ECE 456 – Computer Networks

Programming Assignment # 2: Create a UDP Datagram Sender

1. Objective

This assignment is meant to refresh your programming skills prior to other assignments dealing with network programming. You will write a program to create a UDP datagram for sending a record.

2. Description

UDP is an unreliable, connectionless transport layer protocol. To send a message the UDP protocol entity on the sender side creates a UDP datagram and hands it down to the IP layer for delivery. UDP datagram is formed by encapsulating data with the UDP header, which includes a checksum. At the receiver end, the transport layer uses the checksum to detect errors in the datagram. If it is error-free, the header is removed and data is provided to the appropriate application based on the port number. Generally this phase is implemented in the Operating system as a part of the protocol stack, and the user is not required to code it.

As each node acts as a sender and a receiver, both these modules are required for implementation of UDP at a node. In assignment # 2 you will write the code for the sender. In Assignment # 3 you will write the code for receiver and detect whether a received datagram (using the sender code from Assignment # 2) is error free.

UDP datagram has a simple format where the header consists of only four fields: Source port number, Destination port number, Total length, and Checksum, as indicated in the shaded part of Figure 1. The number of bits in each field is identified within parenthesis.

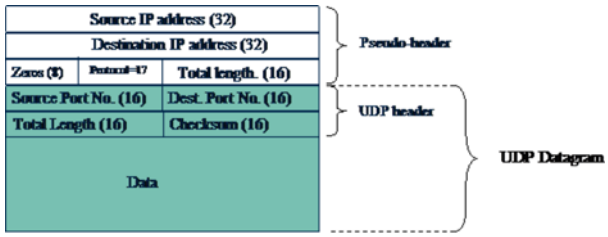


Figure 1. UDP datagram together with the pseudo header

Checksum

The *checksum* is used widely in networking protocols to provide protection against errors. **The UDP checksum is the 16-bit one's complement of the one's complement sum of the pseudo-header the UDP header, and the data. The pseudo header contains information such as source and destination address, protocol number (=17 for UDP), that is embedded in the IP header.**

Details on calculation of checksum can be found in (http://mathforum.org/library/drmath/view/54379.html):

1. Add the 16-bit values up. Each time a carry-out (17th bit) is produced, swing that bit around and add it back into the LSB (one's digit).

2. Once all the values are added in this manner, invert all the bits in the result. A binary value that has all the bits of another binary value inverted is called its "one's complement," or simply its "complement." Although the pseudo-header mimicking the IP header is used for the checksum calculation, the actual UDP datagram does not include the pseudo-header. Data is padded with zeros at the end (if necessary) to make the length of data a multiple of two octets. Figure 1 shows a UDP datagram together with the pseudo-header used for checksum calculation. The *source and destination IP addresses* are represented using 4 octets (an octet consists of 8 bits). The *Zeros* field indicates 8 bits of 0s. The value of *Protocol* field is 17 (21 octal) for the UDP Protocol. (For more detailed description of the fields, refer to http://www.faqs.org/rfcs/rfc768.html). The checksum is then computed over the combination of the pseudo-header and the real data, and the value is placed into the Checksum field. Refer Sections 8.4 and 3.9.3 of the text for details on the checksum calculation.

The source and destination IP addresses

This is an example IP address 192.168.0.10. Get the 32 bit value it. The Big-endian representation of this is 0x0A00A8C0.

192	168	0	1

1	0	168	192

Another example: 128.125.10.4. The Big-endian representation of this is 0x040A7D80.

This is the way one should store the IP addresses of source and destination in the header.

Total length

The *Total length* is the length in octets(bytes) of this datagram including the header and data before padding zeros.

Functions at Sender side:

Sender calculates the UDP checksum, and places it in the *Checksum* field. The UDP datagram sent by the sender does not include the pseudo-header used for the checksum calculation.

Functions at Receiver side:

When a UDP datagram arrives at the receiver side, receiver performs an error check. It forms the pseudo-header, prepends it to the actual UDP datagram, and then performs the checksum. A mismatch indicates an error has occurred and the segment is discarded.

Now it is time to start writing the code of the sender.

3. Procedure

Sender:

1. Accept 6 arguments from command line, which are "data file name", source-ip address, destination-ip address, source-port, destination-port and the "datagram file name"
   - The "data file name" is the input file containing data to be sent. This can be any format, txt, doc, jpeg etc.
   - The "datagram file name" is the output file that you need to generate at the source.
   - Source and Destination IP addresses are as the name suggests IP addresses of the source and destination respectively
   - Source and Destination ports are the port numbers for the source and destination respectively

   Example : .\sender image.jpeg 68.87.85.98 192.168.1.102 53 59887 output

2. Separate the octets from the IP addresses such that each octet can be represented using 8 bits (Use string operations for this) and the IP address as a whole can be represented using the 32 bits available. There are many ways of doing this. One suggestion is using "inet_addr" in C. Also go back to the note given for Big endian IP representation earlier.

