LAB FIVE - Transport Layer Protocols: UDP & TCP

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, and the resulting data unit is called a *UDP segment*. When a UDP segment is transmitted, the datagram is encapsulated in an IP header and delivered to its destination. There is one UDP segment for each application message.

The operation of TCP is more complex. First, 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. 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 acknowledgement segment (ACK). Multiple TCP segments can be acknowledged in a single ACK (cumulative ACK). When a TCP sender does not receive an ACK, the data 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*.

The lab covers the main features of UDP and TCP. Part 1 compares the performance of data transmissions in TCP and UDP. Part 2 explores how TCP and UDP deal with IP fragmentation. The remaining parts address important components of TCP. Part 3 explores connection management, Parts 4 and 5 look at flow control and acknowledgements, and Part 6 explores retransmissions.

This lab uses the topology as shown in Figure 5.1. The IP addresses are given in Table 5.1.

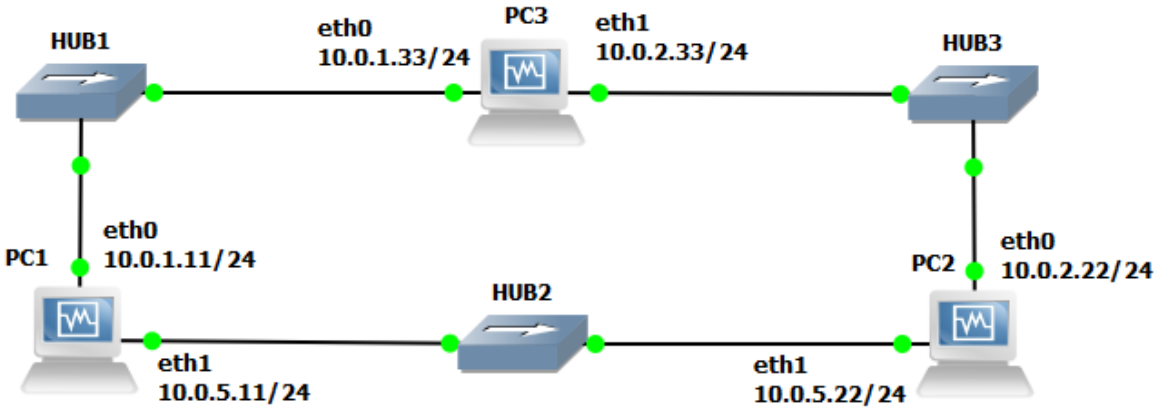

Figure 5.1 Network topology for Lab 5

VMS	eth0	eth1	Default Gateway
PC1	10.0.1.11 / 24	10.0.5.11 / 24	10.0.1.33
PC2	10.0.2.22 / 24	10.0.5.22 / 24	10.0.2.33
PC3	10.0.1.33 / 24	10.0.2.33 / 24	

Table 5.1 IP addresses of the PCs and Routers

PART 1. Learning how to Ping using Echoping on PCs

The **ping** command is a network administration tool to test network reachability of a host on an Internet Protocol (IP) network. The command **echoping** *extends* the capability of the `ping` command, so PCs can issue `ping` commands over either a TCP or a UDP connection. The following table summarizes the main uses of `echoping`. More info can be found on the `echoping` man page at http://manpages.ubuntu.com/manpages/raring/man1/echoping.1.html

```
NAME:
echoping - tests a remote host with TCP or UDP

SYNOPSIS:
      echoping[Options] [hostname] [:port]

OPTIONS:
   -u                : Use UDP instead of TCP
   -v                : Verbose mode
   -s number         : Size of the data to send
   -n number         : # of repeated tests
   -t number         : # of seconds to wait for a reply before giving up
```

The command **ncat** is a feature-packed networking utility which reads and writes data across networks from the command line. The table below contains some usage scenarios for `ncat`, More info can be found on the `ncat` man page at http://man7.org/linux/man-pages/man1/ncat.1.html

