Computer Networks Homework 2

10/20/2016

Username: Ronair

Ch3 P4. 1's complement:

a. Addition = 11000001 and 1's complement = 00111110
b. Addition = 01000000 (carry added to the LSB) and 1's complement = 10111111
c. If the addition result is the same. i.e., if the corresponding bits are flipped. E.g.: first byte = 01011101 and 2nd byte = 01100100.

Ch3 P6. rdt2.1:
(Sender referred to be from fig. 3.11 and receiver from fig. 3.57)

- There can be a condition where the receiver is in the state "Wait for 1 from below" and the sender is in the state "Wait for call 1 from above".
- The sender goes to "Wait for ACK or NAK 1" after sending a packet with a positive sequence number 1.
- The receiver receives this packet and goes to "Wait for 0 from below" after sending an ACK.
- If this ACK is corrupted when it is received at the sender, it resends the packet with sequence number 1.
- But receiver is waiting for 0 now, so it sends a NAK.
- This keeps on happening and neither the sender or receiver will stop this behavior, hence the deadlock.

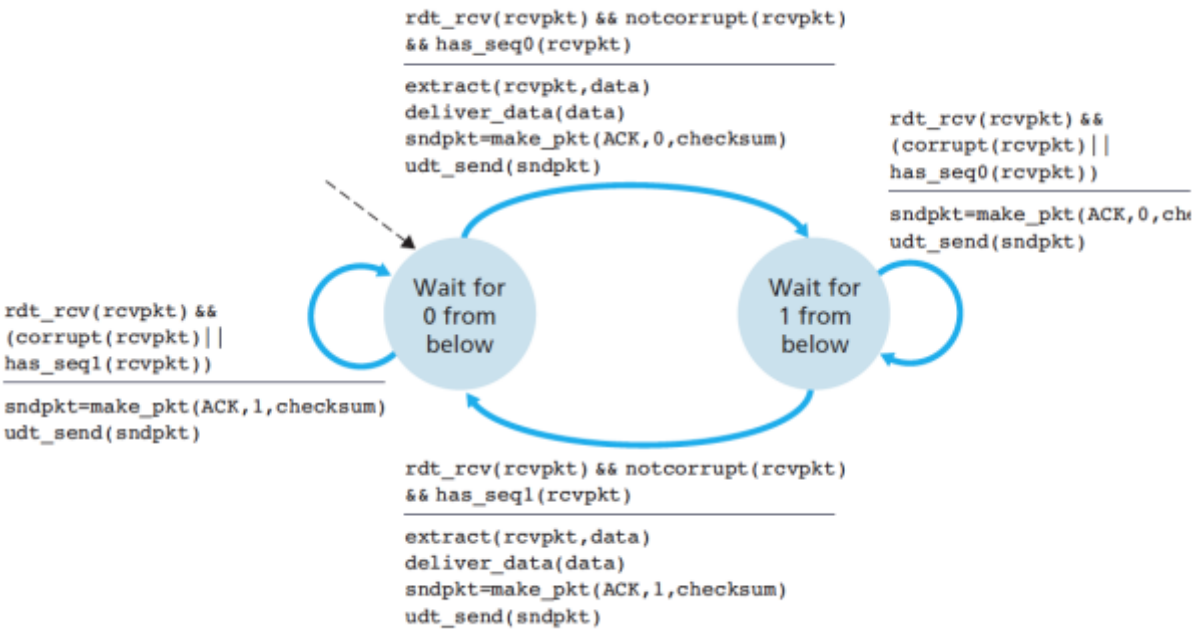
Ch3 P8. rdt3.0 FSM:
Since rdt 3.0 changes the sender, introduces timeouts. This enables it to be able to handle duplicates sent from sender to receiver.

The arrival of duplicates at rdt 3.0's receiver is already handled in rdt 2.2. ∴ we don't need to change it.

Computer Networks Homework 2

10/20/2016

Username: Ronair



Above receiver FSM is from the textbook, fig. 3.14

Ch3 P23.:

What's the problem: when the receiver receives a packet, it slides its window and sends an ack. If this ack is lost or sending time out, the sender retransmit the packet. If the newest sequence number in the receiver window is same as the first sequence number, then the receiver assumes the sender has sent the packet with the latest sequence number in its window. But this is wrong in 2 ways: a) It's actually a duplicate packet and b) receiver marks stores a wrong packet.

Will increase the sequence number range by w+1 solve the issue? It solves the above case, but just like this happened with one slide, it can happen with many, how many is the question.

Let's assume the worst case:

- Receiver receives all packets in its window, say 0 to w-1.
- Receiver sends an ack for all these w packets
- Receiver slides its window by w, new window sequence numbers are w to 2w – 1
- Assume all these acks are lost.
- The sender still has a window 0 to w-1.

- So, in the worst case, sender retransmits 0.
- Receivers window should not wrap around to 0, till the system ensures that the sender's 0 has been received.

Thus, the number of sequence numbers should be at least 2w.

(This can be generalized further by assuming first sequence number to be x and not 0).

