

**>>> SOLUTIONS <<<**

Welcome to Exam #1 in Computer Networks (CIS 6930). You have 75 minutes. Read each problem carefully. There are eight required problems (each worth 12.5 points) and one extra credit problem worth 5 points. You may have with you a calculator, pencils, eraser, blank paper, and one 8.5 x 11 inch "formula sheet". On this formula sheet you may have anything you want (definitions, formulas, etc.) *handwritten by you*. You may use both sides. Computer generated text, photocopies, and scans are not allowed on this sheet. Please submit your formula sheet with your exam. No sharing of calculators. Start each numbered problem on a new sheet of paper and do not write on the back of the sheets. Submit everything in problem order. Good luck and be sure to show your work.

**Problem #1**

- a) Central to the study of networks is the layered model. In this class we have at least two layered models that we work with. They are the OSI model and the Kurose/Ross model that structures our text book. Sketch these two models. For the OSI model, describe the function of the layers with one sentence for each layer.

OSI is Applications, Presentation, Session, Transport, Network, Data Link, and Physical (top to bottom). Kurose/Ross is Application, Transport, Network, Data Link, and Physical (basically, the TCP/IP stack). For the OSI model:

Application – provides access to user applications  
Presentation – provides data independence  
Session – manages end-to-end connections  
Transport – provides reliable end-to-end data transport  
Network – maintains point-to-point connections  
Data Link – provide reliable point-to-point data transport  
Physical – transmission of bit stream

- b) What are the four causes, or components, of delay in a packet switched network? What can be done to reduce each of these components?

The delay components are Processing, queueing, transmission, and propagation. Processing delay can be reduced by faster CPUs, queueing and transmission delay by faster links, and propagation delay only by moving content closer to its user.

**Problem #2**

- a) Describe the operation of a Web server with high-level pseudocode. It is sufficient to show operation for only the case of HTTP GET.

Main process

Listen for a connection on port 80  
Accept a connection and spin-off a thread or process to handle the connection

Handle GET process

Parse the HTTP header for the GET object name  
Open the file for the object  
If file not found, send an HTTP 404 message  
If file found, send an HTTP 200 message followed by the file  
Close the connection  
Exit the thread or process

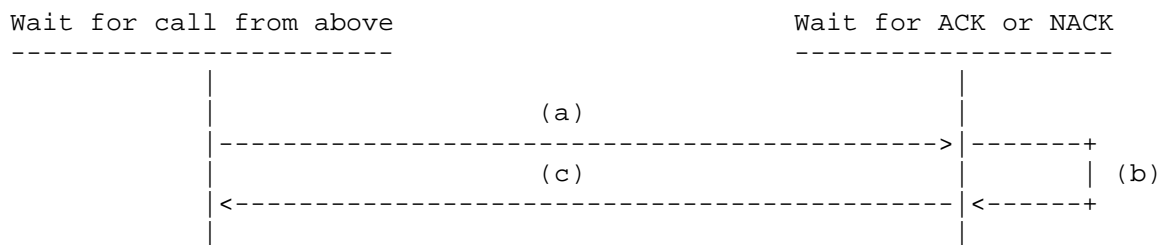
b) Attached is a sockets program. The program compiles successfully, but there are several errors in it. Identify these errors. Use the line numbers to help you describe where the errors are.

The program is missing `hton*()` conversions in lines 31 and 32. A `bind()` is missing following line 32. Also missing after line 32 is a `listen()`. In line 40 these should be a "+1" to the string length (to include the end of string character needed in the receiver). Finally, at the end of the program a `close` of the `connect_s` socket is missing.

### **Problem #3**

Consider a channel that connects a sender and a receiver. Assume that the channel never loses packets or gets them out of order, but it may corrupt them. Assume that the sender always sends unique packets and that the receiver does not care if it receives duplicate packets (e.g., it may have some way to determine if a packet is a duplicate). Design a reliable data transfer protocol for this system. You may use FSMs to describe your protocol.

FSM for the sender

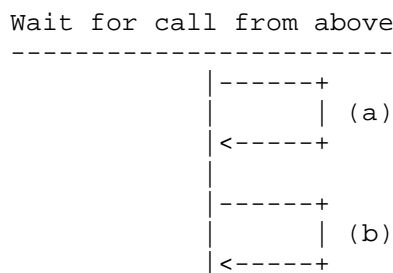


For transition (a) the event is `rdt_send(data)`. The action is `sendpkt=make(data, checksum), udt_send(sendpkt)`.