We are going to use this tool to force a PC to listen to a specific port number so that a receiving PC can send responses back to the sender using a particular transport protocol configured in the terminal (console) window at both ends.

```
NAME:
      ncat - Concatenate and redirect sockets

SYNOPSIS:
      ncat [Options] [hostname] [port]

OPTIONS:
   -e, --exec <command>   : Executes the given command
   -l, --listen           : Bind and listen for incoming connections
   -k, --keep-open        : Accept multiple connections in listen mode
   -u, --udp              : Use UDP instead of the default TCP
```

Exercise 1(A). Network setup

1. Connect the Ethernet interfaces of the VMS as shown in Figure 5.1. Configure the IP addresses of the interfaces and default gateways as given in Table 5.1.

2. PC1 and PC2 are set up as hosts, and IP forwarding should be disabled. Actually, it is disabled by default. To ensure that it is not functioning as a router, on PC1 for example, this is done with the command:

```
PC1 % echo "0" > /proc/sys/net/ipv4/ip_forward
```

3. PC3 is set up as an IP router. Enable IP forwarding on PC3 with the command:

```
PC3% echo "1" > /proc/sys/net/ipv4/ip_forward
```

4. Verify that the setup is correct by issuing a ping command from PC1 to PC2.

```
PC1# ping 10.0.2.22 -c 5
PC1# ping 10.0.5.22 -c 5
```

Exercise 1(B). Transmitting data with UDP

1. On PC1, start Wireshark on eth0 to capture the traffic exchanged.

2. Set up PC2 to listen to the `echoping` protocol on port "7" and transmit using UDP, with the following command:

```
PC2% ncat -l 7 -k -u -e "/bin/cat"
```

3. On PC1, issue an `echoping` using UDP with the following command:

```
PC1% echoping -v -u 10.0.5.22
```

4. Stop Wireshark and save the captured traffic.

5. On PC2 type ^C (control c) to kill the `ncat` process.

```
PC2% ^C
```

Lab Questions:

Use the data captured with Wireshark to answer the following questions.

- How many packets are exchanged in the data transfer? How many packets are transmitted for each UDP datagram? What is the size of the UDP payload of these packets?

- Compare the total number of bytes transmitted, in both directions, including Ethernet, IP, and UDP headers, to the amount of application data transmitted.
- Inspect the fields in the UDP headers. Which fields in the headers do not change in different packets?
- Observe the port number in the UDP header. Why was port number 7 chosen to issue the ping?

Exercise 1(C). Transmitting data with TCP

Here, you repeat the previous exercise, but use TCP for data transfer.

1. On PC1 start Wireshark on eth0 to capture packets.

2. On PC2, set up the system to listen to the echoping on port number "7". Note that the default for `ncat` is to use TCP.

```
PC2% ncat -l 7 -k -e "/bin/cat"
```

3. On PC1 issue an `echoping` over TCP with the following command

```
PC1% echoping -v 10.0.5.22
```

4. Stop Wireshark and save the captured traffic.

5. On PC2 type the following to kill the `ncat` TCP process.

```
PC2% fuser -k 7/tcp
```

Lab Questions:

Use the data captured with Wireshark to answer the following questions.

- How many packets are exchanged in the data transfer? What are the sizes of the TCP segment?
- What is the range of the sequence numbers?
- How many packets are transmitted by PC1, and how many packets are transmitted by PC2?
- How many packets do not carry a payload, that is, how many packets are control packets?
- Compare the total number of bytes transmitted, in both directions, including Ethernet, IP, and TCP headers, to the amount of application data transmitted.
- Inspect the fields in the TCP headers. Which packets contain flags in the TCP header? Which types of flags do you observe?
- Compare the amount of data transmitted in the TCP and the UDP data transfers.
- Take the largest UDP segment and the largest TCP segment that you observed, and compare the amount of application data that is transmitted in the UDP segment and the TCP segment.