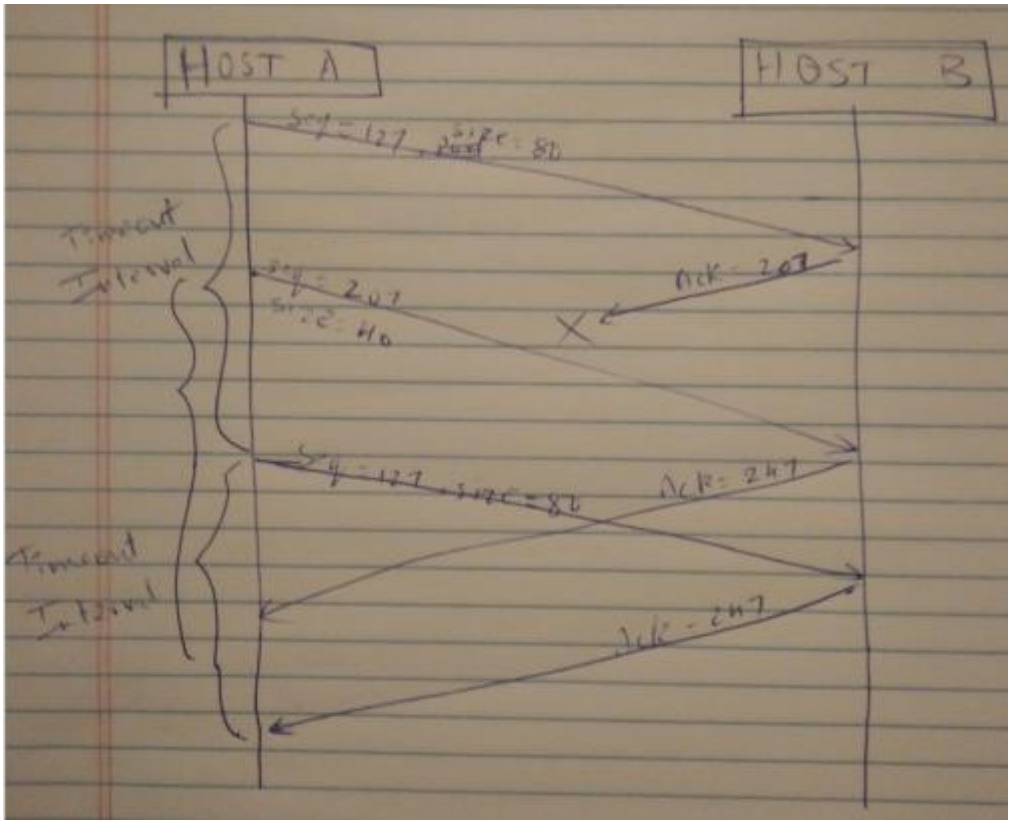
Ch3 P27.:

a. The sequence number will be 127(previous sequence number) + 80(previous packet bytes) = 207.
   The source port number can remain unchanged as 302 since it's again the sender.
   The destination port number can remain unchanged as 80 since it's again the receiver.

b. The first packet arrives at the receiver first with an a sequence number 127, number of bytes 80, source port as 302 and destination port as 80. The receiver sends an ack with sequence number 127 + 80 = 207, source port = 80 and destination port = 302.

c. The 2nd segment arrives first at the receiver, Since the receiver is expecting the segment with seq number 127, it'll send an ack number 127 to indicate the sequence number it expects next.

d.



Ch3 P28.:

- A can't send more than at the link capacity, so it can send at the most at 100 Mbps.
- B is slower at the rate of 40 Mbps, so it gets collected in the buffer. After a point, the buffer gets full and the receiver signals this to the sender by setting RcvWindow = 0.
- Hosts A stops sending till it sees a RcvWindow > 0.

This keeps on happening.

Ch3 P32.

a. EstimatedRTT4 = sampleRTT4
Formula: EstimatedRTT_new = (1 – a) * EstimatedRTT_old + a * SampleRTT4
EstimatedRTT3 = a * SampleRTT3 + (1-a) * SampleRTT4

And so on, we get:
EstimatedRTT1 = a * SampleRTT1 + (1-a)[a * SampleRTT2 + (1-a)[a * SampleRTT3 + (1-a)* SampleRTT4]]

= a * SampleRTT1
+ a * (1-a) * SampleRTT2
+ a * (1-a) * (1–a) * SampleRTT3
+ (1-a)^3 * SampleRTT4

b. We see a clear pattern here:
$EstimatedRTT1$
$= a * \sum_{k=1}^{n-1} (1-\mathrm{a})^{\mathrm{k}-1} * \mathrm{SampleRTT(k)} + (1-a)^{n-1} * SampleRTT\mathrm{n}$

Ch3 P37.

a. GBN:
A initially sends 1 to 5, out of which 2 fails.
B receives 1. So it expects 2 and sends ACK 1 when it receives 2, 3, 4 and 5.
A resends 2 to 5.
B sends 4 Acks.
Answer: A sends 9 packets; B sends 8 Acks.