For transition (b) the event is `rdt_rcv(rcvpkt) && (isCorrupt(rcvpkt) || isNACK(rcvpkt))`. The action is `udt_send(sendpkt)`.

For transition (c) the event is `rdt_rcv(rcvpkt) && isACK(rcvpkt)`. The action is `nothing (null)`.

FSM for the receiver



For transition (a) the event is `rdt_rcv(rcvpkt) && corrupt(rcvpkt)`. The action is `udt_send(NACK)`.

For transition (b) the event is `rdt_rcv(rcvpkt) && notCorrupt(rcvpkt)`. The action is `extract(rcvpkt, data), deliver_data(data), udt_send(ACK)`.

For transition (b) the event is `rdt_rcv(rcvpkt) && (isCorrupt(rcvpkt) || isNACK(rcvpkt))`. The action is `udt_send(sendpkt)`.

#### **Problem #4**

Consider a link connecting a sender and receiver. The sender always has packets to send of length  $L$  bytes. GBN ARQ flow and error control is used. The link has a bit error rate of  $p$ . Write the expression for  $U$  in terms of  $t_{fr}$ ,  $t_{pr}$ ,  $N$ , and  $p$  for the case of window size is greater than “pipe size” (i.e.,  $N$  multiplied by  $t_{fr}$  is greater than two times  $t_{pr}$  plus  $t_{fr}$ ).

$$U = \frac{t_{fr}(1-p)}{t_{fr} + 2t_{pr}p} \text{ for } p = \text{probability of packet error. If } p = \text{probability of bit error, we have } U = \frac{t_{fr}(1-(1-(1-p)^{8L}))}{t_{fr} + 2t_{pr}(1-(1-p)^{8L})}$$

$$\text{which simplifies to: } U = \frac{t_{fr}(1-p)^{8L}}{t_{fr} + 2t_{pr} - 2t_{pr}(1-p)^{8L}}.$$

#### **Problem #5**

Carefully explain how TCP computes its retry time-out value.

TCP uses the following algorithm (with parameters  $g$ ,  $h$ , and  $f$ ) to estimate RTT and the possible deviation to RTT. The deviation is added to the estimated RTT. Exponential smoothing is used here.

$$\begin{aligned} \text{SRTT}(k+1) &= (1-g) \cdot \text{SRTT}(k) + g \cdot \text{RTT}(k+1) \\ \text{SERR}(k+1) &= \text{RTT}(k+1) - \text{SRTT}(k) \\ \text{SDEV}(k+1) &= (1-h) \cdot \text{SDEV}(k) + h \cdot \text{abs}(\text{SERR}(k+1)) \\ \text{RTO}(k+1) &= \text{SRTT}(k+1) + f \cdot \text{SDEV}(k+1) \end{aligned}$$

On a retransmit use exponential backoff of RTO (i.e., do not execute the above).

#### **Problem #6**

Consider five nodes (call them  $x_1, x_2, \dots, x_5$ ) that request bandwidth on a common channel. The nodes request 2, 5, 5, 7, and 10 units of bandwidth, respectively.

a) Give the max-min fair bandwidth allocation for a channel with 25 units of bandwidth.

All nodes can get what they request (i.e., the allocations are 2, 5, 7, and 10).

b) Repeat (a) for a channel with 15 units of bandwidth.

In the first round we allocate  $15/5 = 3$  units of bandwidth and get allocations 2, 3, 3, 3, and 3. There is 1 unit left to allocate to 4 nodes.

In the second round we allocate  $1/4 = 0.25$  units of bandwidth and get allocations of 2, 2.25, 2.25, 2.25, and 2.25. There is nothing leftover, so this is the final allocation

c) Repeat (a) for a channel with 10 units of bandwidth.

In the first round we allocate  $10/5 = 2$  units of bandwidth and get allocations 2, 2, 2, 2, and 2. There is nothing leftover, so this is also the final allocation.

## **Problem #7**

a) What is the key difference between distance-vector and link-state routing protocols in terms of how protocol messages are sent?

