# CS7344 Homework 5

Name: Bingying Liang
ID: 48999397

Dec 2 2023

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

   **Solution:**
   **The reasons for the existence of UDP:** UDP (User Datagram Protocol) exists because it is a lightweight, efficient protocol ideal for applications where speed and low latency are crucial. It's used for tasks where fast, real-time communication is more important than accuracy, such as in live video streaming or online gaming. UDP skips the overhead of connection setup and reliability checks, making it faster but less reliable than TCP (Transmission Control Protocol). The following text provides some detailed reasons:

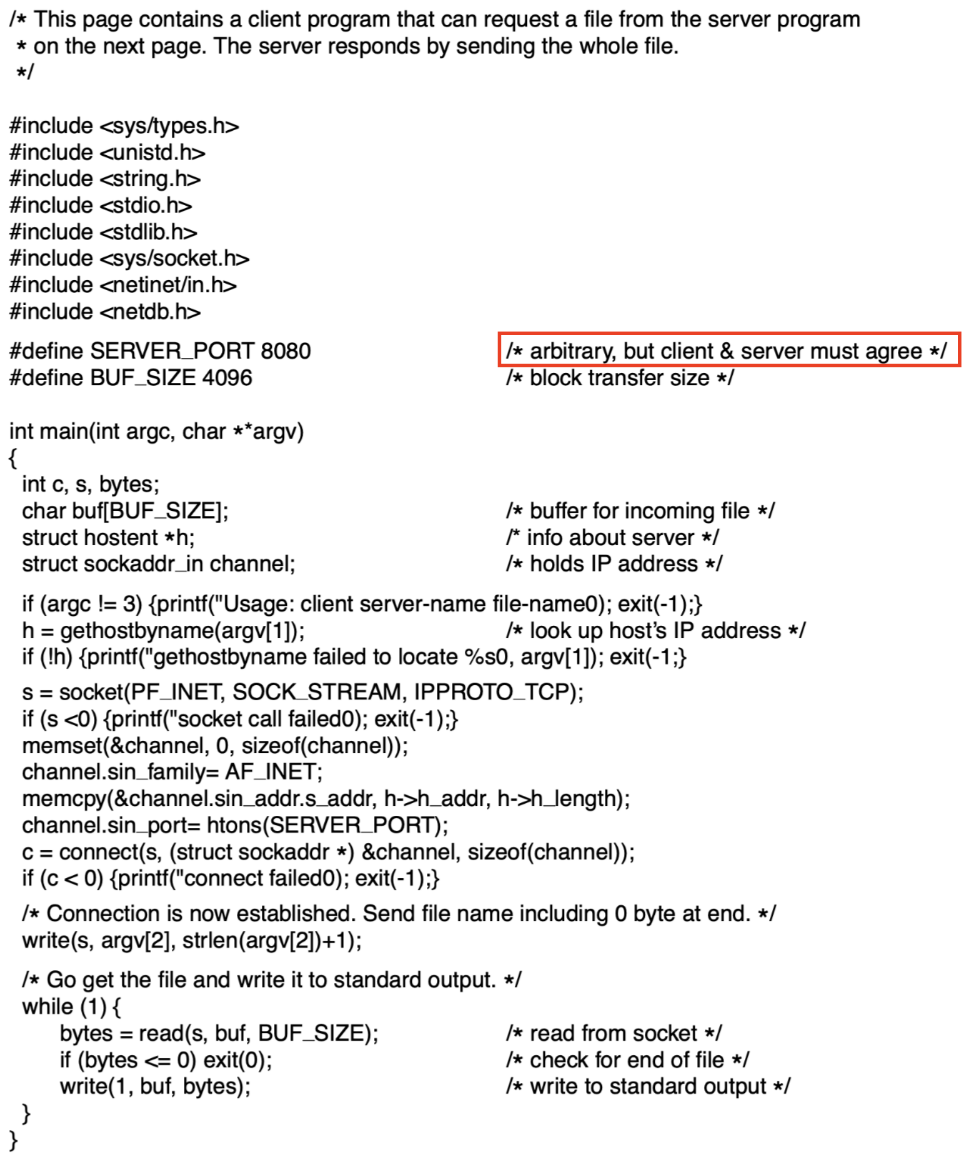
   (a) Simplicity and Efficiency: UDP is a minimal, connectionless protocol that doesn't require a formal connection to send data. This simplicity translates into speed, making it more efficient for certain types of communications, especially those that require small data packets sent over the network quickly.

