**>>> Assignment #3 for Computer Networks (CNT 4004) <<<**

(due on Thursday, October 8th at the start of class)

This assignment covers material from chapter 3 of the textbook and from roughly the second two weeks of class lecture.

**Problem #1**

Answer the following questions about TCP and UDP checksums.

1. TCP and UDP use 16-bit checksums. The checksum code was given to you in class. Give the checksum for the following three 16-bit words 0x3346, 0x7766, 0x71AB.

0x3346 + 0x7766

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 |
| + | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
|  | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 |

(0x3346 + 0x7766) + 0x71AB

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 |
| + | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 |

Wrap around overflow bit 1

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 |
| + | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
|  | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 |

1’s complement

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 |
|  | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 |

1. Suppose that a UDP receiver computes the checksum for a received UDP segment and finds that it matches the value carried in the checksum field. Can the receiver be absolutely certain that no bit errors have occurred? If your answer is “no”, then give an example of bit errors (using three words in (a) above for your example) that could not be detected.
   * No. It is possible that two of the words 0x3346 and 0x7766 were altered/modified and become 0x7346 and 0x3766 respectively. When you add them up they lead to the same checksum as if they were not modified.

0x7346 + 0x3766

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 |
| + | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
|  | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 |

**Problem #2**

Design a protocol to do the following. Provide a service where a sender will broadcast a message every 5 seconds and a receiver will acknowledge each message. Each successive message that is acknowledged should contain an incrementing counter value (“1” for the first message sent after system reset of the sender, “2” for the second message sent, and so on). If a sent message is not acknowledged (that is, no ACK is received since the last message sent) then the next message should have the same sequence number as the previously sent message. It is possible that sent messages and/or ACKs may be lost. You will need to give (reasonable and necessary) assumptions and provide a list of messages, the syntax of each message, the semantics of each message, and the FSMs for the server and receiver. Any timers, variables, constants, etc. used in the FSMs must be defined.

Reference: protocol design lecture

* Assumptions:
  + A network exists for the broadcasting of messages
  + Messages may be lost and a reliable delivery is required
* List of messages:
  + MESSAGE
* Syntax of messages:
  + MESSAGE has one field containing an integer value
* Semantics of messages:
  + MESSAGE delivers count value
* Constants, variables, and timers
  + DELAY is an integer constant and set to 5
  + count is an integer variable
  + delayTimer is a countdown timer (seconds)

Sender Receiver

| |

| Reset 1 |

|<-------------------------------------------------------------------------------------- |

| count = 0, delayTimer = DELAY, start delayTimer |

| |

| |

| 2 delayTimer expired |

|---------------------------------------------------------------------------------------------------------->+

| broadcast MESSAGE with count value, |

| delayTimer = DELAY, start delayTimer |

| |

| ACK received 3 |

|<---------------------------------------------------------------------------------------------------------+

| count = count + 1, |

| |

| 4 ACK not received |

|---------------------------------------------------------------------------------------------------------->+