In distance-vector protocols message exchanges are between neighbors. In link-state protocols message exchanges are global (i.e., broadcast) to all nodes.

b) Give the names of a distance-vector algorithm and protocol. Give the names of a link-state algorithm and protocol.

RIP and BGP are distance-vector protocols. OSPF is a link-state protocol.

c) Give a simple example of the count-to-infinity problem.

Consider three nodes with A connected to B at cost 1 and B connected to C at cost 1. The route from A to C is through B at cost 2. If the link from B to C fails (now cost from B to C is infinity) the following will happen. At the next vector exchange A will learn that cost to B is infinity and enter this in its vector. However, B will learn that the cost to C is 2 via A and will enter 3 as its cost to C (going through A as the case may be). In the next exchange A will learn from B that the cost to C is 3 and now enter 4 as its cost to C. In this same exchange, B will learn from A that the cost to C is infinity, so it will enter infinity as its cost. And so on, until the cost to C is infinity for both A and B.

## **Problem #8**

a) What is the purpose of hierarchical address in the Internet (i.e., why are IP addresses hierarchical)? Explain how the hierarchical address is used in IP routing.

By having an IP address hierarchical as network-host, we can reduce the size of routing tables by having routing tables only handle forwarding from network to network. Only at the destination network is the host address portion used for delivering the packet to the destination host.

b) Assume that hosts A and B have a TCP connection established. Assume that the two hosts are separate by one router (i.e., they are one hop apart). Why does host A not directly use the MAC (LAN) address of host B when constructing its packets to send to host B?

Host A has no way of knowing the MAC address of host B on the other side of a router. A router does not pass broadcast ARP frames. In any case, the MAC address of the router port connecting host A's LAN to host B's LAN must be used to forward an A-to-B packet through the router. If some other MAC address (e.g., that of host B) were used, routing would not take place.

## **Extra Credit**

Give the names of up to five networking conferences. Each correct name will earn you one point of extra credit. Trade shows and local (e.g., from your university only) conferences do not count.

ICC, INFOCOM, GLOBECOM, SIGCOMM, and LCN are all networking conferences. Other possible answers include IPCCC, IC3N, Mobicom, and even VTC is acceptable. All have been discussed or mentioned in some context in the class. In any case, each of these conferences are likely to appear in the references in any paper you pick-up.

## Program for problem #2

```
1 //===== file = s.c =====
2 // = A message "server" program =
3 //=====
4 #include <stdio.h> // Needed for printf()
5 #include <string.h> // Needed for memcpy() and strcpy()
6 #include <windows.h> // Needed for all Winsock stuff
7
8 #define PORT_NUM 1050 // Arbitrary port number for the server
9
10 void main(void)
11 {
12     WORD wVersionRequested = MAKEWORD(1,1); // Stuff for WSA functions
13     WSADATA wsaData; // Stuff for WSA functions
14     unsigned int server_s; // Server socket descriptor
15     struct sockaddr_in server_addr; // Server Internet address
16     unsigned int connect_s; // Connection socket descriptor
17     struct sockaddr_in client_addr; // Client Internet address
18     struct in_addr client_ip_addr; // Client IP address
19     int addr_len; // Internet address length
20     char out_buf[100]; // 100-byte output buffer for data
21     char in_buf[100]; // 100-byte input buffer for data
22
23     // Initializes winsock
24     WSStartup(wVersionRequested, &wsaData);
25
26     // Create a socket
27     server_s = socket(AF_INET, SOCK_STREAM, 0);
28
29     // Fill-in my socket's address information
30     server_addr.sin_family = AF_INET; // Address family to use
31     server_addr.sin_port = PORT_NUM; // Port number to use
32     server_addr.sin_addr.s_addr = INADDR_ANY; // Listen on any IP address
33
34     // Accept a connection.
35     addr_len = sizeof(client_addr);
36     connect_s = accept(server_s, (struct sockaddr *)&client_addr, &addr_len);
37
38     // Send to the client
39     strcpy(out_buf, "Test message from server to client");
40     send(connect_s, out_buf, strlen(out_buf), 0);
41
42     // Receive from the client
43     recv(connect_s, in_buf, sizeof(in_buf), 0);
44     printf("Received from client... data = '%s' \n", in_buf);
45
46     // Close and clean-up
47     closesocket(server_s);
48     WSACleanup();
49 }
```