Computer Networks
Problem Set 6

Instructor: Shashi Prabh

Chapter 6. The Transport Layer

6.1 In our example transport primitives of Figure 1 (Figure 6-2 in the textbook), `LISTEN` is a blocking call. Is this strictly necessary? If not, explain how a nonblocking primitive could be used. What advantage would this have over the scheme described in the text?

Primitive	Packet sent	Meaning
LISTEN	(none)	Block until some process tries to connect
CONNECT	CONNECTION REQ.	Actively attempt to establish a connection
SEND	DATA	Send information
RECEIVE	(none)	Block until a DATA packet arrives
DISCONNECT	DISCONNECTION REQ.	Request a release of the connection

Figure 1: Exercise 6.1

6.2 Primitives of the transport service assume asymmetry between the two end points during connection establishment: one end (server) executes `LISTEN` while the other end (client) executes `CONNECT`. However, in peer-to-peer applications such as file sharing systems, e.g., BitTorrent, all end points are peers. There is no server or client functionality. How can transport service primitives be used to build such peer-to-peer applications?

6.3 A chat application using TCP repeatedly calls `receive()`, and prints the received data as a new message. Can you think of a problem with this approach?

6.4 In both parts of Figure 2 and Figure 3 (Figure 6-6 in the textbook), there is a comment that the value of `SERVER_PORT` must be the same in both client and server. Why is this so important?

```
/* This page contains a client program that can request a file from the server program
 * on the next page. The server responds by sending the whole file.
 */

#include <sys/types.h>
#include <unistd.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

#define SERVER_PORT 8080                /* arbitrary, but client & server must agree */
#define BUF_SIZE 4096                   /* block transfer size */

int main(int argc, char **argv)
{
  int c, s, bytes;
  char buf[BUF_SIZE];                   /* buffer for incoming file */
  struct hostent *h;                    /* info about server */
  struct sockaddr_in channel;           /* holds IP address */

  if (argc != 3) {printf("Usage: client server-name file-name0); exit(-1);}
  h = gethostbyname(argv[1]);           /* look up host's IP address */
  if (!h) {printf("gethostbyname failed to locate %s0, argv[1]); exit(-1;}

  s = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
  if (s <0) {printf("socket call failed0); exit(-1);}
  memset(&channel, 0, sizeof(channel));
  channel.sin_family= AF_INET;
  memcpy(&channel.sin_addr.s_addr, h->h_addr, h->h_length);
  channel.sin_port= htons(SERVER_PORT);
  c = connect(s, (struct sockaddr *) &channel, sizeof(channel));
  if (c < 0) {printf("connect failed0); exit(-1);}

  /* Connection is now established. Send file name including 0 byte at end. */
  write(s, argv[2], strlen(argv[2])+1);

  /* Go get the file and write it to standard output. */
  while (1) {
      bytes = read(s, buf, BUF_SIZE);   /* read from socket */
      if (bytes <= 0) exit(0);          /* check for end of file */
      write(1, buf, bytes);             /* write to standard output */
  }
}
```

Figure 2: Exercise 6.4

```
#include <sys/types.h>                  /* This is the server code */
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/fcntl.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

#define SERVER_PORT 8080                /* arbitrary, but client & server must agree */
#define BUF_SIZE 4096                   /* block transfer size */
#define QUEUE_SIZE 10

int main(int argc, char *argv[])
{ int s, b, l, fd, sa, bytes, on = 1;
  char buf[BUF_SIZE];                   /* buffer for outgoing file */
  struct sockaddr_in channel;           /* holds IP address */

  /* Build address structure to bind to socket. */
  memset(&channel, 0, sizeof(channel)); /* zero channel */
  channel.sin_family = AF_INET;
  channel.sin_addr.s_addr = htonl(INADDR_ANY);
  channel.sin_port = htons(SERVER_PORT);

  /* Passive open. Wait for connection. */
  s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);   /* create socket */
  if (s < 0) {printf("socket call failed0); exit(-1);}
  setsockopt(s, SOL_SOCKET, SO_REUSEADDR, (char *) &on, sizeof(on));

  b = bind(s, (struct sockaddr *) &channel, sizeof(channel));
  if (b < 0) {printf("bind failed0); exit(-1);}

  l = listen(s, QUEUE_SIZE);            /* specify queue size */
  if (l < 0) {printf("listen failed0); exit(-1);}

  /* Socket is now set up and bound. Wait for connection and process it. */
  while (1) {
      sa = accept(s, 0, 0);             /* block for connection request */
      if (sa < 0) {printf("accept failed0); exit(-1);}
      read(sa, buf, BUF_SIZE);          /* read file name from socket */

      /* Get and return the file. */
      fd = open(buf, O_RDONLY);         /* open the file to be sent back */
      if (fd < 0) {printf("open failed");

      while (1) {
          bytes = read(fd, buf, BUF_SIZE); /* read from file */
          if (bytes <= 0) break;        /* check for end of file */
          write(sa, buf, bytes);        /* write bytes to socket */
      }
      close(fd);                        /* close file */
      close(sa);                        /* close connection */
  }
}
```

Figure 3: Exercise 6.4

6.5 In the Internet File Server example igure 2 and Figure 3 (Figure 6-6 in the textbook), can the `connect()` system call on the client fail for any reason other than listen queue being full on the server? Assume that the network is perfect.

