Mid-Term Exam for Computer Networks Spring 2006

>>> SOLUTION <<<

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 writing your name on the exam). There is also an additional extra credit question worth 10 points. You may have with you a calculator, pencils, blank paper, lucky rabbit's foot, 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 text are not allowed on this sheet. You have 75 minutes for the exam. Please use a separate sheet of paper for each question. Read the last sentence again. 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.

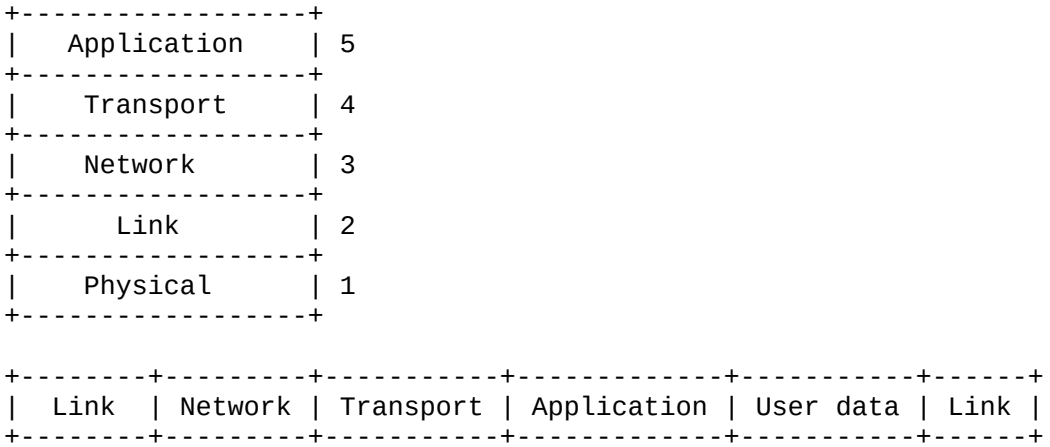
a) What are the fundamental measures of interest for a communications system?

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

b) What are the basic tasks or functions of a communications system?

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

c) In this class (using the Kurose book) we have a five-layer protocol model. Sketch this model and also sketch the associated packet (showing headers and trailers).



d) Precisely define protocol and interface (as we described them in class).

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.

Problem #2 (10 minutes)

Answer the following questions regarding the Internet.

a) What are the four causes of packet delay?

Processing, transmission, propagation, and queueuing.

b) What are the possible mechanisms of packet loss?

Buffer overflow (e.g., due to congestion) and electrical noise corrupting a packet

c) What is a tool that can be used to determine the number of hops to a destination and the round trip time (RTT) for each hop?

traceroute

d) California is about 3000 miles from here. What is (approximately) the minimum possible RTT? Why would the typical RTT be greater than this minimum? Show your work.

Propagation is about 1 nanosecond per foot, so about 5 microseconds per mile. This results in an about 30 ms minimum RTT. In reality, RTT would be greater due to having to travel through multiple hops and thus also experience multiple transmission delays and possible queueing delays.

e) What is quality of service?

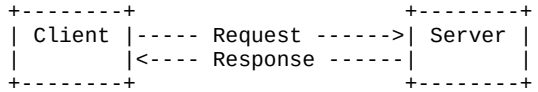
QoS is the requirements an application places on the network.

Problem #3 (10 minutes)

Answer the following questions regarding the Application Layer.

a) What is HTTP (in about 100 words, use a drawing if it will help)?

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

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).

c) We say that FTP has out-of-band control. What do we mean by that?

We mean that commands and data flow across different TCP connections.

Problem #4 (10 minutes)

Answer the following question regarding sockets programming:

Attached are `server.c` and `client.c` with some "bugs". Identify the bugs and explain how to fix them.

