Lab IV: UDP/TCP

Objectives

- To understand the differences between UDP and TCP
- To understand the connection establishment process of TCP
- To understand how TCP deals with connection failures

Reading List

- **nuttcp:** Read the manual page of nuttcp at
  `http://manpages.ubuntu.com/manpages/trusty/man8/nuttcp.8.html`
- Read the man page of `telnet`, `ftp` and `tftp` commands.

Equipment and Package List

	Quantity
Linux PC with two Ethernet interfaces	3
Unmanaged switch	1
Ubuntu packages: nttcp, vsftpd, openbsd-inetd	

Background

This lab explores the operation of the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP), the two transport protocols of the Internet protocol architecture. UDP is a simple protocol for exchanging messages from a sending application to a receiving application. UDP adds a small header to the message. The resulting encapsulated data unit is called a *UDP segment,* which is then encapsulated in an IP header and delivered to the destination.

The operation of TCP is more complex. TCP is a connection-oriented protocol, in which a TCP client establishes a logical connection to a TCP server before data transmission can take place. TCP requires three segments to open a connection. This procedure is called a three-way handshake. During the handshake the TCP client and TCP server negotiate essential parameters of the TCP connection, including the initial sequence numbers, the maximum segment size, and the size of the windows for the sliding window flow control. TCP requires three or four packets to close a connection. Each end of the connection is closed separately, and each part of the closing is called a

half-close. TCP uses bit flags in the TCP header to indicate that a TCP header carries control information. The flags involved in the opening and the closing of a connection are SYN, ACK, and FIN.

Once a connection is established, data transfer can proceed in both directions. The data unit of TCP, called a *TCP segment,* consists of a TCP header and payload that contains application data. A sending application submits data to TCP as a single stream of bytes without indicating message boundaries in the byte stream. The TCP sender decides how many bytes are put into a segment.

TCP ensures reliable delivery of data, and uses checksums, sequence numbers, acknowledgments, and timers to detect damaged or lost segments. The TCP receiver acknowledges the receipt of data by sending an acknowledgment segment (ACK). Multiple TCP segments can be acknowledged in a single ACK, When a TCP sender does not receive an ACK within a timeout period, the segment is assumed lost and is retransmitted.

TCP has two mechanisms that control the amount of data that a TCP sender can transmit. First, the TCP receiver informs the TCP sender how much data the TCP sender can transmit. This is called *flow control.* Second, when the network is overloaded and TCP segments are lost, the TCP sender reduces the rate at which it transmits traffic. This is called *congestion control.*

In Linux Raspberry Pi, the `nuttcp` command is a tool used to generate synthetic UDP and TCP traffic loads. Together with `ping` and `traceroute`, `nuttcp` is an essential utility program for debugging problems in IP networks. Running the *nuttcp* tool consists of setting up a *nuttcp* receiver (server) on one host and then a *nuttcp* sender on another host. Once the *nuttcp* sender is started, it sends the specified amount of data as fast as possible to the *nuttcp* receiver.

An *nuttcp* receiver process is started with the command

```
nuttcp –S --nofork [-lbuflen] [-pport] [-u]
```

An *nuttcp* sender process is started with the command

```
nuttcp -t [-lblocksize] [-nnumblock] [-pport] [-u] [-D] IPaddress
```

The options of the command are:

`-S` Make nuttcp wait for connections (server/receiver)

`-t` Specifies the transmit mode (client/transmitter)

`-u` Specifies to use UDP instead of TCP. By default, *nuttcp* uses TCP to send data.

`-n`*`numblock`* Number of data blocks to be transmitted (default value is 2048 blocks).

`-l`*`blocksize`* Size of the data blocks that are passed to UDP or TCP in bytes (default is 4096 bytes). When UDP is used, this value is the number of data bytes in UDP datagram.

`-D` Disables buffering of data in TCP and forces immediate transmission of the data at the *nttcp* sender. Used only in the context of TCP.

