ITS 332 Networking Lab

UDP Client/Server Programming

Dr Steven Gordon

6 November 2009

1 Introduction

There are two main transport protocols used in the Internet, and corresponding to each of them is a socket type:

1. Stream sockets use the Transmission Control Protocol (TCP) to communicate. TCP is stream-oriented, sending a stream of bytes to the receiver. It is also a reliable transport protocol, which means it guarantees that all data arrives at the receiver, and arrives in order. TCP starts be setting up a connection (we have seen the 3-way handshake in other labs), and then sending data between sender and receiver. TCP is used for most data-oriented applications like web browsing, file transfer and email.
2. Datagram sockets use the User Datagram Protocol (UDP) to communicate. UDP is an unreliable protocol. There is no connection setup or retransmissions. The sender simply sends a packet (datagram) to the receiver, and hopes that it arrives. UDP is used for most real-time oriented applications like voice over IP and video conversations.

In the previous lab we created a client/server program using TCP (stream sockets) – in this lab you will create simple client/server program using UDP (datagram sockets).

With UDP, as there is no connection setup (e.g. no 3-way handshake), then the socket coding is much simpler:

- At both the client and server you must create the socket using the `socket()` system call;
- At the server you must bind the socket to the server address using the `bind()` system call;
- `sendto()` and `recvfrom()` system calls are then used to send data to a specific address and received data from a socket, respectively.

We must specify a destination address in the `sendto()` system call because there is no destination associated with a UDP socket. (Recall in TCP or stream sockets, when we create a socket we associate a destination address with it – therefore whenever we send to the socket, the destination address is known).

Remember that UDP is *unreliable*! When you send data to a UDP socket, you cannot guarantee that it will arrive at the destination, nor the order that it will arrive. For example, if your client used two calls to `sendto()`, with first with message “Hello” and the second with message “World”, then it is possible that the server will receive “World” first, and then “Hello”. When creating an application using UDP, you (the programmer) must ensure that the messages are processed correctly.

1.1 Further Explanation

You should read the source code for the `server_udp.c`, and then the source code for `client_udp.c`. The comments contain further explanations of how the sockets communication is performed.

The example code for `client_udp.c` and `server_udp.c` came from:

http://www.cs.rpi.edu/courses/sysprog/sockets/sock.html

You may read through the details on this web page.

Most of the socket system calls are described in detail in their individual `man` pages. You should use the `man` pages for finding out further details of each function.

2 Tasks

In this lab you will work in pairs again (one client computer and one server computer). All tasks must be completed using Ubuntu Linux.

You can use the switched Ethernet for this lab. In other words, you do not have to configure the two computers in a peer-to-peer mode. Instead, make note of the IP address assigned to your computer and use that to communicate with your partner's client/server.

Using the same approach as the TCP lab, compile and run the `client_udp` and `server_udp` programs.

2.1 Exercise 1 – Capture the Exchange

Using Wireshark, capture the messages sent when running the server program and connecting and sending a "hello" message from the client.

Question: Compare the fields in a UDP header with those in a TCP header. Explain why they are different (in particular, why doesn't UDP contain all the fields that TCP does?).

Question: Draw a time sequence diagram showing the packet exchange. You do not have to show the times, sequence number or ACK numbers. Simply show the information (message types, protocol information) that is exchanged between sender and receiver.

3 Cleaning Up

At the end of the Lab class you must reset your computer back to the default settings, which include:

- Make sure you copy your source code to a USB drive – you will need it in future labs.
- Delete your source code from the PC.

```c
/* ****************************************************************
 * ITS 332 Information Technology II (Networking) Lab
 * Semester 2, 2006
 * SIIT
 *
 * Client/Server Programming Lab
 * File: client_idp.c
 * Date: 29 Jan 2007
 * Version: 1.0
 *
 * Description:
 * Client to demonstrate UDP sockets programming. You should read the
 * server_udp.c code as well.
 *
 * Usage:
 * client server_ip_address server_port_number
 *
 * Acknowledgement:
 * This code is based on the examples and descriptions from the
 * Computer Science Department, Rensselaer Polytechnic Institute at:
 * http://www.cs.rpi.edu/courses/sysprog/sockets/sock.html
 *
 * **************************************************************** */

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

/* ================================================================ */
/* error: display an error message and exit                        */
/* ================================================================ */
void error(char *msg)
{
    perror(msg);
    exit(0);
}

/* ================================================================ */
/* main: create socket and send message to server                  */
/* ================================================================ */
int main(int argc, char *argv[])
{
   int sock, n;
   struct sockaddr_in server, from;
   struct hostent *hp;
   char buffer[256];
   size_t length;

   /* User must input server and port number */
   if (argc != 3) { printf("Usage: server port\n");
                    exit(1);
   }
   /* Create  a Datagram (UDP) socket */
   sock= socket(AF_INET, SOCK_DGRAM, 0);
   if (sock < 0) error("socket");

   server.sin_family = AF_INET;

   /* Get the IP address for destination server */
   hp = gethostbyname(argv[1]);
   if (hp==0) error("Unknown host");

   /* Set the server address and port */
   bcopy((char *)hp->h_addr,
        (char *)&server.sin_addr,
         hp->h_length);
   server.sin_port = htons(atoi(argv[2]));

   length=sizeof(struct sockaddr_in);

   /* Prompt for message from user */
   printf("Please enter the message: ");
   bzero(buffer,256);
   fgets(buffer,255,stdin);

   /* Send message to socket (server) */
   n=sendto(sock,buffer,
            strlen(buffer),0,(struct sockaddr *) &server,length);
   if (n < 0) error("Sendto");

   /* Receive response from server */
   n = recvfrom(sock,buffer,256,0,(struct sockaddr *) &from, &length);
   if (n < 0) error("recvfrom");

   /* Display response to user */
   write(1,"Got an ack: ",12);
   write(1,buffer,n);
}
```

```
/* ******************************************************************
 * ITS 332 Information Technology II (Networking) Lab
 * Semester 2, 2006
 * SIIT
 *
 * Client/Server Programming Lab
 * File: server_udp.c
 * Date: 29 Jan 2007
 * Version: 1.0
 *
 * Description:
 * Server to demonstrate UDP sockets programming
 *
 * Usage:
 * server server_port_number
 *
 * Acknowledgement:
 * This code is based on the examples and descriptions from the
 * Computer Science Department, Rensselaer Polytechnic Institute at:
 * http://www.cs.rpi.edu/courses/sysprog/sockets/sock.html
 *
 * ****************************************************************** */

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

/* ================================================================ */
/* error: display an error message and exit                     */
/* ================================================================ */
void error(char *msg)
{
    perror(msg);
    exit(0);
}

/* ================================================================ */
/* main: create socket and receive/send to socket                */
/* ================================================================ */
int main(int argc, char *argv[])
{
   int sock, length, n;
   struct sockaddr_in server; /* server address structure */
   struct sockaddr_in from; /* source address structure */
   char buf[1024];
   size_t fromlen;

   /* Port number must be passed as parameter */
   if (argc < 2) {
      fprintf(stderr, "ERROR, no port provided\n");
      exit(0);
   }
```

```
    /* Create a Datagram (UDP) socket */
    sock=socket(AF_INET, SOCK_DGRAM, 0);
    if (sock < 0) error("Opening socket");

    length = sizeof(server);
    bzero(&server,length);

    /* Set the server address */
    server.sin_family=AF_INET;
    server.sin_addr.s_addr=INADDR_ANY;
    server.sin_port=htons(atoi(argv[1]));

    /* Bind the socket to the address */
    if (bind(sock,(struct sockaddr *)&server,length)<0)
        error("binding");

    fromlen = sizeof(struct sockaddr_in);
    /* Infinite loop, receiving data and sending response */
    while (1) {
        /* Receive data from socket. Parameters are:
     - server socket
     - buffer to read data into
     - maximum buffer size
     - flags to control the receive operation
     - structure to store source address
     - source address length
     */
        n = recvfrom(sock,buf,1024,0,(struct sockaddr *)&from,&fromlen);
        if (n < 0) error("recvfrom");
        write(1,"Received a datagram: ",21);
        write(1,buf,n);
        /* Write data to socket. Parameters are:
     - server socket
     - data to write
     - length of data
     - flags to control send operation
     - destination address
     - length of destination address
     */
        n = sendto(sock,"Got your message\n",17,
                   0,(struct sockaddr *)&from,fromlen);
        if (n  < 0) error("sendto");
    }
}
```