

ECE 456 – Computer Networks

Programming Assignment #1 : UDP Datagram

Aim

This assignment is meant to refresh your programming skills prior to other assignments dealing with network programming. You will write two programs, one creates a UDP datagram for sending a record, and the other to detect whether a received datagram is error free.

Description

UDP is an unreliable, connectionless transport layer protocol. To send a message, the UDP entity at the sender 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, that includes a checksum. At the receiver's 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. As each node acts as a sender and receiver, both these modules are required for implementation of UDP at a node.

Sender:

The checksum is used widely in networking protocols to provide protection against errors. UDP checksum is calculated over the UDP datagram and the pseudo header that mimics the IP header. *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 zero field indicates 8 bits of 0s. The protocol field is 17 (21 octal) for the Internet Protocol. (For more detailed description of the fields, refer to <http://www.faqs.org/rfcs/rfc768.html>). The UDP length is the length in octets of this datagram including the header and data.

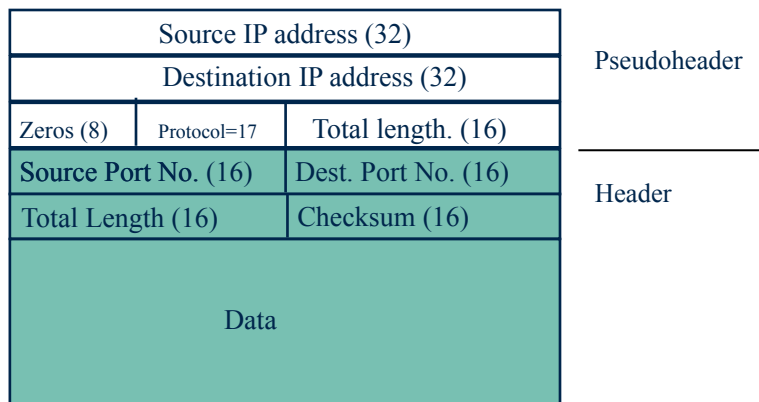


Figure 1

Checksum is the 16-bit one's complement of the one's complement sum of the pseudo header of information from the IP header, the UDP header, and the data. Data is padded with zeros at the end (if necessary) to make the length of data a multiple of two octets. Refer Sections 8.4 and 3.9.3 of the text for details on the checksum calculation. The total length of the "pseudo header" is 11 bytes. It is padded to 12 bytes with a byte of zeros and then prepended to the UDP datagram. 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. The pseudo header is used only for this calculation and is then discarded; it is not actually transmitted. The UDP entity at the destination device creates the same pseudo header when calculating its checksum to compare to the one transmitted in the UDP header.

Receiver:

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

Procedure

There are two parts in this assignment. You need to submit 2 programs, one for the sender and one for the receiver.

Sender module:

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”
2. The “data file name” is the input file (http://www.engr.colostate.edu/ECE456/ECE456_Sp08/pa1Input.txt)
3. The “datagram file name” is the output file that you need to generate at the source.
4. Source and Destination IP addresses are as the name suggests IP addresses of the source and destination respectively
5. Source and Destination ports are the port numbers for the source and destination respectively
6. 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.
7. Represent the port numbers using 16 bits
8. UDP length refers to the length of the data and the 8-byte header provided. 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)
9. Calculate the checksum using the data thus obtained (binary data).
10. Dump the contents of the **UDP datagram** into the output file in **binary**
11. (Sample ~~binary read write~~ implementation available at <http://www.metalshell.com/view/source/27/>)

Receiver Module:

1. Accept 5 arguments from command line, which are source-ip address, destination-ip address, source-port, destination-port and the “datagram file name”
2. The “datagram file name” is the input file that you obtained using the source.
3. Source and Destination IP addresses are as the name suggests IP addresses of the source and destination respectively
4. Source and Destination ports are the port numbers for the source and destination respectively
5. Read the contents of the **UDP datagram** from the datagram file
6. Calculate the checksum.
7. If the checksum is not correct, display “Checksum Error”
8. If checksum is correct, display “Datagram from source-address xxx source-port xxx to to dest-address xxx dest-port xxx; Length xxx bytes. Here xxx corresponds to the value in appropriate format.

A sample run of your programs could look something like this

```
send_udp "data file name" source-ip destination-ip source-port dest-port "datagram file name"  
recv_udp "datagram file name" source-ip destination-ip source-port destination-port
```

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>

http://www.liacs.nl/~herbertb/courses/networks/handouts/udp_pseudo.html

String Operations

<http://www.scit.wlv.ac.uk/cgi-bin/mansec?3C+strncpy>

File I/O

<http://www.elook.org/programming/c/stdio.html>

NOTES

- Comments in your source code help in the understanding of the same, try to include comments wherever possible
- Try to include error checks wherever possible to ensure robustness of your programs
- You should work individually. Each individual however is responsible for the entire assignment, and should be able to demonstrate and explain the different aspects of the program.
- Write your code in C, C++, Java or another programming language that you are familiar with.
- The TA will answer questions related only to C/C++. Use of other platforms and languages is OK, but you are on your own. TA will answer your questions, but will not debug your program.
- Submit your code and the makefile (if any) by email to the TA: nandan@simla.colostate.edu.
- Include readme.txt file with instructions for compilation. Do NOT submit executables.
- You should be able to explain and demonstrate your code.