`-pport` Port number to send to or listen on. The port number must be identical at the sender and at the receiver. The default value is 5000.

`IPaddress` IP address of the *nttcp* receiver to send data to.

Use `nuttcp -h` or to verify the command usage option before using it to send/receive files.

Lab Procedures

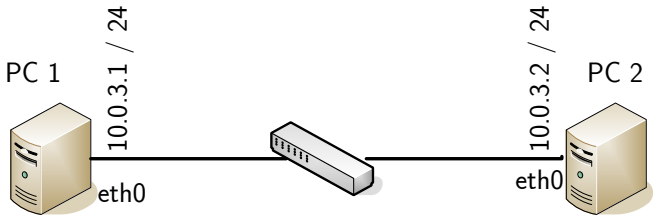

Figure 4.1 Network configuration

Part I: Transmitting Data with UDP

1. Connect two PCs with a switch as shown in Figure 4.1. Appropriately configure the IP addresses of the PCs and verify their connectivity.
2. At PC2, start a *nuttcp* receiver to receive UDP traffic with 1024-byte block size at the default server port.

   ```
   PC2% nuttcp -S --nofork -l1024 -u
   ```

   Run the command only once as it is executed as a daemon process.
3. At PC 1,
   a. Start *wireshark* to capture packets on interface *eth0* with display filter *'udp'*.
   b. Start a *nuttcp* sender (or nuttcp if the receiver is *nuttcp*) that transmits 10 blocks of UDP traffic with 1024-byte block size to PC2.

      ```
      PC1% nuttcp -t -l1024 -n10 -u 10.0.3.2
      ```
4. Stop *wireshark* capture on PC1.

Lab Questions:

Observe the captured traffic captured by *Wireshark* and answer the following questions.

1.1 How many packets are exchanged in the data transfer? How many UDP datagrams are there? What is the size of the UDP payload of these packets?

1.2 Compare the total number of bytes transmitted, in both directions, including Ethernet, IP, and UDP headers, to the amount of application data transmitted.

1.3 Inspect the fields in the UDP headers. Which fields in the headers do not change in different packets?

1.4 Observe the port numbers in the UDP header. How did the *nuttcp* sender select the source port number?

Part II: Transmitting Data with TCP

1. At PC1, start *wireshark* and capture packets on interface *eth0* with display filter 'tcp'
2. Start an *nuttcp* receiver on PC2 and sender on PC1 to send 10 blocks of data over TCP using the following commands:

   ```
   PC2% nuttcp -S --nofork -l1024
   PC1% nuttcp -t -l1024 -n10 10.0.3.2
   ```

3. Stop *wireshark* capture on PC1, and save the captured traffic to files.

Report Questions:

Include relevant parts of the Wireshark data to support each of your answers.

2.1 How many TCP segments (both control and data) are exchanged between PC1 and PC2 ?

2.2 What are the sizes of the data segments in Question 2.1? Are they all the same ? If not, explain why they have different sizes.

2.3 How many segments are TCP control segments? What are they (look at the Flag field) ?

2.4 Compare the total number of bytes transmitted, in both directions, including Ethernet, IP, and TCP headers, to the amount of application data transmitted.

Part III: File Transfers Using TCP and UDP

1. At PC1,
   a. Go to `/etc/inetd.conf` and set the default directory that TFTP accesses file to `/tftpboot` with the following line:

      ```
      tftp dgram udp wait root /usr/sbin/tcpd /usr/sbin/in.tftpd /tftpboot
      ```

b. Delete all files in */tftpboot* and create a large 20MB file with random content in the directory with the command

```
dd if=/dev/urandom of=largefile.txt bs=2048 count=10000
```

Then, copy the file to directories /root and /tftpboot. Make sure that the directory `/tftpboot` and everything inside are readable by all user groups by using the command

```
chmod –R 777 /tftpboot
```