Server program:

- Missing welcome_s = socket() after line 25
- Missing htons() in line 28
- Missing bind() after line 29
- Listen must have more than 0 connections in line 31
- Missing addr_len assignment after line 32
- Missing +1 for strlen for \0 in line 38
- Should recv() on connect_s in line 38
- Should send() on connect_s in line 38
- Missing closesocket() of welcome_s after line 42

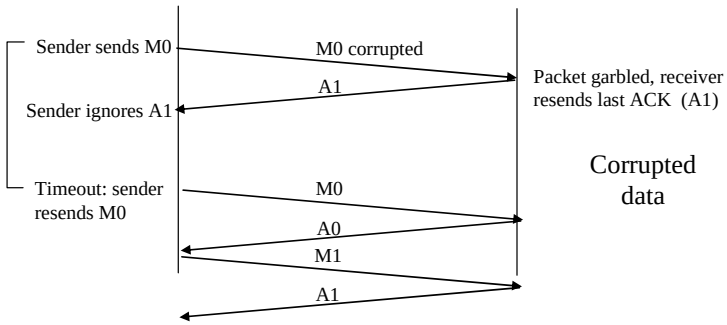
Client program:

- Type of server_addr should be struct sickaddr_in in line 18
- Size of out_buf and in_buf is too small in lines 19 and 20
- Should be SOCK_STREAM and not DATAGRAM in line 24
- Missing htons() in line 28
- Missing +1 for strlen for \0 in line 36

Problem #5 (15 minutes)

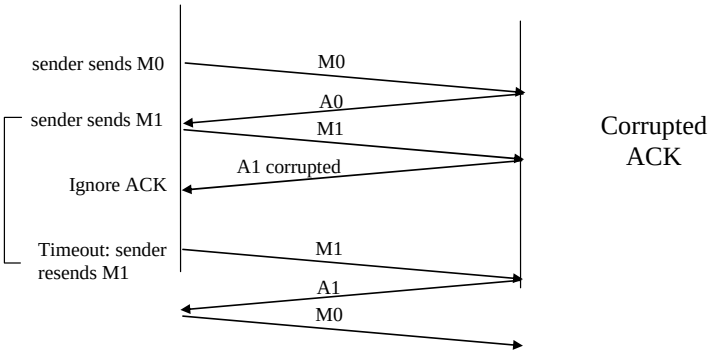
Answer the following questions regarding reliable data transfer and its performance:

a) Give a trace (a packet timing diagram) of the operation of RDT 3.0 when data packets and acknowledgement packets are garbled (corrupted). Suppose the protocol has been in operation for some time. The RDT 3.0 sender and RDT 2.2 receiver FSMs are attached at the end of this exam.



Corrupted data

We can suppose that the sender is in state "Wait for call from above" (top left hand corner) and the receiver is in state "Wait for 0 from below". The scenarios for corrupted data and corrupted ACK are shown in the following figure



Corrupted ACK

b) Assume that you have a 1000 mile long 1 Mb/s link. For a packet length of 1500 bytes, what is the link utilization if stop-and-wait protocol is used? What if sliding window protocol is used with a window size of 10? You may assume that the sender always has packets to send and that no packets are lost or corrupted.

1000 miles is about 5 milliseconds propagation (for 1 nanosecond per foot). The packet transmission time is 12 milliseconds. So, U_saw = (12 / (12 + 2x5)) = 54.5%. If the window size is 10, then clearly the utilization will be 100% because (10x12) > 12 + 2x5.

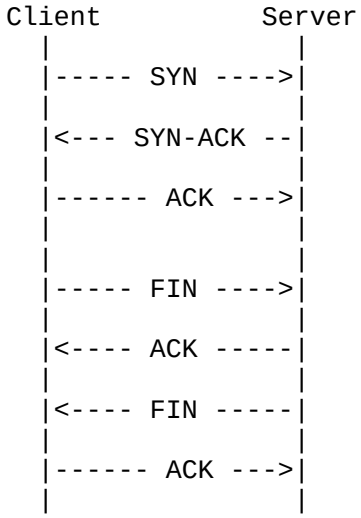
Problem #6 (15 minutes)

Answer the following questions regarding TCP, UDP, IP, and routing:

a) What are the important fields in the TCP, UDP, and IP headers? Without these fields, the protocols would clearly not "work".

TCP = ports #s, seq/ack #s, flags, checksum, and windows size
UDP = port #s and checksum
IP = addresses, checksum for header

b) Sketch the TCP connection initiation and connection termination packet flows using a timing diagram.



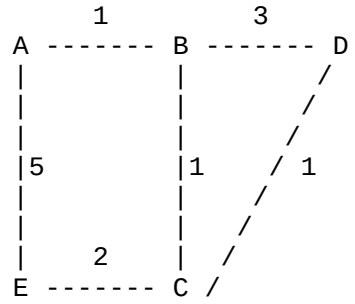
c) What are the important attributes for a good routing algorithm?