PART 2. IP Fragmentation of UDP and TCP Traffic

In this part of the lab, you observe the effect of IP fragmentation on UDP and TCP traffic. Fragmentation occurs when the transport layer sends a packet of data to the IP layer that exceeds the Maximum Transmission Unit (MTU) of the underlying data link network. For example, in Ethernet networks, the MTU is 1500 bytes. If an IP datagram exceeds the MTU size, the IP datagram is fragmented into multiple IP datagrams, or, if the Don't Fragment (DF) flag is set in the IP header, the IP datagram is discarded and an ICMP message is sent back to the sender indicating the problem.

When an IP datagram is fragmented, its payload is split across multiple IP datagrams, each satisfying the limit imposed by the MTU. Each fragment is an independent IP datagram and is routed in the network independently from the other fragments. Fragmentation can occur at the sending host or at intermediate IP routers. Fragments are reassembled only at the destination host.

Even though IP fragmentation provides flexibility that can hide differences of data link technologies to the higher layers, it incurs considerable overhead and, therefore, should be avoided. TCP tries to avoid fragmentation with a Path MTU Discovery scheme that determines a maximum segment size (MSS), which does not result in fragmentation.

In this part, you explore the issues with IP fragmentation of TCP and UDP transmissions in the network configuration shown in Figure 5.1, with PC1 as sending host, PC2 as receiving host, and PC3 as intermediate IP router.

Exercise 2(A). UDP and Fragmentation

In this exercise you will observe IP fragmentation of UDP traffic. You will use ping to generate UDP traffic between PC1 and PC2, across PC3, and gradually increase the size of UDP segment until fragmentation occurs. You will observe that IP headers do not set the DF bit for UDP payloads.

1. Verify that the network is configured as shown in Figure 5.1 and Table 5.1. The PCs should be configured as described in Exercise 1(A).

2. Start Wireshark on the eth0 interface of both PC1 and PC2, and start to capture traffic. Do no set any filters.

6. Set up PC2 to listen for the `echoping` command on port "7" and to transmit using UDP:

```
PC2% ncat –l 7 -k -u -e "/bin/cat"
```

3. Use `echoping` to generate UDP traffic between PC1 and PC2. The connection parameters are selected so that IP fragmentation does not occur yet. On PC1, execute the command:

```
PC1% echoping –v -u –s 512 10.0.2.22
```

4. To observe the UDP segment size at which fragmentation occurs and the maximum size of the UDP segment that the system can send, we gradually increment the size of the UDP segment by increasing the argument given with the "-s" option. Repeat step 3 with increasing segment sizes until you have observed both these events.

5. Stop the traffic capture and save the Wireshark output. Kill the `ncat` udp process with ^C.

Lab Questions:

- Determine the exact UDP segment size at which fragmentation occurs.
- Determine the UDP header size.
- Determine the maximum size of the UDP segment that the system can send and receive, regardless of fragmentation (i.e. fragmentation of data segments occurs until a point beyond which the segment size is too large to be handled by UDP/IP).
- From the saved Wireshark data, select one IP datagram that is fragmented. Look at the complete datagram before fragmentation and include all fragments after fragmentation. For each fragment of this datagram, determine the values of the fields in the IP header that are used for fragmentation (Identification, Fragment Offset, Don't Fragment Bit, More Fragments Bit).

Exercise 2(B). TCP and Fragmentation

TCP tries to completely avoid fragmentation with the following two mechanisms:

- When a TCP connection is established, it negotiates the maximum segment size (MSS) to be used. Both the TCP client and the TCP server send the MSS as an option in the TCP header of the first transmitted TCP segment. Each side sets the MSS so that no fragmentation occurs at the outgoing network interface, when it transmits segments. The smaller value is adopted as the MSS value for the connection.