   (b) Low Overhead: Unlike TCP (Transmission Control Protocol), UDP doesn't have mechanisms for error checking, retransmission, or flow control. This means there's less overhead in terms of data packet size and processing requirements. For applications that can tolerate some loss of data but need fast transmission, UDP is ideal.

   (c) Real-time Applications: UDP is well-suited for real-time applications like video streaming, VoIP (Voice over IP), and online gaming. These applications can tolerate some loss of data packets but require low latency and fast transmission speeds. UDP's lack of retransmission for lost packets means that it can maintain a steady flow of data, which is crucial for these time-sensitive applications.

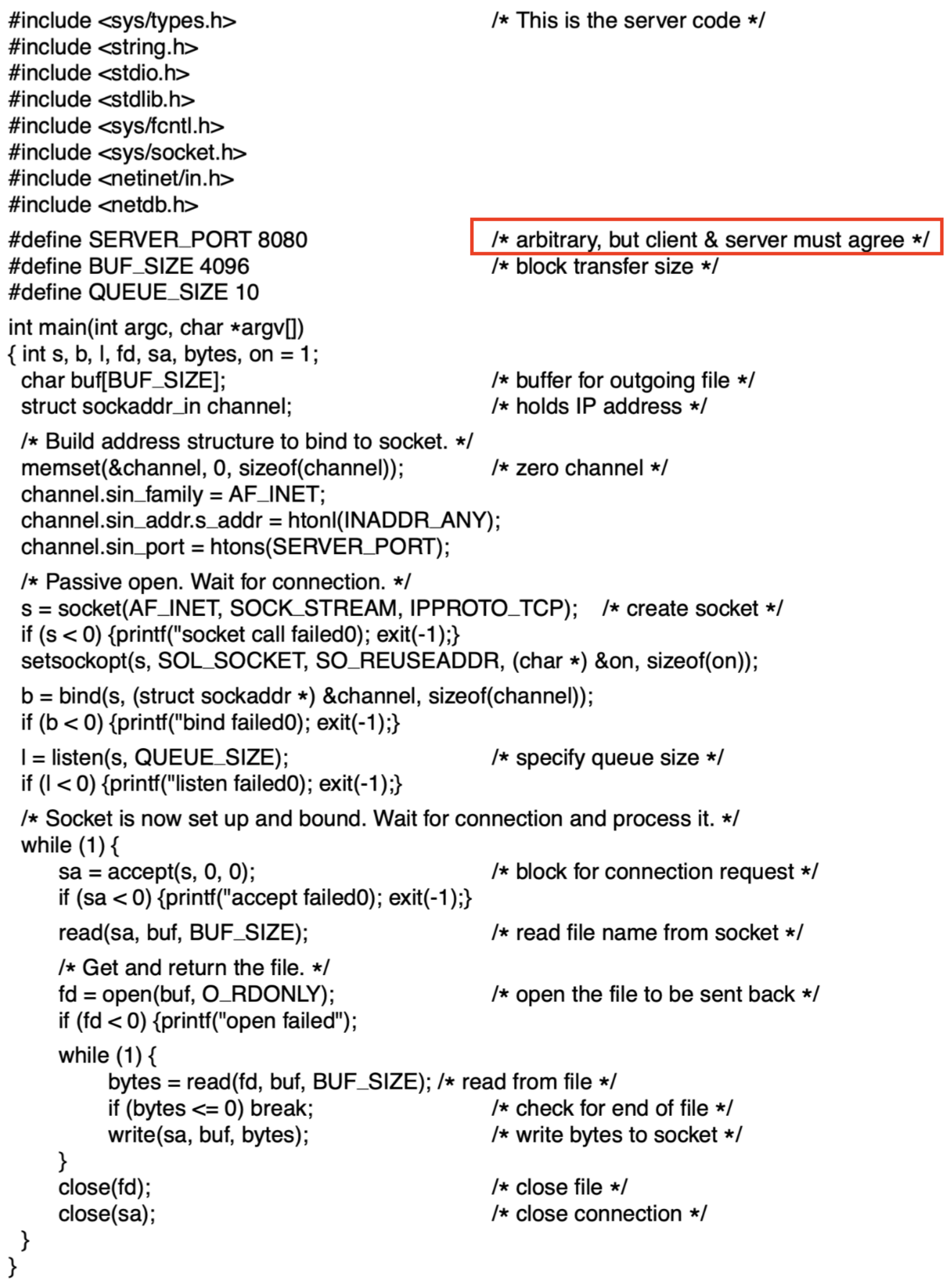
   **Second question:** It will not be enough to just let user processes send raw IP packets. The reasons is in the following: IP packets contain IP addresses, which specify a destination machine. Once such a packet arrived, how would the network handler know which process to give it to? UDP packets contain a destination port. This information is essential so they can be delivered to the correct process. And allowing user processes to send raw IP packets instead of using protocols like UDP would increase complexity for developers, pose security risks, and complicate port multiplexing and network traffic management. Transport protocols like UDP and TCP provide standardized, efficient ways to handle network communications, ensuring interoperability and simplifying the development of networked applications.