2. **FTP server setup at PC1:** Make sure that the following lines in `/etc/vsftpd.conf` are uncommented (no # at the beginning of the line)

```
local_enable=YES
write_enable=YES
```

Then, restart `vsftpd` (FTP server daemon) by typing the command[1]:

```
/etc/init.d/vsftpd restart
```

3. Start *wireshark* at PC1.
4. **Perform *FTP* file transfer at PC2**:
   a. Change the current directory to */labdata* and delete every files in there. If the directory does not exist, create it.
   b. Invoke an *FTP* session to PC1 and log in as the root user[2].

```
PC2% cd /labdata
PC2% ftp 10.0.3.1
```

   c. Download the large file you have created in Step 1 from PC1 to directory */labdata* on PC2 with the following commands at the FTP prompt:

```
ftp> cd /tftpboot
ftp> get largefile.txt
ftp> quit
```

   d. Observe the output of the FTP session and save the Wireshark output to a file.

5. **TFTP server setup at PC1**: Make sure the following line appear in `/etc/inetd.conf`

[1] New version of Linux uses '`service vsftpd restart`'

[2] To allow root login, comment out root in `/etc/ftpusers` at PC1 and restart. If you use another account to log in, create `/labdata` directory in the home directory.

```
tftp dgram udp wait nobody /usr/sbin/tcpd in.tftpd /tftpboot
```

The server won't run if `in.tftpd` does not exist. Then, run

```
service openbsd-inetd restart
```

6. **Perform *TFTP* file transfer at PC2**:
   a. Invoke an *TFTP* session to PC1 and download the large file you have created earlier with the following commands:

      ```
      PC2% cd /labdata
      PC2% tftp 10.0.3.1
      ```

      By default, *TFTP* copies data from the remote directory */tftpboot.* Therefore, we can transfer the file by using command immediately with.

      ```
      tftp> get largefile.txt
      tftp> quit
      ```

   b. Observe the output of the *TFTP* session and save the Wireshark output to a file.
   c. Delete the directory */labdata*
7. Stop *wireshark* on PC1.

Report Questions:

3.1 From the timestamps recorded by *Wireshark,* obtain the times it took to transfer the large file with *FTP* and with *TFTP.* Which one, *FTP* or *TFTP*, transfers the file faster? Look at the Wireshark data and use your knowledge of *FTP, TFTP,* TCP, and UDP to explain the outcome.

3.2 How many parallel connections were created in the FTP session? What are those connections for? What are the source/destination port numbers of those connections?

Part IV: TCP Connection Management

Opening and closing a TCP connection

1. **Telnet server setup**: At PC2, make sure that the following lines exist in `/etc/securetty`

```
pts/0
pts/1
pts/2
pts/3
pts/4
```

```
pts/5
```

Then, restart the telnet server service by typing the command

```
service openbsd-inetd restart
```

2. Start *wireshark* on PC1 to capture packets on eth0.
3. Establish a *Telnet* session from PC1 to PC2 with the command

```
PC1% telnet 10.0.3.2
```

Terminate the connection after the connection has been successfully established by suspending the telnet session with 'Ctrl-]' key sequence (Control key with Right Square Bracket character) and type 'quit' at the *Telnet* prompt.

4. The closing of a connection can also be initiated by the server. Reestablish a *Telnet* session on PC1 to PC2 as in Step 3 but do not type anything when the username prompt shows up. After a few minutes, the connection will be closed by the TCP server and a message displayed at the *Telnet* client application.
5. Save the *wireshark* output to a file.

Report Questions:

Use the saved Wireshark output to answer the following questions. Include relevant parts of the Wireshark data to support each of your answers.

4.1 Identify the segments of the three-way handshake. Which flags are set in the TCP headers? Explain how these flags are interpreted by the receiving TCP server or TCP client.

4.2 During the connection setup, the TCP client and TCP server tell each other the first sequence number they will use for data transmission. What are the initial sequence numbers of the TCP client and the TCP server? In Wireshark, you need to go to Preferences → Protocols → TCP and uncheck the box that says "Relative Sequence Number" before answering this question.