6.6 Suppose that the clock-driven scheme for generating initial sequence numbers is used with a 15-bit wide clock counter. The clock ticks once every 100 msec, and the maximum packet lifetime is 60 sec. How often need resynchronization take place

a. in the worst case?

b. when the data consumes 240 sequence numbers/min?

6.7 Why does the maximum packet lifetime, T, have to be large enough to ensure that not only the packet but also its acknowledgements have vanished?

6.8 Consider a connection-oriented transport layer protocol that uses a time-of-day clock to determine packet sequence numbers. The clock uses a 9-bit counter, and ticks once every 250 msec. The maximum packet lifetime is 32 seconds. If the sender sends 3 packets per second, how long could the connection last without entering the forbidden region?

6.9 Imagine that a two-way handshake rather than a three-way handshake were used to set up connections. In other words, the third message was not required. Are deadlocks now possible? Give an example or show that none exist.

6.10 Imagine a generalized n-army problem, in which the agreement of any two of the blue armies is sufficient for victory. Does a protocol exist that allows blue to win?

6.11 In Figure 4 (Figure 6-20 in the textbook), suppose the flows are rearranged such that A goes through $R1, R2, R3$, B goes through $R1, R5, R6$, C goes through $R4, R5, R6$, and D goes through $R4, R5, R6$. What is the max-min bandwidth allocation?

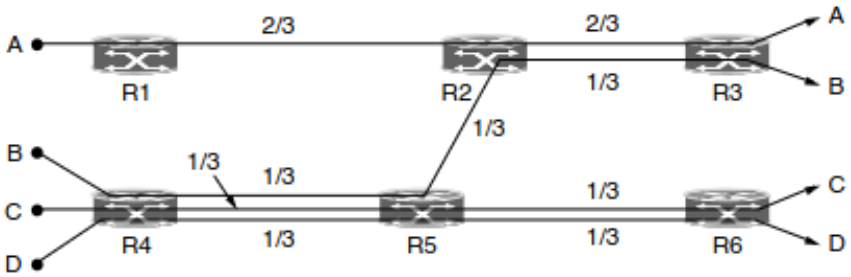

Figure 4: Exercise 6.11

6.12 Discuss the advantages and disadvantages of credits versus sliding window protocols.

6.13 Some other policies for fairness in congestion control are Additive Increase Additive Decrease (AIAD), Multiplicative Increase Additive Decrease (MIAD), and Multiplicative Increase Multiplicative Decrease (MIMD). Discuss these three policies in terms of convergence and stability.

6.14 Two hosts simultaneously send data through a network with a capacity of 1 Mbps. Host A uses UDP and transmits a 100 bytes packet every 1 msec. Host B generates data with a rate of 600 kbps and uses TCP. Which host will obtain higher throughput?

6.15 Why does UDP exist? Would it not have been enough to just let user processes send raw IP packets?

6.16 Both UDP and TCP use port numbers to identify the destination entity when delivering a message. Give two reasons why these protocols invented a new abstract ID (port numbers), instead of using process IDs, which already existed when these protocols were designed.

6.17 Datagram fragmentation and reassembly are handled by IP and are invisible to TCP. Does this mean that TCP does not have to worry about data arriving in the wrong order?

6.18 RTP is used to transmit CD-quality audio, which makes a pair of 16-bit samples 44,100 times/sec, one sample for each of the stereo channels. How many packets per second must RTP transmit?

6.19 The maximum payload of a TCP segment is 65,495 bytes. Why was such a strange number chosen?

6.20 Consider the effect of using slow start on a line with a 10-msec round-trip time and no congestion. The receive window is 24 KB and the maximum segment size is 2 KB. How long does it take before the first full window can be sent?

6.21 Suppose that the TCP congestion window is set to 18 KB and a timeout occurs. How big will the window be if the next four transmission bursts are all successful? Assume that the maximum segment size is 1 KB.

6.22 If the TCP round-trip time, RTT, is currently 30 msec and the following acknowledgements come in after 26, 32, and 24 msec, respectively, what is the new RTT estimate using the Jacobson algorithm? Use $\alpha = 0.9$.

6.23 For a 1-Gbps network operating over 4000 km, the delay is the limiting factor, not the bandwidth. Consider a MAN with the average source and destination 20 km apart. At what data rate does the round-trip delay due to the speed of light equal the transmission delay for a 1-KB packet?