2. In both parts of Fig. 6-6. in your 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 6-6.** Client code using sockets. The server code is on the next page.



```
#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 */
  }
}
```

**Solution:**
If the client sends a packet to SERVER_PORT and the server is not listening to that port, the packet will not be delivered to the server. The server port must be the same on both client and server for effective communication because it ensures that messages are correctly routed to the intended service on the server. This consistency is key for proper network functioning, security, and avoiding conflicts with other services. Without using the correct server port, clients cannot successfully connect to or communicate with the server's services.

3. For each of the following applications, tell whether it would be (1) possible and (2) better to

use a PHP script or JavaScript, and why:

(a) Displaying a calendar for any requested month since September 1752.

(b) Displaying the schedule of flights from Amsterdam to New York.

(c) Graphing a polynomial from user-supplied coefficients.

**Solution:**

(a) JavaScript. Because there are only 14 annual calendars, depending on the day of the week on which 1 January falls and whether the year is a leap year. Thus, a JavaScript program could easily contain all 14 calendars and a small database of which year gets which calendar. A PHP script could also be used, but it would be slower.

(b) PHP. Because this requires a large database. It must be done on the server by using PHP.

(c) Both work, but JavaScript is faster.

4. You request a Web page from a server. The server's reply includes an Expires header, whose value is set to one day in the future. After five minutes, you request the same page from the same server. Can the server send you a newer version of the page? Explain why or why not?

**Solution:**
Yes, the server can send you a newer version of the page even if the Expires header is set to one day in the future. The Expires header is primarily a directive for caches, not for servers. It tells the browser and intermediate caches (like proxy servers) how long the response is considered fresh, i.e., for how long it can be served from the cache without checking with the server. Here are key points to consider:

(a) Client-Side Caching: The Expires header is a way for the server to inform the client (and any intermediate caches) about how long the content is considered valid. If the client requests the same resource within the time frame where the content is considered fresh (one day, in your example), the browser might not contact the server and may instead load the page from its cache.

(b) Server Discretion: The server has complete discretion over what content it serves. If you request the same page after five minutes, the server can choose to ignore the previously set Expires header and send a newer version of the page. The Expires header does not bind the server to serve the same content until it expires; it's more about guiding the behavior of the client and intermediate caches.

(c) Conditional Requests: If the client or a proxy makes a conditional request (using headers like If-Modified-Since or If-None-Match), the server can evaluate whether the content has changed since the last request. If it has, the server can send the new content regardless of the Expires header.

(d) Dynamic Content and Server Configuration: Modern web applications often serve dynamic content, which can change frequently. The server configuration and the application logic can determine how to handle requests for such content, potentially providing updated content even if the Expires time has not been reached.