

>>> SOLUTIONS <<<

Welcome to the Mid-Term Exam for *Computer Networks*. Read each problem carefully. There are six required problems (each worth 16 points – you get 4 points for following these instructions correctly). There is also an additional extra credit question worth 10 points. You may have with you a calculator, pencils, erasers, blank paper, and one 8.5 x 11 inch “formula sheet”. On this formula sheet you may have anything you want (definitions, formulas, homework answers, old exam answers, etc.) as **handwritten by you** on both sides of the sheet. Photocopies, scans, or computer generated/printed text are not allowed on this sheet. You have 75 minutes for the exam. Please use a separate sheet of paper for each question. Good luck and be sure to show your work!

**Problem #1** (10 minutes)

Answer the following questions regarding the basics of computer networks and the Internet.

a) What are the fundamental measures of a communications system?

0.5 each and round-up (3 pts total)

Throughput, delay, loss, cost, mobility, robustness, and secrecy

b) What are the basics tasks of a communications system?

0.5 each and round-up (4 pts total)

Message formatting, error detection and recovery, addressing, routing, flow control, system management, security, and QoS

c) Precisely define *protocol* and *interface*

0.5 each and round-up (3 pts total)

A protocol is a complete set of rules regarding information exchange between same level layers between sites. An interface is a complete set of rules regarding information exchange between adjacent layers in a single site.

d) Sketch the OSI model. Describe the purpose of each layer.

0.5 each and round-up (3 pts total)

Application	7 - Provides access to applications
Presentation	6 - Provides data independence
Session	5 - Manages end-to-end connections
Transport	4 - Provides reliable end-to-end data transfer
Network	3 - Maintains point-to-point connections
Data link	2 - Provides reliable error-free point-to-point data transfer
Physical	1 - Transmission of bit stream (cabling, voltages, etc.)

e) Name at least three standards organizations (in the area of communications) and briefly describe what they standardize and how they operate.

1 each (3 pts total)

IETF – Internet standards (called RFCs). Entirely open, anyone welcome to come and present, everything posted on the web. Build it, standardize it. IEEE – Ethernet and WiFi standards. Mostly open, mostly anyone can present, most stuff presented on the web. ITU – Telephony standards. Closed. Invited by special permission. Only final standard is available (and at a non-trivial cost). Standardize it, build it. ISO and FCC are other possible answers.

## **Problem #2** (10 minutes)

Consider two hosts A and B connected by a single link of rate  $r$  bits/sec. The two hosts are separated by  $d$  meters. Signal propagation is  $p$  meters per second. Host A is sending to host B a packet of size  $S$  bits.

a) What is the propagation delay,  $t_{pr}$ ?

$$t_{pr} = d/p \quad \text{2 pt}$$

b) What is the transmission delay,  $t_{tr}$ , of the packet?

$$t_{tr} = S/r \quad \text{2 pt}$$

c) Ignoring any possible processing and queueing delays, what is the end-to-end delay?

$$(d/p) + (S/r) \quad \text{3 pts}$$

d) If Host A starts to transmit the packet at time  $t = 0$ , where is the packet at time  $t = t_{tr}$ ?

The last bit is just leaving Host A. **3 pts**

e) Suppose that  $t_{pr} > t_{tr}$ . At time  $t = t_{tr}$  where is the first bit of the packet?

The first bit is on the link and has not reached Host B. **3 pts**

f) Suppose that  $t_{pr} < t_{tr}$ . At time  $t = t_{tr}$  where is the first bit of the packet?

The first bit has reached (and likely received by) Host B. **3 pts**

## **Problem #3** (15 minutes)

Answer the following questions regarding the Application Layer.

a) What is HTTP?

**0.5 each and round-up (4 pts total)**

The HyperText Transfer Protocol is a stateless application layer protocol. HTTP is used to transfer web content between a browser application (client) and an HTTP server. All web content is identified by a URL. HTTP is a request response protocol that uses TCP for assured delivery. HTTP uses ASCII encoded headers. The HTTP GET command retrieves HTML files and other objects. The GET header includes the URL of the object and other optional fields such as capability, language, and so on. The response includes a response header with a response code (code 200 is OK and 404 is page not found). Other commands include POST and HEAD.

b) HTTP supports both non-persistent and persistent connections. Describe each type of connection and state which HTTP protocol version supports each type. How is the type of connection specified in the HTTP protocol?