Fast, simple, little traffic, no loops, converge to optimum

d) What is a key difference between a distance vector and link state routing protocol?

DV uses neighbor exchange of routing vectors, LS uses broadcast of link state information.

e) For the following five node network topology with link costs as shown, find the shortest path from node A to all other nodes using Dijkstra's algorithm. Clearly describe the order of links as they as added one-by-one by the algorithm and give the path costs.



Order of edges added and cost of new path added
  AB at cost 1
  AB, BC at cost 2
  AB, BC, CD at cost 3
  AB, BC, CD, CE at cost 4

Extra Credit (5 minutes)

Consider 5 users (call them A, B, C, D, and E) that have connections on a single 10 Mb/s link. Connections last for several minutes or perhaps even hours (i.e., they are bounded in time). Assume that user A requests 2 Mb/s for its connection, B requests 1 Mb/s, C requests 3 Mb/s, D requests 4 Mb/s, and E request 10 Mb/s. Note that the total requested bandwidth (20 Mb/s) exceeds the link bandwidth (10 Mb/s). Describe at least three ways of fairly allocating the 10 Mb/s to the five users. Carefully discuss/describe what is "fair". Carefully discuss the trade-offs between your different definitions of "fair". Multiple definitions of fair are possible.

One definition of fair can be to allocate resource proportional to need. So,

A gets (2/20)(10) Mb/s = 1.0 Mb/s
B gets (1/20)(10) = 0.5
C gets (3/20)(10) = 1.5
D gets (4/20)(10) = 2.0
E get (10/20)(10) = 5.0

This definition of fairness rewards over-requesting (cheating) to get a larger share against other users. We note that in this example, no user is 100% happy (i.e., has the full allocation that it requested).

Max-min allocation is another definition of fairness. So,

All get 2 Mb/s in a first round with A satisfied, B satisfied and with 1 Mb/s to give back, C, D, and E all with bandwidth still needed. The 1 Mb/s is then allocated between C, D, and E (each get 1/3 Mb/s) such that in the end we have

A gets 2 Mb/s
B gets 1 Mb/s
C gets 2.33 Mb/s
D gets 2.33 Mb/s
E gets 2.33 Mb/s

This definition of fairness does not reward cheating (asking for more than you need) and does also allow some users to be 100% happy. This definition of fairness, however, does not fully consider need (i.e., the user requesting the most does not necessarily get the most).

