CMPSCI 453
HW7, Homework Problem Set 3

Tony Gao

March 12, 2018

1 Problem 1

$$01011011 + 01110010 = \begin{array}{r} {\scriptstyle 1\,1\,1\quad\ 1\quad} \\ 0\,0101\,1011 \\ +\,0\,0111\,0010 \\ \hline 0\,1100\,1101 \end{array} \tag{1}$$

$$+01100100 = \begin{array}{r} {\scriptstyle 1\,1\quad\ 1\,1\quad\quad} \\ 0\,1100\,1101 \\ +\,0\,0110\,0100 \\ \hline 1\,0011\,0001 \end{array} \tag{2}$$

$$\text{With wraparound} = 00110010 \tag{3}$$

$$\text{1's Complement} = 11001101 \tag{4}$$

1. The receiver detects error with the 1's complement checksum by adding with wraparound all three 8-bit bytes, and then the checksum. If the sum is 0xFF, there are no errors.
2. A 1-bit error cannot possibly go undetected. The error bit would result in the three 8-bit bytes adding to a different sum, such that adding the checksum would result in sum $\neq$ 0xFF.
3. A 2-bit error could possibly go undetected. If one bit at the same significance index in two of the 8-bit words are flipped, the sum of the three words would still be the same, resulting in the same sum. For example, if the first two words were 01011010 + 01110011 instead of 01011011 + 01110010 (LSB flipped), the sum is clearly still the same. If a 2-bit error in the same 8-bit word occurred, then the error would be detected.

2 Problem 2

Deadlock condition:

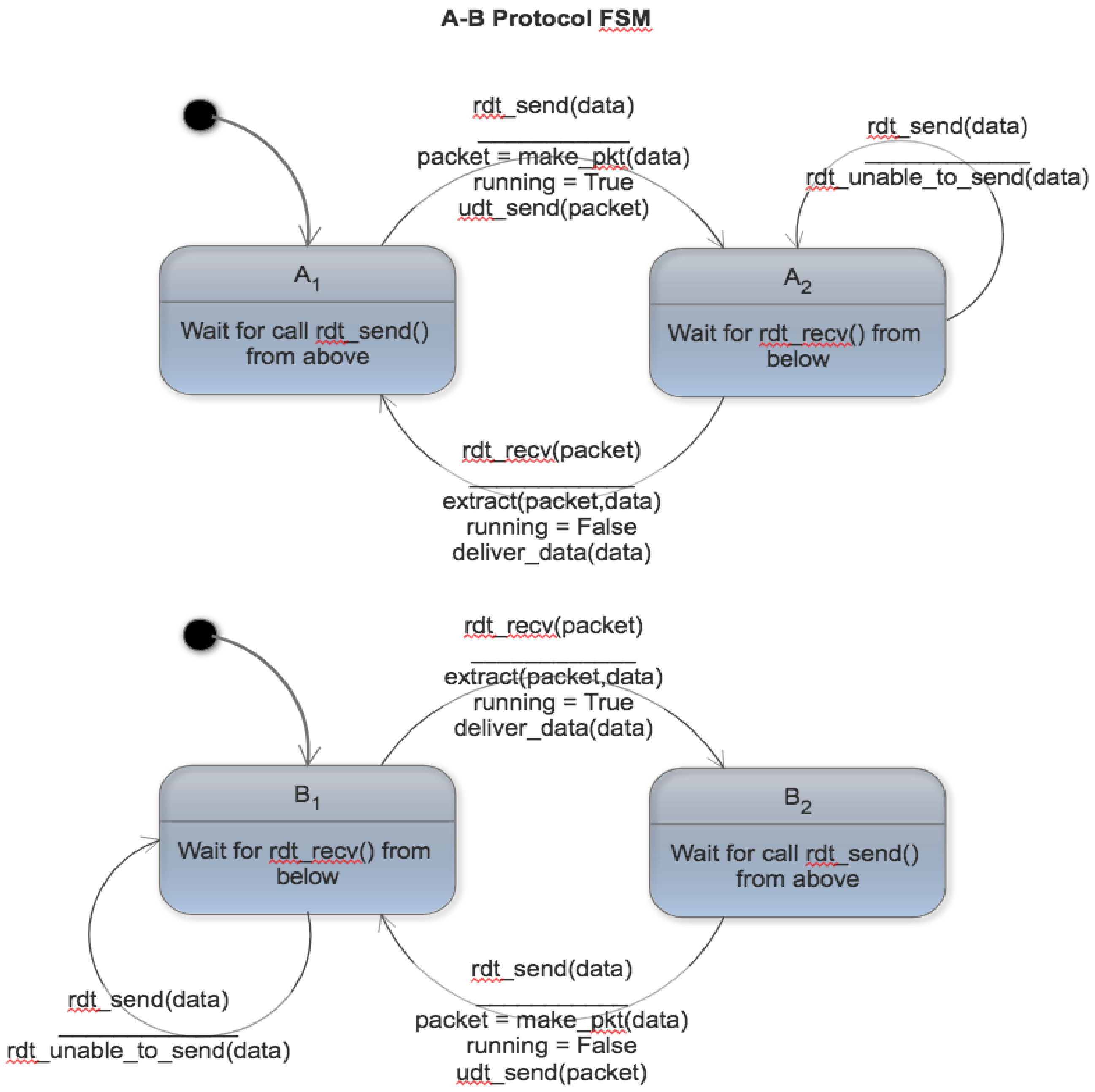
1. Sender sends packet 0, transitions from state (Wait for call 0 from above) to (Wait for ACK or NAK 0)
2. Receiver never receives packet 0, it is lost by the Sender's underlying Network/Link/Physical layer.
3. Sender is waiting for ACK or NAK 0.
4. Receiver is waiting for packet 0. Deadlock.