**1 each (4 pts total)**

In a non-persistent connection each HTTP request/response pair has its own TCP connection. A persistent connection supports multiple request/response pairs over a single TCP connection. HTTP 1.0 supports only non-persistent connection, HTTP 1.1 supports both types. The HTTP GET header specifies the type of connection to be used.

c) How is it that sender addresses can be spoofed in an email?

4 pts

There is no authentication of commands in SMTP. Thus anyone with access to an SMTP server can enter any from address (i.e., spoof who they are). The sender address is entered as part of the SMTP MAIL FROM command. There is no authentication or checking of the validity of this address. Thus, any address can be entered, including that of another person.

d) DNS uses a distributed approach as opposed to a single server. Why? If DNS were to “crash” or entirely go down, what would happen? Could the Internet still be “used” and, if so, how?

2 each (4 pts total)

A distributed hierarchy of servers gives better scalability and does not present a single point of failure. If DNS were to crash one could only use IP addresses, and not hostnames, when accessing servers on the Internet.

#### **Problem #4** (10 minutes)

The appendix contains `tcpServer.c` with some “bugs”. Identify the bugs and explain how to fix them. The program compiles successfully. The bugs are, I think, fairly major in scope.

2 each (16 pts total)

The following are the bugs:

- No `WSAStartup()` after line 27
- Wrong protocol family in line 29 (should be streams, not datagram)
- Wrong socket is in bind in line 39 (should be `welcome_s`)
- No `listen()` after line 45
- Only 2 bytes being copied in `memcpy` in line 56 (should be 4 bytes)
- No `ntohs()` in `printf` in line 59
- No “+ 1” for eol character in line 62
- Extra “+ 1” in line 69
- No `closesocket()` for `connect_s` after line 82

#### **Problem #5** (10 minutes)

Answer the following questions about the transport layer:

a) What is the formula for link utilization for the stop and wait protocol? Give the formula in terms of link data rate,  $R$  (b/s), packet size,  $S$  (b), link distance,  $D$  (m), and medium propagation speed,  $p$  (m/s). You may assume that ACK packets and processing delay are negligible.

2 each part (4 pts total)

$$U = t_{fr} / (2t_{pr} + t_{fr}) \text{ where } t_{pr} = D/p \text{ and } t_{fr} = S/R.$$

b) What is a SYN attack? Describe what the attack is and describe what it does (i.e., how it works) to deny service in a server.

1 each (4 pts total)

A SYN attack is where one, or more, malicious hosts send a stream of TCP SYN packets to a server, but with no intention to complete the connection. The server will allocate each resources for each received SYN (in anticipation of a connection to be established), send a SYN-ACK, and wait for an ACK (which will never come). Even a rate of a few SYNs per second can cause a server to consume all of its connection resources and thus prevent legitimate clients (users) from connecting to the server. If legitimate users cannot connect, the server has effectively been taken “offline” and service denied.

c) How does TCP determine the time-out (RTO we have called it) for implicit detection of packet loss? Be brief.

1 each (4 pts total)

An optimum RTO value should be just slightly longer than the pending RTT. TCP samples RTT values (i.e., measures them) and then uses exponential smoothing to get a "smoothed" SRTT value. The SRTT value is then multiplied by a constant to get RTO. Sampling of RTT values only occurs on non-retry transmissions. On retry transmissions, RTO is backed-off (doubled) with each successive time-out until a successful transmission occurs (i.e., an ACK is received). Exponential smoothing is implemented as

$$SRTT(k+1) = \alpha \cdot SRTT(k) + (1-\alpha) \cdot RTT\_sample(k).$$

And then,

$$RTO(k+1) = \beta \cdot SRTT(k+1).$$

d) Assume that you have 10 Mb/s of bandwidth to allocate using proactive flow control. Assume you have five nodes labeled A, B, C, D, and E that request the following amounts of bandwidth, A requests 5 Mb/s, B requests 10 Mb/s, C requests 2 Mb/s, D requests 1 Mb/s, and E requests 50 Mb/s. Using a max-main fair allocation, what amount of bandwidth should be allocated to each node?