| broadcast MESSAGE with same count value, |

| delayTimer = DELAY, start delayTimer |

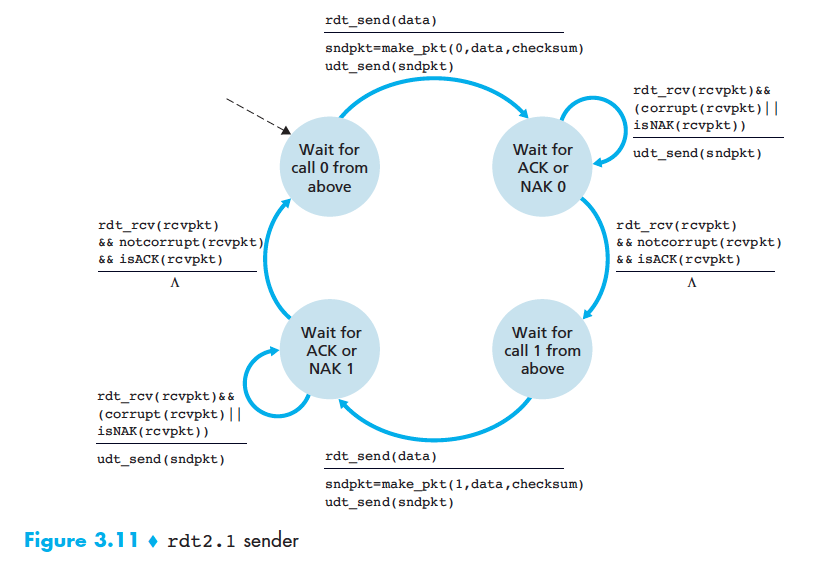
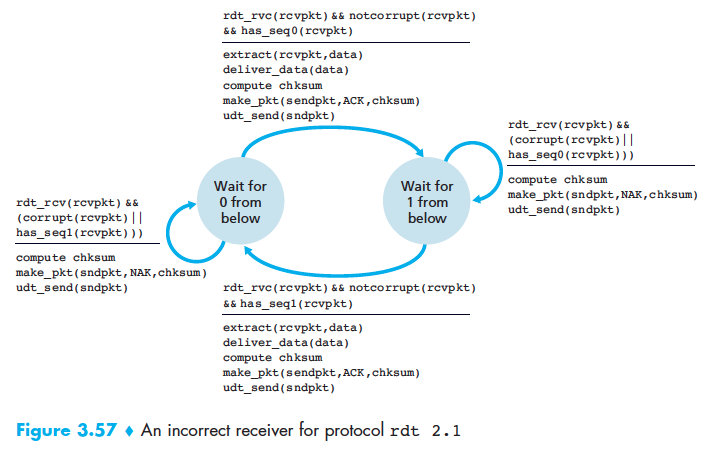
| |

**Problem #3**

Consider our motivation for correcting protocol rdt2.1. Show that the receiver, shown in Figure 3.57 (pg. 289), when operating with the sender shown in Figure 3.11 (pg. 211), can lead the sender and receiver to enter into a deadlock state, where each is waiting for an event that will never occur.

Answer Reference: http://www.ics.uci.edu/~keldefra/teaching/spring2013/uci\_cs132/problemsets/CS132\_EECS148\_ProblemSet3\_Solution.pdf )

Suppose the sender is in state “Wait for call 1 from above” and the receiver (the receiver shown in the homework problem) is in state “Wait for 1 from below.” The sender sends a packet with sequence number 1, and transitions to “Wait for ACK or NAK 1,” waiting for an ACK or NAK. Suppose now the receiver receives the packet with sequence number 1 correctly, sends an ACK, and transitions to state “Wait for 0 from below,” waiting for a data packet with sequence number 0. However, the ACK is corrupted. When the rdt2.1 sender gets the corrupted ACK, it resends the packet with sequence number 1. However, the receiver is waiting for a packet with sequence number 0 and (as shown in the home work problem) always sends a NAK when it doesn't get a packet with sequence number 0. Hence the sender will always be sending a packet with sequence number 1, and the receiver will always be NAKing that packet. Neither will progress forward from that state.