- The exchange of the MSS addresses MTU constraints only at the hosts, not at the intermediate routers. To determine the smallest MTU on the path from the sender to the receiver, TCP employs a method known as Path MTU Discovery, which works as follows. The sender always sets the DF bit in all IP datagrams. When a router needs to fragment an IP packet with the DF bit set, it discards the packet and generates an ICMP error message of type "destination unreachable; fragmentation needed". Upon receiving such an ICMP error message, the TCP sender reduces the segment size. This continues until a segment size that does not trigger an ICMP error message is determined.

1. Modify the MTU of the interfaces with the values shown in Table 5.2.

PCs	**MTU size on eth0**	**MTU size on eth1**
PC1	1500	Not used
PC2	500	Not used
PC3	1500	1500

Table 5.2. MTU sizes.

In Linux, you can view the MTU values of all interfaces using the `ifconfig` command. For example, on PC2, you type

```
PC2% ifconfig
```

The same command is used to modify the MTU value. For example, to set the MTU value of interface *eth0* on PC2 to 500 bytes, use the `ifconfig` command as follows:

```
PC2% ifconfig eth0 mtu 500
```

2. Start Wireshark on `eth0` of both PC1 and PC2, and start to capture traffic with no filters set.

3. On PC2, set up the system to listen to the `echoping` on port number "7". Note that the default for `ncat` is to use TCP. The command is different from the previous ncat command.

```
PC2% ncat -l 7 -k -e "/bin/cat"
```

4. Issue an `echoping` command from PC1 to PC2 to generate TCP traffic with the following command:

```
PC1% echoping -v -s 512 10.0.2.22
```

5. Stop Wireshark capture and save the output. Analyze the captured traffic.
   a) Do you observe fragmentation? If so, where does it occur? Explain your observation.

6. Restart Wireshark on `eth0` of both PC1 and PC2 with no filters set.

7. Repeat Step 4-6, incrementing the size of the TCP segments, by increasing the argument given with the "-s" option to 256, 512, 1024, 2048, and 5000.

8. Stop the Wireshark capture and save the output. Analyze the captured traffic.
   a) Do you observe fragmentation? If so, where does it occur? Explain your observations.
   b) Explain why there is no ICMP error message generated at this point. Is the DF bit set in the IP datagrams?

9. On PC2 kill the `ncat` TCP process by typing: `fuser -k 7/tcp`

10. Now change the MTU size of interface eth1 on PC3 to 500 bytes. Change the MTU size of interface eth0 on PC2 to 1500 bytes.

11. Repeat Steps 2 - 9.

12. Make sure that when you are done, to kill the `ncat` TCP process on PC2 by typing: `fuser -k 7/tcp`

13. When done with this exercise, reset the MTU value to 1500 on all eth0 and eth1 interfaces of the PCs.

Lab Questions:

- Do you observe fragmentation? If so, where does it occur? Explain your observation.
- If you observe ICMP error messages, describe how they are used for Path MTU Discovery. Look at the first TCP segment that is sent after PC1 has received the ICMP error message. Note the segment size.

PART 3. TCP CONNECTION MANAGEMENT

TCP is a connection-oriented protocol. The establishment of a TCP connection is initiated when a TCP client sends a request for a connection to a TCP server. The TCP server must be running when the connection request is issued.

TCP requires three packets 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 can be closed separately, requiring 4 packets. This is called a half-close on each side. If both sides close at the same time, then the FIN packet and the ACK can be combined and transmitted in the same segment, giving rise to only 3 packets for closing.

TCP does not have separate control packets for opening and closing connections. Instead, 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.

Here, you will use Telnet to set up a TCP connection and observe the control packets that establish and terminate a TCP connection. The experiments involve PC1 and PC2 in the network shown in Figure 5.1.

Exercise 3(A). Opening and closing a TCP connection

Set up a TCP connection and observe the packets that open and close the connection. Determine how the parameters of a TCP connection are negotiated between the TCP client and the TCP server.