SR:
A initially sends 1 to 5, out of which 2 fails.
B acks 1, 3, 4, 5.
A resends only 2.
B sends an ack for this.
Answer: A sends 6 packets; B sends 5 Acks.

TCP:
A initially sends 1 to 5, out of which 2 fails.
B sends 4 Acks with seq number 2.
A resends 2.
B sends Ack with seq number 6.
Answer: A sends 6 packets; B sends 5 Acks.

b. TCP has the fast retransmission feature where the sender does not wait till timeout expires in order to resend the packet. It assumes the packet is lost if it doesn't get an Ack within the expected time and retransmits the packet even before the timeout expires. This is the fastest among the 3 mentioned.

Ch3 P40.:

a. In TCP slow start, the send rate grows exponentially after every transmission round. We see this behavior from [1,6] and [23,26].
b. In TCP congestion avoidance, the send rate increases. We see this behavior from [6,16] and [17,22].
c. After the 16th transmission round, the send rate would've dropped to 1 if there was a timeout. But we see no such behavior, it was because of triple duplicate ACKs.
d. After the 22nd transmission round, the send rate drops to 1 due to timeout as a segment loss was detected.
e. We can see at transmission round 6 that the it starts operating congestion avoidance and stops slow start at congestion window size 32 (ssthresh)
f. ssthresh is set to half the value when a loss is detected. At transmission round 16, when a loss is detected, it reduces the congestion window from 42 to 21.
g. Packet loss is detected at 22 and the threshold was decreased from 29 to 14 (floor of half of existing window size). That's the threshold at 24.
h. Below are the segment numbers transmitted per transmission round:
   1 – 1
   2 – 2, 3
   3 – 4 to7
   4 – 8 to 15
   5 – 16 to 31
   6 – 32 to 63
   7 – 64 to 96
   Hence the 70^{th} segment was sent during the 7^{th} transmission round

i. ssthresh will be set to half of the congestion window size when the loss occurred, hence 4.
   Congestion window size will be set to new threshold + 3 MSS, which is 7.

j. When triple duplicate ACKs are received, in TCP Tahoe, we reduce the threshold to half i.e., 21 and congestion window to 4 (doubles twice from 1).

k. Below are the number of segments transmitted per transmission round:

17 – 1
18 – 2
19 – 4
20 – 8
21 – 16
22 – 21
∴ Total = 52 segments

Ch3 P45.:

a. The loss rate, L, is the ratio of the number of packets lost over the number of packets sent. In a cycle
1 packet is lost.
The number of packets sent
$= \left(\frac{w}{2} + 0\right) + \left(\frac{w}{2} + 1\right) + \cdots + w$ …. [the last term can be written as $\left(\frac{w}{2} + \frac{w}{2}\right)$]

$= \sum_{k=0}^{\frac{w}{2}} \left(\frac{w}{2} + k\right)$

$= \left(1 + \frac{w}{2}\right).\frac{w}{2} + \sum_{k=0}^{\frac{w}{2}} k$

$= \left(1 + \frac{w}{2}\right).\frac{w}{2} + \frac{1}{2}.\frac{w}{2}.\left(1 + \frac{w}{2}\right)$

$= \frac{3}{4}.w.\left(1 + \frac{w}{2}\right)$

$= \frac{3w}{4} + \frac{3w^2}{8}$

Hence, loss rate:
$L = \frac{1}{\frac{3w}{4} + \frac{3w^2}{8}}$

b. We have from above: $L = \frac{1}{\frac{3w}{4} + \frac{3w^2}{8}}$

As per the hint provided, we can say that the major contributor is the higher power of w

So we can say that $L \approx \frac{1}{\frac{3w^2}{8}}$ …. (~ is approximately equal to)

So, $w \approx \sqrt{\frac{8}{3L}}$

Average throughput of a connection $= 0.75.\frac{w}{RTT}$ …. (equation from textbook)

$$= \frac{3}{4}.\sqrt{\frac{8}{3L}}.\frac{1}{RTT}.MSS$$

$$= \frac{1.22474.MSS}{\sqrt{L}.RTT}$$

Ch3 P46.

a. Packets will be dropped if sending rate exceeds link capacity. Hence we can say that $\text{link capacity} = \frac{w.MSS}{RTT}$
We are given the following:
- link capacity $= 10Mbps = 10 * 10^6 = 10^7$
- MSS = 1500
- RTT = 0.15

∴ max w ≈ 125 segments

b. Congestion window varies from 50% to 100% of w.
Hence, average window size $= \frac{3}{4}w \approx 94$ segments
Average throughput $= \frac{w'.MSS}{RTT} = \frac{94*1500*8}{0.15} = 7{,}520{,}000 = 7.52Mbps$

c. After a packet loss the window size is w/2. Our aim is for it to reach w.

Window size increases by 1 after every RTT. So we need w/2 RTTs which is $\frac{94}{2} * 0.15 = 7.05$ seconds.