Finally, a random allocation of the entire link to a user can be long term fair, but inefficient (and not short term fair either). So, first give A all its wants. When A is done, give B all it wants, and so on. This method also does not reward cheating since a connection simply uses all it needs and requesting an excess has no benefit.

```
1.    //======================================================== file = server.c =====
2.    //=  A message "server" program to demonstrate sockets programming           =
3.    //=   - TCP/IP client/server model is implemented                             =
4.    //=============================================================================

5.    //----- Include files ---------------------------------------------------------
6.    #include <stdio.h>          // Needed for printf()
7.    #include <string.h>         // Needed for memcpy() and strcpy()
8.    #include <windows.h>        // Needed for all Winsock stuff

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

11.   //===== Main program ==========================================================
12.   void main(void)
13.   {
14.   WORD wVersionRequested = MAKEWORD(1,1);       // Stuff for WSA functions
15.   WSADATA wsaData;                              // Stuff for WSA functions
16.   unsigned int         welcome_s;       // Welcome socket descriptor
17.   struct sockaddr_in   server_addr;     // Server Internet address
18.   unsigned int         connect_s;       // Connection socket descriptor
19.   struct sockaddr_in   client_addr;     // Client Internet address
20.   struct in_addr       client_ip_addr;  // Client IP address
21.   int                  addr_len;        // Internet address length
22.   char                 out_buf[100];    // Output buffer for data
23.   char                 in_buf[100];     // Input buffer for data

24.   // This stuff initializes winsock
25.   WSAStartup(wVersionRequested, &wsaData);

26.   // Fill-in server (my) address information and bind the welcome socket
27.   server_addr.sin_family = AF_INET;
28.   server_addr.sin_port = PORT_NUM;
29.   server_addr.sin_addr.s_addr = htonl(INADDR_ANY);

30.   // Listen on welcome socket for a connection
31.   listen(welcome_s, 0);

32.   // Accept a connection.
33.   connect_s = accept(welcome_s, (struct sockaddr *)&client_addr, &addr_len);

34.   // Print an informational message that accept completed
35.   printf("Accept completed \n”);

36.   // Send to the client using the connect socket
37.   strcpy(out_buf, "Test message from server to client");
38.   send(welcome_s, out_buf, strlen(out_buf), 0);

39.   // Receive from the client using the connect socket
40.   recv(welcome_s, in_buf, sizeof(in_buf), 0);
41.   printf("Received from client... data = '%s' \n", in_buf);

42.   // Close sockets and clean-up
43.   closesocket(connect_s);
44.   WSACleanup();
45.   }
```

```
1.    //======================================================== file = client.c =====
2.    //=  A message "client" program to demonstrate sockets programming           =
3.    //=   - TCP/IP client/server model is implemented                             =
4.    //=============================================================================

5.    //----- Include files ---------------------------------------------------------
6.    #include <stdio.h>          // Needed for printf()
7.    #include <string.h>         // Needed for memcpy() and strcpy()
8.    #include <windows.h>        // Needed for all Winsock stuff

9.    //----- Defines ---------------------------------------------------------------
10.   #define  PORT_NUM         1050   // Port number used at the server
11.   #define  IP_ADDR    "127.0.0.1"  // IP address of server (*** HARDWIRED ***)

12.   //===== Main program ==========================================================
13.   void main(void)
14.   {
15.   WORD wVersionRequested = MAKEWORD(1,1);       // Stuff for WSA functions
16.   WSADATA wsaData;                              // Stuff for WSA functions
17.   unsigned int         client_s;        // Client socket descriptor
18.   double               server_addr;     // Server Internet address
19.   char                 out_buf[10];     // Output buffer for data
20.   char                 in_buf[10];      // Input buffer for data

21.   // This stuff initializes winsock
22.   WSAStartup(wVersionRequested, &wsaData);

23.   // Create a client socket
24.   client_s = socket(AF_INET, DATAGRAM, 0);

25.   // Fill-in the server's address information and do a connect with the
26.   // listening server
27.   server_addr.sin_family = AF_INET;
28.   server_addr.sin_port = PORT_NUM;
29.   server_addr.sin_addr.s_addr = inet_addr(IP_ADDR);
30.   connect(client_s, (struct sockaddr *)&server_addr, sizeof(server_addr));

31.   // Receive from the server using the client socket
32.   recv(client_s, in_buf, sizeof(in_buf), 0);
33.   printf("Received from server... data = '%s' \n", in_buf);

34.   // Send to the server using the client socket
35.   strcpy(out_buf, "Test message from client to server");
36.   send(client_s, out_buf, strlen(out_buf), 0);

37.   // Close and clean-up
38.   closesocket(client_s);
39.   WSACleanup();
40.   }
```