3 Problem 3

1. For infrequent data transfer: A protocol that uses ACK is preferable so packets that never make it to the receiver can be guaranteed to be resent in a short period of time. For example, given a packet n sent by A to B and is lost, B does not know that packet n is lost until it receives packet n+1 out of order, at which point it can send a NAK, but a long time may already have elapsed due to infrequent data transfer. ACK is clearly better here, since when A does not receive an ack for packet n within some time, A can infer a NAK and retransmit packet n.

2. For high volume data transfer over a low-loss channel, a nak-only protocol is preferable. Because data is arriving at a high rate to the receiver, the amount of time to send a NAK for a lost packet becomes very small. There is a significant reduction in packets sent from receiver to sender in the NAK-only protocol versus the ACK-only protocol.

4 Problem 4


A-B Protocol FSM
rdt_send(data)
packet = make_pkt(data)
running = True
udt_send(packet)
rdt_send(data)
rdt_unable_to_send(data)
A1
Wait for call rdt_send() from above
A2
Wait for rdt_recv() from below
rdt_recv(packet)
extract(packet,data)
running = False
deliver_data(data)
rdt_recv(packet)
extract(packet,data)
running = True
deliver_data(data)
B1
Wait for rdt_recv() from below
B2
Wait for call rdt_send() from above
rdt_send(data)
rdt_unable_to_send(data)
rdt_send(data)
packet = make_pkt(data)
running = False
udt_send(packet)

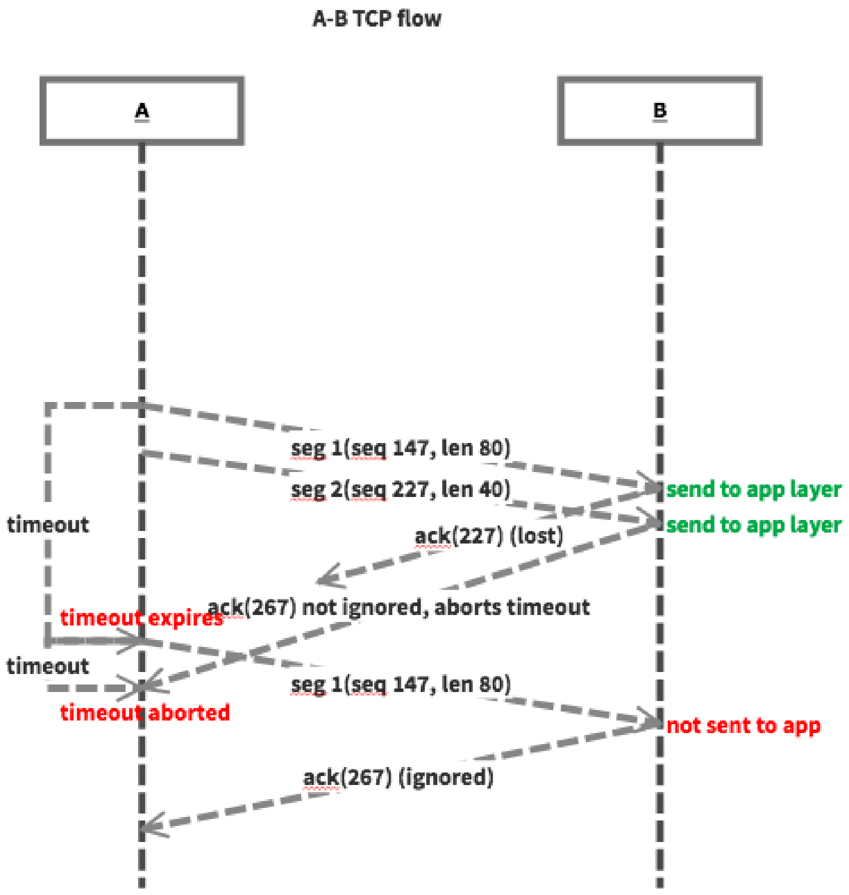
5 Problem 5

a) Since the sequence number of the first segment is 147, and there are 80 bytes, the sequence number of the second segment would be $147 + 80 = 227$. The source port is 304. The destination port is 80.