2 each round (4 pts total)

First round gives everyone  $10 / 2 = 2$ , so 1, 2, 2, 2, 2 (for D, C, A, B, and E, respectively). This leaves 1 Mb/s leftover to distribute to A, B and E. The final distribution is then 1, 2, 2.33, 2.33, 2.33 for D, C, A, B, and E.

### Problem #6 (10 minutes)

Answer the following questions about the network layer.

a) What is the fundamental purpose or service of this layer?

2 each (4 pts total)

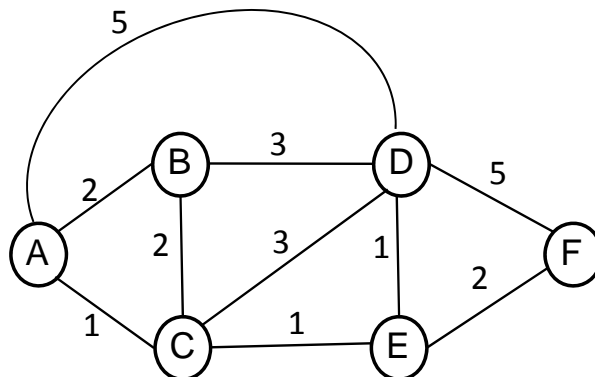
To deliver packets end to end.

b) The purpose of a routing algorithm is to find the least cost path from source to destination. What are some of the possible cost metrics? Which metrics could cause stability problems (i.e., oscillation or route flapping)? Why?

2 each (6 pts total)

Cost metrics include hops, link capacity, throughput, congestion, and delay. Any metric that is a measurement of a realtime function of traffic is prone to causing stability problems. This is because the routing algorithm directly affects these measures.

c) For the six node network topology given below with link costs as shown, find the shortest path from node A to node F using Dijkstra's algorithm. Clearly describe the order of links as they are added one-by-one by the algorithm and give the path cost from node A to the added node.



1 each step (6 pts total)

One possible solution is:

- AC (added cost = 1, path cost from A to C = 1)
- CE (added cost = 1, path cost from A to E = 2)
- AB (added cost = 2, path cost from A to B = 2)
- ED (added cost = 1, path cost from A to D = 3)
- EF (added cost = 2, path cost from A to E = 4)

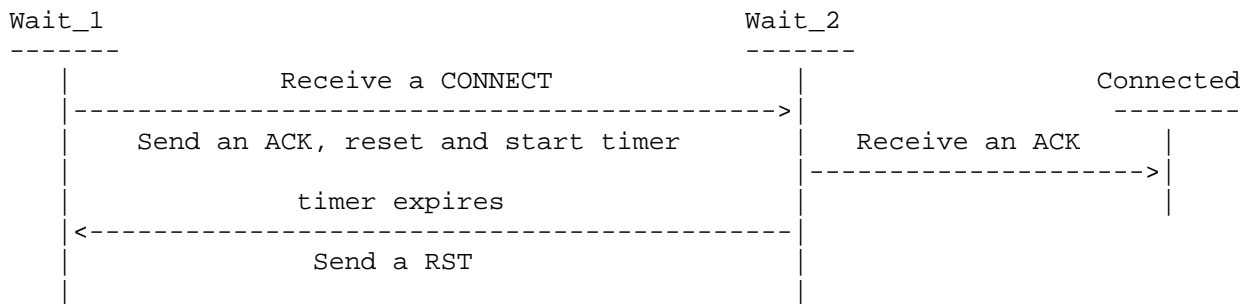
Another possible solution is:

- AC (added cost = 1, path cost from A to C = 1)
- AB (added cost = 2, path cost from A to B = 2)
- CE (added cost = 1, path cost from A to E = 2)
- ED (added cost = 1, path cost from A to D = 3)
- EF (added cost = 2, path cost from A to E = 4)