FSMs for RDT 3.0 server and RDT 2.2 receiver from Kurose

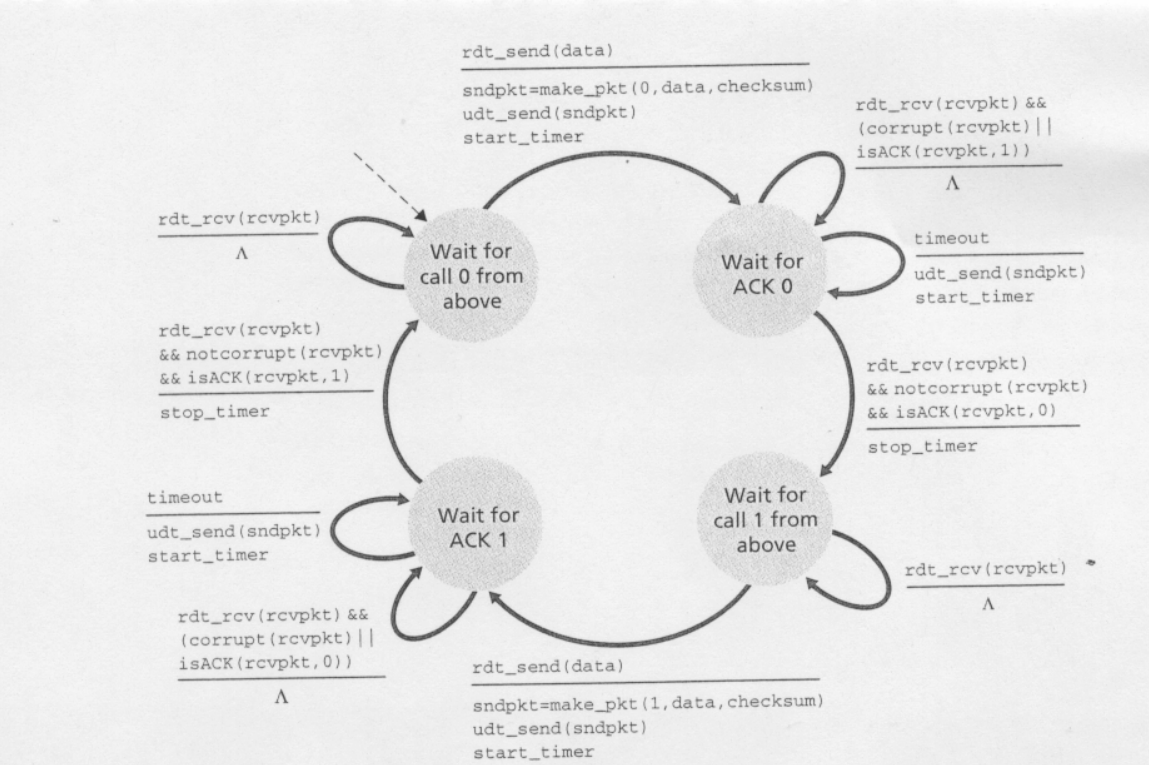


Figure 3.15 ♦ rdt3.0 sender

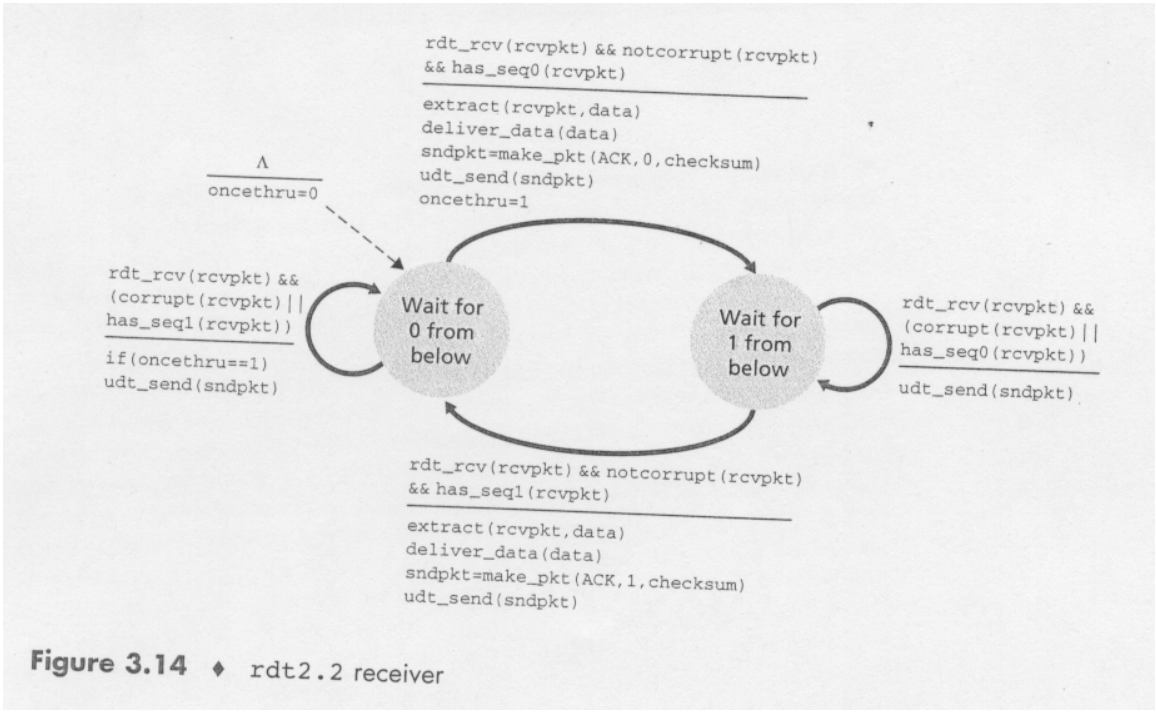


Figure 3.14 ♦ rdt2.2 receiver