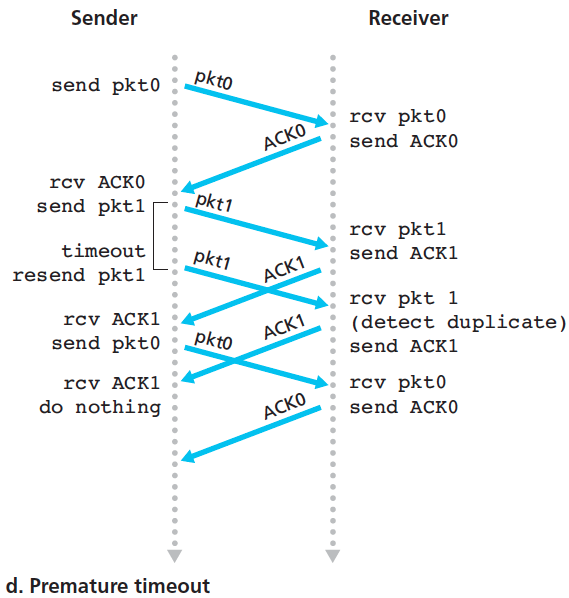


**Problem #4**

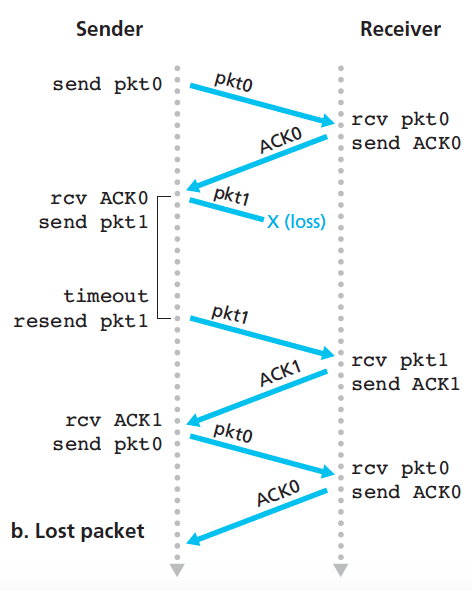
Consider the RDT protocol developed in the book using RDT 2.2 receiver and RDT 3.0 sender.

Reference pg. 216

1. If the time-out is less than the RTT between a sender and receiver sketch a timing diagram of the packet and ACK flows between the sender and receiver.



1. If the time-out is much greater than the RTT between a sender and receiver sketch the time diagram for the case of a packet loss.



1. Comment on the trade-off between time-out less than RTT and time-out much greater than RTT. Which condition could cause the most harm to other senders and receivers (that is, to other users of the network)? Explain why.

When the time-out is less than RTT it has to generate duplicate packets in the connection and takes up more bandwidth. When the time-out is greater than RTT then utilization goes down due to waiting. Time-out < RTT is worse because there are more duplicate packets generated and the network bandwidth will become congested.

**Problem #5**

Consider the rdt 3.0 protocol. Draw a diagram showing that if the network connection between the sender and receiver can reorder messages (that is, that two messages propagating in the medium between the sender and receiver can be reordered), then the alternating-bit protocol will not work correctly (make sure you clearly identify the sense in which it will not work correctly). Your diagram should have the sender on the left and the receiver on the right, with the time axis running down the page, showing data (D) and acknowledgement (A) message exchange. Make sure you indicate the sequence number associated with any data or acknowledgement segment.

Reference: https://www.coursehero.com/file/7190247/f11hw3sol/

**Problem #6**

Consider the following scenario.

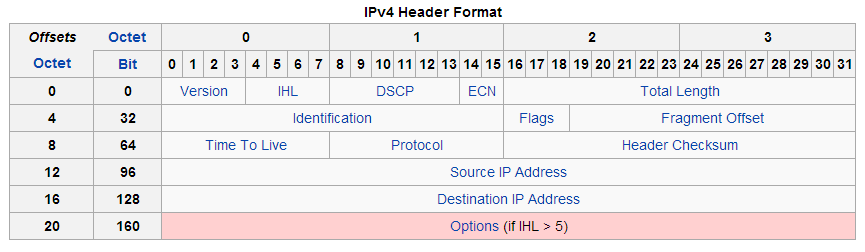
* Distance from sender to receiver is 3000 miles
* Data rate between sender and receiver is 1Mbps
* Data packet length is 1250 bytes
* ACK packet length is 64 bytes