4.3 Identify the first segment that contains application data. What is the sequence number used in the first byte of application data sent from the TCP client to the TCP server?

4.4 The TCP client and TCP server exchange the initial window sizes to get the maximum amount of data that the other side can send at any time. Determine the values of the initial window sizes for the TCP client and the TCP server.

4.5 What is the MSS value that is negotiated between the TCP client and the TCP server?
4.6 How long does it take to open the TCP connection?
4.7 In Step 3, identify the packets that are involved in closing the TCP connection. Which flags are set in these packets? Explain how these flags are interpreted by the receiving TCP server or TCP client.
4.8 Describe how the closing of the connection in Step 4 is different from Step 3. How long does the *Telnet* server wait until it closes the TCP connection?

Requesting a connection to a nonexisting host.

1. Start a new traffic capture with *wireshark* on PC1.
2. Create a static entry in the ARP table (Mapping of hardware address and IP address) for a non-existing IP address 10.0.3.100 in your network with the command:

   ```
   PC1% arp -s 10.0.3.100 00:01:02:03:04:05
   ```

   This fake ARP entry prevents ARP resolution timeout to take effect.
3. At PC1, establish a *Telnet* session to the non-existing host in Step 2 with

   ```
   PC1% telnet 10.0.3.100
   ```

4. Wait until the TCP client gives up trying to establish a connection. Save the *wireshark* output.

Report Questions:

Include relevant parts of the Wireshark output to support each of your answers.
4.9 How often does the TCP client try to establish a connection? How much time elapses between the repeated attempts to open a connection?
4.10 Does the TCP client send out any control segments when it gives up on establishing a connection? Why or Why not?

Requesting a connection to a nonexisting port.

1. Start a new traffic capture with *wireshark* on PC1.
2. At PC1, telnet to port 80 of PC2 with the command

   ```
   PC1% telnet 10.0.3.2 80
   ```

There should be no TCP server running at this port number on PC2. Observe the TCP segments of the packets that are transmitted.

3. Save the *wireshark* output.

Report Question:

Include relevant parts of the Wireshark data to support your answers.

4.12 What kind of segment is returned by TCP at PC2 to close this connection? How long does the process of ending the connection take?

4.13 Suppose you want to know if a certain network process is running at a given host, how would you exploit the knowledge of TCP connection establish to do it?

Part V: TCP Bulk Data Transfer

1. At PC1, start *wireshark* to capture packets on eth0.
2. Start an *nuttcp* receiver on PC2 and sender on PC1 to send 50 blocks of data over TCP with the following commands:

```
PC2% nuttcp -S --nofork -l1024
PC1% nuttcp -t -l1024 -n50 10.0.3.2
```

3. Stop *wireshark* capture on PC1 and save the captured traffic to a file.

Report Questions:

Include relevant parts of the Wireshark data to support each of your answers.

5.1 How frequent does the receiver send ACKs? Determine and explain the rule used by TCP to send ACKs.

5.2 How many bytes of data does the receiver acknowledge in a typical ACK? What is the largest amount of data acknowledged in a single ACK?

5.3 What is the maximum and minimum window size advertised by the receiver? How does the window size vary during the lifetime of the TCP connection?

5.4 Select an arbitrary ACK segment sent by PC2 to PC1 and relate it to a segment sent by PC1. How long did it take from the transmission of the segment until the ACK arrives at PC1?

5.5 Does the TCP sender generally transmit the maximum amount of data allowed by the advertised window size? Explain.

5.6 After the TCP sender has sent all its data, what segment does it send?

Review Questions

1. Explain the role of port numbers in TCP and UDP.
2. What are the commands “ftp” and “tftp” for ? Explain the differences between them.
3. What does a telnet command do? What is the port number used by Telnet server?
4. Explain the roles of the SYN, FIN, ACK, and RST flags in the TCP header.