3/134/

Assignment 4

Problem Statement: Write a program using UDP Sockets to enable file transfer between two machines. Demonstrate the packets captured traces using Wireshark Packet Analyzer Tool for peer to peer mode.

Objective:

1. To create a UDP socket
2. Send file name from client and receive file contents from server

Learning Outcome: Students will be able to

- Demonstrate working of UDP sockets
- Perform file handling using these sockets.

Requirements:

- Open source linux based OS
- Eclipse IDE or Python interpreter

Theory

A transport layer protocol usually has several responsibilities. One is to create a process-to process communication; UDP uses port numbers to accomplish this. Another responsibility is to provide control mechanisms at the transport level. UDP does this task at a very minimal level. There is no flow control mechanism and there is no acknowledgement for received packets. UDP, however, does provide error control to some extent. If UDP detects an error in the received packet, it silently drops it. UDP is a connection-less, unreliable transport protocol. It does not add anything to the services of IP except for providing process-to-process communication instead of host-to-host communication. UDP packets, called user datagrams, have a fixed-size header of 8 bytes.

- Source port number. This is the port number used by the process running on the source host. It is 16 bits long, which means that the port number can range from 0 to 65,535. If

the source host is the client (a client sending a request), the port number, in most cases, is an ephemeral port number requested by the process and chosen by the UDP software running on the source host. If the source host is the server (a server sending a response), the port number, in most cases, is a well-known port number.

• Destination port number. This is the port number used by the process running on the destination host. It is also 16 bits long. If the destination host is the server (a client sending a request), the port number, in most cases, is a well-known port number. If the destination host is the client (a server sending a response), the port number, in most cases, is an ephemeral port number. In this case, the server copies the ephemeral port number it has received in the request packet.

• Length. This is a 16-bit field that defines the total length of the user datagram, header plus data. The 16 bits can define a total length of 0 to 65,535 bytes. How- ever, the total length needs to be much less because a UDP user datagram is stored in an IP datagram with the total length of 65,535 bytes. The length field in a UDP user datagram is actually not necessary. A user datagram is encapsulated in an IP datagram. There is a field in the IP datagram that defines the total length. There is another field in the IP datagram that defines the length of the header. So if we subtract the value of the second field from the first, we can deduce the length of the UDP datagram that is encapsulated in an IP datagram.

UDP length = IP length – IP header's length

However, the designers of the UDP protocol felt that it was more efficient for the destination UDP to calculate the length of the data from the information provided in the UDP user datagram rather than ask the IP software to supply this information. We should remember that when the IP software delivers the UDP user datagram to the UDP layer, it has already dropped the IP header.

• Checksum. This field is used to detect errors over the entire user datagram (header plus data).

3/134/

UDP provides a connection-less service. This means that each user datagram sent by UDP is an independent datagram. There is no relationship between the different user datagrams even if they are coming from the same source process and going to the same destination program. The user datagrams are not numbered. Also, there is no connection establishment and no connection termination as is the case for TCP. This means that each user datagram can travel on a different path.

Flow Control: UDP is a very simple protocol. There is no flow control, and hence no window mechanism. The receiver may overflow with incoming messages. The lack of flow control means that the process using UDP should provide for this service, if needed.

Error Control: There is no error control mechanism in UDP except for the checksum. This means that the sender does not know if a message has been lost or duplicated. When the receiver detects an error through the checksum, the user datagram is silently dis- carded. The lack of error control means that the process using UDP should provide for this service if needed.

Checksum: UDP checksum calculation is different from the one for IP. Here the checksum includes three sections: a pseudo header, the UDP header, and the data coming from the application layer. The pseudo header is the part of the header of the IP packet in which the user datagram is to be encapsulated with some fields filled with 0s (see Figure 14.3) If the checksum does not include the pseudo header, a user datagram may arrive safe and sound. However, if the IP header is corrupted, it may be delivered to the wrong host. The protocol field is added to ensure that the packet belongs to UDP, and not to TCP. We will see later that if a process can use either UDP or TCP, the destination port number can be the same. The value of the protocol field for UDP is 17. If this value is changed during transmission, the checksum calculation at the receiver will detect it and UDP drops the packet. It is not delivered to the wrong protocol. Note the similarities between the pseudo header fields and the last 12 bytes of the IP header.

3/34/

Encapsulation: When a process has a message to send through UDP, it passes the message to UDP along with a pair of socket addresses and the length of data. UDP receives the data and adds the UDP header. UDP then passes the user datagram to IP with the socket addresses. IP adds its own header, using the value 17 in the protocol field, indicating that the data has come from the UDP protocol. The IP datagram is then passed to the data link layer. The data link layer receives the IP datagram, adds its own header (and possibly a trailer), and passes it to the physical layer. The physical layer encodes the bits into electrical or optical signals and sends it to the remote machine.

De-encapsulation: When the message arrives at the destination host, the physical layer decodes the signals into bits and passes it to the data link layer. The data link layer uses the header (and the trailer) to check the data. If there is no error, the header and trailer are dropped and the datagram is passed to IP. The IP software does its own checking. If there is no error, the header is dropped and the user datagram is passed to UDP with the sender and receiver IP addresses. UDP uses the checksum to check the entire user datagram. If there is no error, the header is dropped and the application data along with the sender socket address is passed to the process. The sender socket address is passed to the process in case it needs to respond to the message received. When a process has a message to send through UDP, it passes the message to UDP along with a pair of socket addresses and the length of data. UDP receives the data and adds the UDP header. UDP then passes the user datagram to IP with the socket addresses. IP adds its own header, using the value 17 in the protocol field, indicating that the data has come from the UDP protocol. The IP datagram is then passed to the data link layer. The data link layer receives the IP datagram, adds its own header (and possibly a trailer), and passes it to the physical layer. The physical layer encodes the bits into electrical or optical signals and sends it to the remote machine.

Multiplexing: At the sender site, there may be several processes that need to send user datagrams However, there is only one UDP. This is a many-to-one relationship and requires multiplexing. UDP accepts messages from different processes, differentiated by their assigned port numbers. After adding the header, UDP passes the user datagram to IP.

De-multiplexing: At the receiver site, there is only one UDP. However, we may have many processes that can receive user datagrams. This is a one-to-many relationship and requires demultiplexing. UDP receives user datagrams from IP. After error checking and dropping of the header, UDP delivers each message to the appropriate process based on the port numbers.

Conclusion: UDP (User Datagram Protocol) is an alternative communications protocol to Transmission Control Protocol (TCP) used primarily for establishing low-latency and loss tolerating connections between applications on the Internet. Both UDP and TCP run on top of the Internet Protocol (IP). Thus we have successfully implemented the socket programming for UDP using C.