b) The acknowledgment number of the first byte is 227, which is the sequence number of the next byte expected from A. The source port number is 80, the destination port number is 304.

c) The ACK number for the first segment would be $147 + 80 + 40 = 267$. When the second segment arrives, B would send (ACK, 147) since 147 was the next expected sequence number. The second segment would be buffered. When the first segment arrives, B would send (ACK, 267) since both segments arrived. Finally, the transport layer of B will deliver these in order packets to the application layer.

d) The first timeout expires, the ack for segment 2 is received after the first timeout expires. Since ack(267) $>$ SendBase(147), SendBase is set to 267. There are no more not-yet-acked segments, so the timer is stopped. When the first timeout expires, the sender has not yet received a cumulative ack that includes sequence 147, so sender retransmits 147, sets the second timeout, receives ack 267, and ignores it because ack(267) $\leq$ SendBase, so no action is taken by the sender state machine.



6 Problem 6

a.

t	sender state	receiver state	packet type sent	seq # or ACK # sent
0	Wait for ACK 0	Wait for 0 from below	data	0
1	Wait for ACK 0	Wait for 1 from below	ack	0
2	Wait for ACK 0	Wait for 1 from below	data	0
3	Wait for ACK 0	Wait for 1 from below	ack	0
4	Wait for ACK 1	Wait for 1 from below	data	1
5	Wait for ACK 1	Wait for 0 from below	ack	1
6	Wait for ACK 0	Wait for 0 from below	data	0

b. The payload is delivered 2 times. Once for packet 1 (seq 0) and once for packet 3(seq 1). Packet 2 is received while receiver is in state (Wait for 1 from below) so it is not delivered. Data is passed up at t=1, t=5.

7 Problem 7

$$\begin{aligned}
\alpha &= 0.125, \beta = 0.25 && (5)\\
EstimatedRTT &= (1-\alpha)\cdot EstimatedRTT + \alpha \cdot SampleRTT && (6)\\
DevRTT &= (1-\beta)\cdot DevRTT + \beta \cdot |SampleRTT - EstimatedRTT| && (7)\\
TimeoutInterval &= EstimatedRTT + 4\cdot DevRTT && (8)\\
EstimatedRTT_0 &= 360ms && (9)\\
DevRTT_0 &= 21ms && (10)\\
SampleRTT_1 &= 270ms && (11)\\
DevRTT_1 &= (1-.25)\cdot 21ms + .25\cdot |270ms - 360ms| && (12)\\
&= 38.25ms && (13)\\
EstimatedRTT_1 &= (1-0.125)\cdot 360ms + 0.125\cdot 270ms && (14)\\
&= 348.75ms && (15)\\
TimeoutInterval_1 &= 348.75ms + 4\cdot 38.25ms = 501.75ms && (16)\\
SampleRTT_2 &= 320ms && (17)\\
DevRTT_2 &= (1-.25)\cdot 38.25ms + .25\cdot |320ms - 348.75ms| && (18)\\
&= 35.875ms && (19)\\
EstimatedRTT_2 &= .875\cdot 348.75ms + .125\cdot 320ms && (20)\\
&= 345.15625ms && (21)\\
TimeoutInterval_2 &= 345.15625ms + 4\cdot 35.875ms = 488.65625ms && (22)\\
SampleRTT_3 &= 250ms && (23)\\
DevRTT_3 &= (1-.25)\cdot 35.875ms + .25\cdot |250ms - 345.15625ms| && (24)\\
&= 50.6953125ms && (25)\\
EstimatedRTT_3 &= .875\cdot 345.15625ms + .125\cdot 250ms && (26)\\
&= 333.26171875ms && (27)\\
TimeoutInterval_3 &= 333.26171875ms + 4\cdot 50.6953125ms = 536.042969ms && (28)
\end{aligned}$$

8 Problem 8

The largest allowable sender window is of size $N \leq \frac{k}{2}$. This is the largest window that will guarantee that any resent packets as a result of dropped acks are definitely packets that were originally sent before the receive_base pointer. If the window was larger, it is possible for the sender and receiver windows to desynchronize in both SR and GBN such that the receiver will believe retransmissions to be new packets.

