) * DevRTT + b * |SampleRTT – EstimatedRTT|
TimeoutInterval = EstimatedRTT + 4*DevRTT

After obtaining first sampleRTT 106ms:
EstimatedRTT = 0.875*100 + 0.125*106 = 100.75ms
DevRTT = 0.75*5 + 0.25 * 106 -100 = 5.25ms
TimeoutInterval =100.75 + 4*5.25= 121.75ms

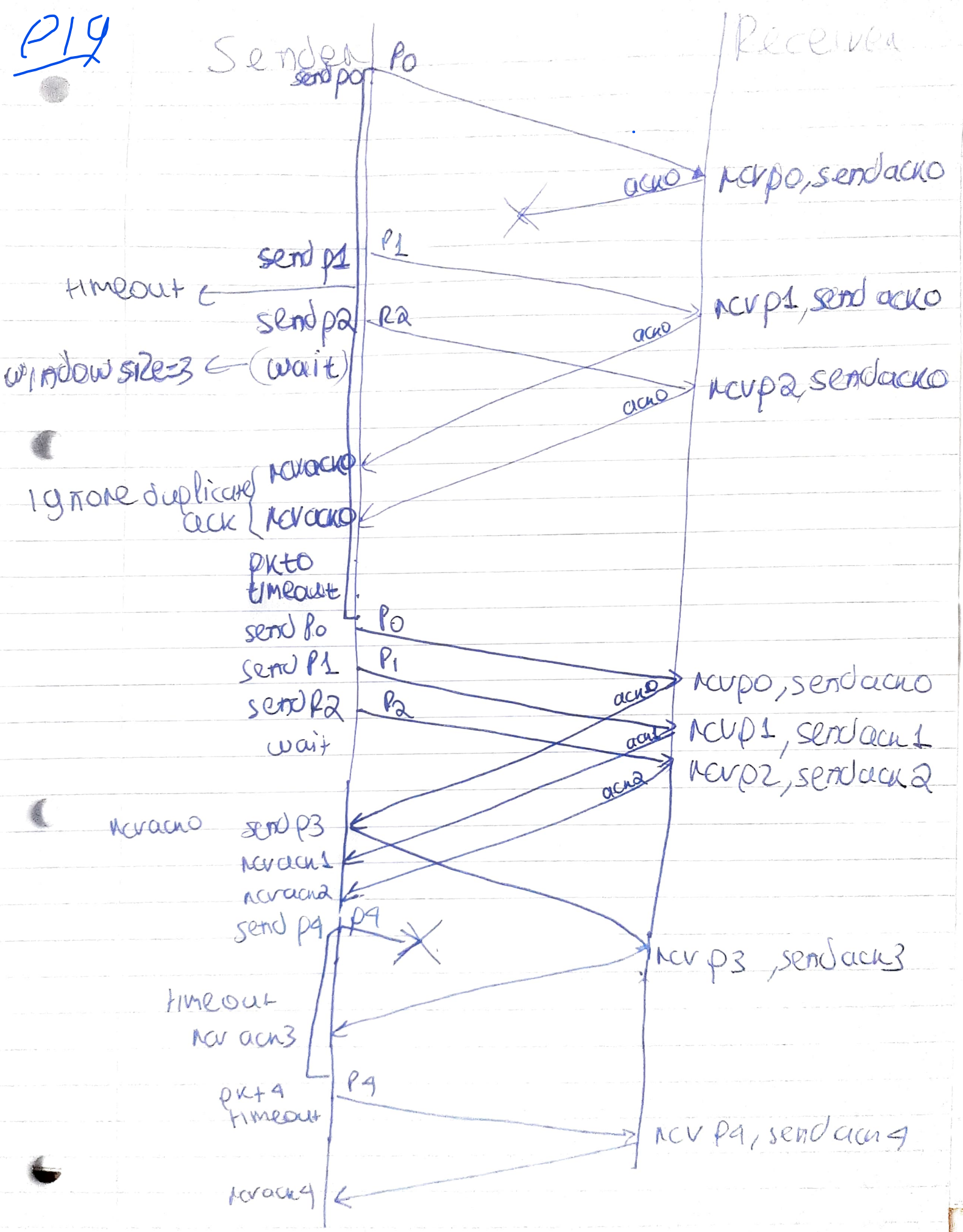
After obtaining first sampleRTT 120ms:
EstimatedRTT = 0.875*100.75 + 0.125*120 = 103.16ms
DevRTT = 0.75*5.25 + 0.25 * 120 -100.75 = 8.75ms
TimeoutInterval =103.16+ 4*8.75= 138.16ms

After obtaining first sampleRTT 140ms:
EstimatedRTT = 0.875*103.16 + 0.125*140 = 107.77ms
DevRTT = 0.75*8.75 + 0.25 * 140 -103.16 = 15.77ms
TimeoutInterval =107.7+ 4*15.77= 170.85ms

After obtaining first sampleRTT 90ms:
EstimatedRTT = 0.875*107.77 + 0.125*90 = 105.55ms
DevRTT = 0.75*15.77 + 0.25 * 90 -107.77 = 16.27ms
TimeoutInterval =105.55 + 4*16.27= 170.63ms

After obtaining first sampleRTT 115ms:
EstimatedRTT = 0.875*105.55 + 0.125*115 = 106.73ms
DevRTT = 0.75*16.27 + 0.25 * 115 -105.55 = 14.57ms
TimeoutInterval =106.73+ 4*14.57= 165.01ms

P19


Sender
Receiver
send p0
p0
ack0
rcv p0, send ack0
send p1
p1
timeout
rcv p1, send ack0
send p2
p2
window size=3
wait
ack0
rcv p2, send ack0
ack0
rcv ack0
ignore duplicate ack
rcv ack0
pkt0 timeout
send p0
p0
send p1
p1
send p2
p2
wait
rcv p0, send ack0
ack0
rcv p1, send ack1
ack1
rcv p2, send ack2
ack2
rcv ack0
send p3
rcv ack1
rcv ack2
send p4
p4
rcv p3, send ack3
timeout
rcv ack3
pkt4 timeout
p4
rcv p4, send ack4
rcv ack4