3. Represent the port numbers using 16 bits. Use the command 'atoi' in C.

4. Read the data in binary format. For that use 'fread' in C.
5. Calculate the UDP length, i.e. length of the data without zero padding and the 8-byte header provided (refer the definition given earlier). This is represented using the 16 bit fields available in the UDP datagram structure (The data available may not be in multiples of 16 bit words, so you add sufficient padding of zeros to ensure that the UDP data field is complete)
6. Calculate the checksum using the data thus obtained (binary data).
7. Write the contents of the ***UDP datagram*** into the output file which is ***binary file***. *Please read the following instructions to write in to a binary file and read data as binary.*
8. Note that you are required to generate a **Binary** file containing your UDP Datagram dump. Binary files are not same as Text files, where you store 7-bit ASCII characters (and do not use the MSB).

In C you would use *fprintf* to write to a text file, and *fscanf* to read from a text file. But when you work with binary files you would use *fwrite*, and *fread* to write, and read from a file. Say, you wanted to write an unsigned short to a file and read it back. With text files you would do something similar to the following:

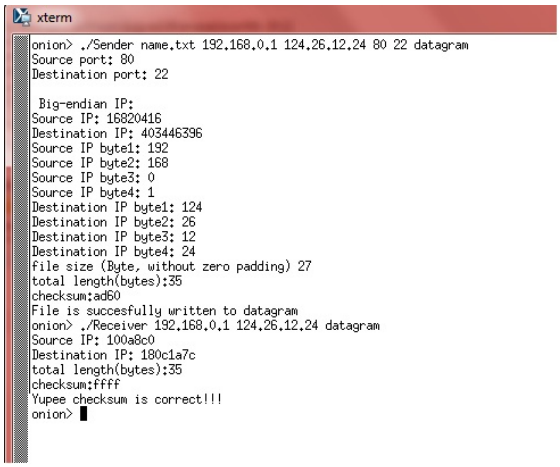
To write:
fprintf(fout,"%hx",val);
To read:
fscanf(fout,"%hx",&val);

But with binary files you would do something similar to the following:

To write:
fwrite(&val,sizeof(unsigned short),1,fout);
To read:
fread(&val,sizeof(unsigned short),1,fin);

Go through the *manual pages* of *fwrite* and *fread* for more details on the usage.

Executing and testing your 'sender' code with the receiver code provided. Example is given below.



NOTES

- Sample executables are available for you for Linux and Solaris platforms. They may be useful for verifying your program.

  Receiver(executable) can be downloaded from Canvas under Assignments--> Lab2.

, and to change mode use *chmod u+x file*. An example is given below.

The output dump obtained using send modules will have to be viewed in binary mode using one of the viewers available. On Unix, you can use "od" to do the same. "od" allows you to view the binary data in octal or hexadecimal mode.

- Write your code in C, C++, Java or another programming language, or any scripting language like Python, Perl, that you are familiar with.

  If you are not comfortable with any programming language, we would like to suggest "Python", a powerful scripting language extensively used in the field of Networking. Web-links to find python and related tutorials are listed below.

- Each individual is responsible for the entire assignment, and should be able to demonstrate and explain the different aspects of the program.

What To Submit (Submite all requirments as one zip file through Canvas)

- Source codes for sender.
- Executable and the test input file you used.
- A readme.txt file with instruction for compilation and execution.

- Include comments in your source code help in the understanding of the same; it is a good practice to include comments wherever possible and needed.
- Include error checks wherever possible to ensure robustness of your programs.
- You require to explain and demonstrate your code before demo due date.

Useful Links

UDP datagrams

http://www-net.cs.umass.edu/kurose/transport/UDP.html

http://penguin.dcs.bbk.ac.uk/academic/networks/transport-layer/udp/index.php

Get Python

http://www.python.org/download/

Python Tutorial

http://docs.python.org/tutorial/

String Operations

[C] http://www.cs.cf.ac.uk/Dave/C/node19.html

Arrays

[python] http://docs.python.org/library/array.html

File I/O

[C] http://www.elook.org/programming/c/stdio.html

[python] http://www.johnny-lin.com/cdat_tips/tips_fileio/bin_array.html

IP address conversion

[python] http://code.activestate.com/recipes/66517/