With a GBN sequence space of 2 and window size 2: if sender sends packets 0,1 back to back, and receiver acks 0,1 but both acks are lost, the sender window will be [0,1],0,1 and the receiver window will be 0,1,[0,1]. The sender would then retransmit 0 and 1 after timeout on 0 expires. Receiver will receive 0,1, deliver duplicate data upstream since it arrives in order and within the receiver window, and respond with ack. This moves sender window to 0,1,[0,1] and receiver window to 0,1,0,1,[0,1]. This delivers the same packets (0,1) twice upstream on the receiver side.

With a SR sequence space of 4 with window size 3 : if sender sends packets 0,1,2 back to back, and the receiver acks [0,1,2] but all acks are dropped, the sender window will still be at [0,1,2], but the receiver window will have progressed to

[3,0,1]. The sender ack timeouts on 0,1,2 would tick retransmitting packets [0,1,2]. The receiver will receive packets 0 and 1 and believe that they are new packets since they lie within the current receiver window. The receiver will discard packet 2. The receiver will ack packets [0,1,2] leading to sender sending packet 3, causing the receiver to deliver [3,0,1], of which only [3] is an actual new packet.

With a window size of 2, this problem is avoided. Sender sends [0,1]. Receiver ack [0,1] is dropped. Receiver window moves to [2,3]. Sender retransmits [0,1]. Receiver acks [0,1], discards retransmitted packets since $0, 1 \leq SendBase - 1$. Sender window moves to [2,3]. Sender sends [2,3]. Receiver acks [2,3], windows are synchronized.

9 Problem 9

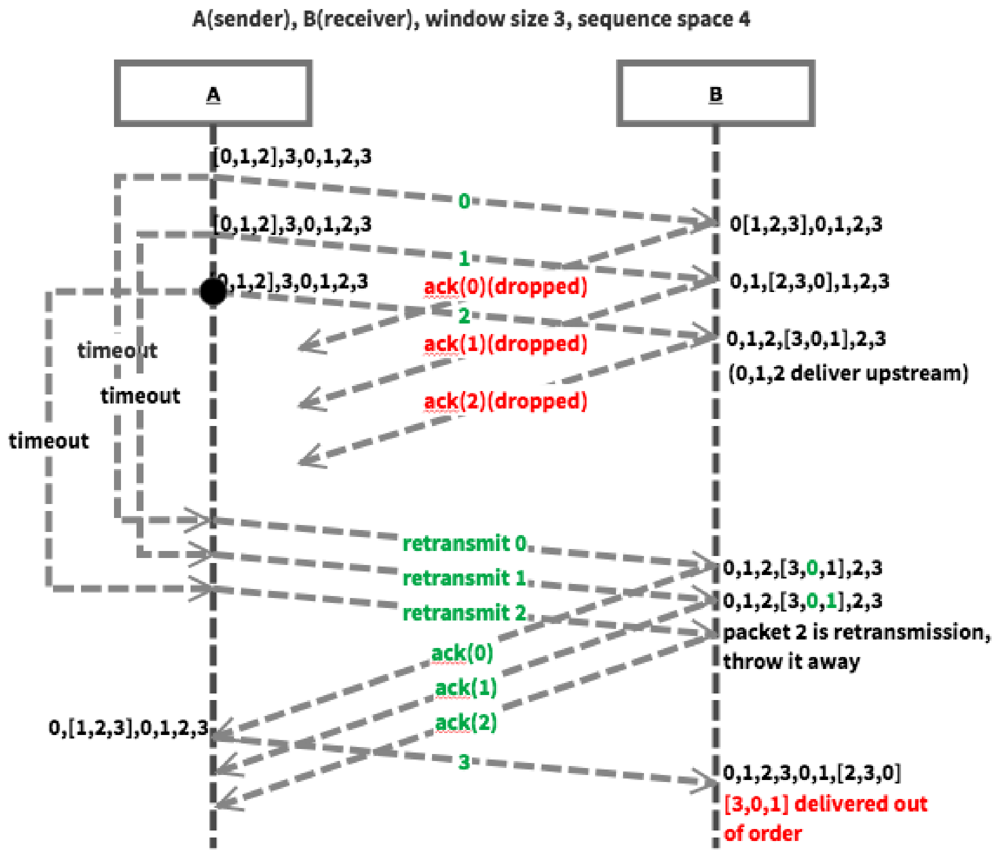


Figure 1: Illustration of the Selective Repeat condition described in Problem 8