**Extra Credit** (10 minutes)

Sketch the FSM for the following connection establishment protocol between a sender and a receiver. When the receiver receives a CONNECT packet from the sender it replies to the sender with an ACK. The receiver then waits for an ACK packet from the sender. If an ACK is not received within 10 seconds, the receiver sends an RST and waits for another SYN packet, otherwise it enters a connected state

1 for each state, 1 for timer, and 2 for each transition (10 pts total)



Note: The timer is set to 10 seconds on reset

## Appendix – Program tcpServer.c with some bugs

```
1.  //===== file = tcpServer.c =====
2.  // = A message "server" program to demonstrate sockets programming =
3.  //=====
4.  // = - Modified for exam #1 to include some bugs for the students to find =
5.  //=====
6.  //----- Include files -----
7.  #include <stdio.h>           // Needed for printf()
8.  #include <string.h>         // Needed for memcpy() and strcpy()
9.  #include <windows.h>        // Needed for all Winsock stuff

10. //----- Defines -----
11. #define PORT_NUM    1050    // Arbitrary port number for the server

12. //===== Main program =====
13. void main(void)
14. {
15.     #ifdef WIN
16.     WORD wVersionRequested = MAKEWORD(1,1);           // Stuff for WSA functions
17.     WSADATA wsaData;                                   // Stuff for WSA functions
18.     #endif
19.     int welcome_s;                                     // Welcome socket descriptor
20.     struct sockaddr_in server_addr;                   // Server Internet address
21.     int connect_s;                                    // Connection socket descriptor
22.     struct sockaddr_in client_addr;                   // Client Internet address
23.     struct in_addr client_ip_addr;                   // Client IP address
24.     int addr_len;                                     // Internet address length
25.     char out_buf[4096];                               // Output buffer for data
26.     char in_buf[4096];                               // Input buffer for data
27.     int retcode;                                      // Return code

28.     // Create a socket
29.     welcome_s = socket(AF_INET, SOCK_DGRAM, 0);
30.     if (welcome_s < 0)
31.     {
32.         printf("*** ERROR - socket() failed \n");
33.         exit(-1);
34.     }

35.     // Fill-in server (my) address information and bind the socket
36.     server_addr.sin_family = AF_INET;
37.     server_addr.sin_port = htons(PORT_NUM);
38.     server_addr.sin_addr.s_addr = htonl(INADDR_ANY);
39.     retcode = bind(connect_s, (struct sockaddr *)&server_addr,
40.     sizeof(server_addr));
41.     if (retcode < 0)
42.     {
43.         printf("*** ERROR - bind() failed \n");
44.         exit(-1);
45.     }

46.     // Accept a connection.
47.     printf("Waiting for accept() to complete... \n");
48.     addr_len = sizeof(client_addr);
49.     connect_s = accept(welcome_s, (struct sockaddr *)&client_addr, &addr_len);
```

```
50.  if (connect_s < 0)
51.  {
52.      printf("*** ERROR - accept() failed \n");
53.      exit(-1);
54.  }

55.  // Copy the four-byte client IP address into an IP address structure
56.  memcpy(&client_ip_addr, &client_addr.sin_addr.s_addr, 2);

57.  // Print an informational message that accept completed
58.  printf("Accept completed (IP address of client = %s port = %d) \n",
59.      inet_ntoa(client_ip_addr), client_addr.sin_port);

60.  // Send to the client using the connect socket
61.  strcpy(out_buf, "This is a message from SERVER to CLIENT");
62.  retcode = send(connect_s, out_buf, strlen(out_buf), 0);
63.  if (retcode < 0)
64.  {
65.      printf("*** ERROR - send() failed \n");
66.      exit(-1);
67.  }

68.  // Receive from the client using the connect socket
69.  retcode = recv(connect_s, in_buf, (sizeof(in_buf) + 1), 0);
70.  if (retcode < 0)
71.  {
72.      printf("*** ERROR - recv() failed \n");
73.      exit(-1);
74.  }
75.  printf("Received from client: %s \n", in_buf);

76.  // Close the welcome and connect sockets
77.  retcode = closesocket(welcome_s);
78.  if (retcode < 0)
79.  {
80.      printf("*** ERROR - closesocket() failed \n");
81.      exit(-1);
82.  }

83.  // Clean-up winsock
84.  WSACleanup();
85. }
```