1. What is the link utilization if an SAW protocol is used and you assume no bit errors (i.e., lost packets or lost ACKs never occur)?
   * tpack = 1250bytes \* 8bits/byte \* 1/1x106bits/s = 0.01  
     tprop = 1.584x107ft \* 1x10-9s/ft = 0.01584s  
     U = tpack/(tpack + tprop) = 0.01s/(0.01s + 0.01584s) = 0.387  
     U = 38.7%
2. What is the link utilization if an SAW protocol is used and you assume a bit error rate of 10-5 (you may assume that only data packets will have errors and that all errors are detectable at the receiver, ACK packets never contain bit errors).
   * U = (1 – 10-5s)\*(0.01s)/(0.01s + 2\*0.01584s) = 0.2399  
     U = 24%
3. What window size would be needed (given an SW protocol) to achieve 100% link utilization assuming no bit errors?
   * 1 = W\*tpack\*(1 - tpack)/[(1 – tpack + W\*tpack)\*(tpack + 2\*tprop)]  
     tpack + 2\*tprop = W\*tpack\*(1 – tpack)/(1 – tpack + W\*tpack)  
     W = 4.35

**Problem #7**

When decoding an Internet packet how can the decoder (be it you doing it by hand or a program doing it automatically) know if the packet is a TCP segment, a UDP segment, or something else? What might the “something else” be? “Guessing” is not the right answer, even if this is more-or-less what we did when decoding the packet in class?

* Looking at the IPv4 header we can see that the protocol number is located in the third row. If that specific byte is 6 in decimal then it refers to TCP. If that specific byte is 17 in decimal then it refers to UDP.



**Problem #8**

p.25 We have said that an application may choose UDP for a transport protocol because UDP offers finer application control (than TCP) of what data is sent in a segment and when.

Reference: https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=0CB4QFjAAahUKEwidlciO\_LHIAhXHuB4KHUpdC68&url=https%3A%2F%2Funiteng.com%2Fwiki%2Flib%2Fexe%2Ffetch.php%3Fmedia%3Dclasslog%3Acomputernetwork%3Ahw3\_report.pdf&usg=AFQjCNE91AMJqKeuks10s9TVMz84z7kUDg

1. Why does an application have more control of what data is sent in a segment?

* Consider sending an application message over a transport protocol. With TCP, the application writes data to the connection’s send buffer and TCP will grab bytes without necessarily putting a single message in the TCP segment (TCP may put more or less than a single message in a segment). UDP, on the other hand, encapsulates in a segment whatever the application gives it. So, if the application gives UDP an application message, this message will be payload of the UDP segment. Thus, with UDP, an application has more control of what data is sent in a segment.

1. Why does an application have more control on when the segment is sent?

* With TCP, due to flow control and congestion control, there may be significant delay from the time when an application writes data to its sender buffer until when the data is given to the network layer. UDP does not have delays due to flow control congestion control.

**Problem #9**

There are two ways to terminate a TCP connection, what are they? What are the implications (that is, what happens?) of each way?

Reference: https://en.wikipedia.org/wiki/Transmission\_Control\_Protocol

1. 4-way handshake
   1. The connection termination phase uses a four-way handshake, with each side of the connection terminating independently. When an endpoint wishes to stop its half of the connection, it transmits a FIN packet, which the other end acknowledges with an ACK. Therefore, a typical tear-down requires a pair of FIN and ACK segments from each TCP endpoint. After the side that sent the first FIN has responded with the final ACK, it waits for a timeout before finally closing the connection, during which time the local port is unavailable for new connections; this prevents confusion due to delayed packets being delivered during subsequent connections.
2. Reset
   1. Directly commands the server/client to terminate the connection immediately.

**Problem #10**

Write a program that will implement max-min scheduling by simulating the pouring process. The program should take as command line input the available bandwidth and up to 10 requested data rates (this for up to 10 sources). The bandwidth and data rates are all in integer increments of 1 Mb/s and the final allocation is also to be an integer value (that is, no allocations of less than 1 Mb/s allowed). Here below is an execution of the program written for the solution. You should give a screen shot showing the same inputs and outputs as in the below screen shot. Also, run your program for the following two cases:

* Available is 100 Mb/s – Source A requests 50 Mb/s, source B requests 70 Mb/s, and source C requests 10Mb/s (input: 100 50 70 10)
* Available is 100Mb/s – Source A requests 50 Mb/s, source B requests 700 Mb/s, and source C requests 10 Mb/s (input: 100 50 700 10)