1. This part of the lab uses PC1 and PC2 set up as in the network configuration shown in Figure 5.1. If the network is not set up, follow the instructions of Exercise 1(A).

2. First set up a telnet server on PC2 by editing and configuring three files. The commands to do so are as follows:

   - The first file we will be configuring is /etc/inetd.conf. Create a new file with vi and edit it to include this line: `telnet stream tcp nowait telnetd /usr/sbin/tcpd /usr/sbin/in.telnetd`

```
PC2% vi /etc/inetd.conf
telnet stream tcp nowait telnetd /usr/sbin/tcpd /usr/sbin/in.telnetd
```

- Next file we will remove and re-create is a file /etc/xinetd.conf that will contain the following lines:

```
PC2% rm /etc/xinetd.conf
PC2% vi /etc/xinetd.conf
include /etc/xinetd.d/defaults
{
# Please note that you need a log_type line to be able to use
# log_on_success and log_on_failure. The default is the following:
# log_type = SYSLOG daemon info
instances = 60
log_type = SYSLOG authpriv
log_on_success = HOST PID
log_on_failure = HOST
cps = 25 30
}
```

- Lastly, we will create /etc/xinetd.d/telnet to contain the following lines:

```
PC2% vi /etc/xinetd.d/telnet
# default: on
{
disable = no
flags = REUSE
socket_type = stream
wait = no
user = root
server = /usr/sbin/in.telnetd
log_on_failure += USERID
}
```

3. Verify that the MTU values of all interfaces of PC1, PC2 and PC3 are set to 1500 bytes, which is the default MTU for Ethernet networks.

4. **Note**: please check contents of the configuration files to make sure that everything is correct before proceeding.

5. Start Wireshark on interface eth0 on PC1 to capture the traffic of the telnet connection. Do not set any filters.

6. **Establishing a TCP connection**: Establish a Telnet session from PC1 and PC2 as follows:

   a) On PC2, enable the Telnet Service with the following command:

   ```
   PC2% /etc/init.d/xinetd restart
   ```

   You can verify if the Telnet service is active or not using the command below. The service is active as long as the port number accepts connections, so port numbers should be listening to incoming connections.

   ```
   PC2% netstat -nlpt
   ```

   b) On PC1, establish a Telnet session to PC2 with the default username ***user*** and password *password* using the command below:

   ```
   PC1% telnet 10.0.5.22
   ```

7. **Closing a TCP connection (initiated by client):** On PC1, type `exit` at the Telnet prompt to terminate the connection.

8. Terminate Wireshark traffic capture and save the output

Lab Questions:

Analyze the TCP segments of the transmitted packets during connection set up:

- Identify the packets of the three-way handshake. Which flags are set in the TCP headers? Explain how these flags are interpreted by the receiving TCP server and TCP client.
- During the connection setup, the TCP client and TCP server tell each other the initial sequence number (ISN#) they will use for data transmission. What are the initial sequence numbers of the TCP client and the TCP server?
- Identify the first packet 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?
- The TCP client and TCP server exchange window sizes to get the maximum amount of data that the other side can send at any time. Determine the values of the window sizes for the TCP client and the TCP server.
- What is the MSS value that is negotiated between the TCP client and the TCP server?
- How long does it take to open a TCP connection?

Analyze the TCP segments of the transmitted packets during connection tear down:

- 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 and TCP client. How many transmissions were involved in the tear down?

Exercise 3(B). Requesting a connection to a non-existing host

Observing how a TCP client tries to establish a connection to a host that does not exist.

1. Start a new traffic capture with Wireshark on the interface eth1 of the PC1.

2. Set a static entry in the ARP table for a non existing IP address **10.0.1.100** on PC1.

```
PC1% arp –s 10.0.5.100 00:01:02:03:04:05
```

3. From PC1, establish a Telnet session to the non-existing host:

```
PC1% telnet 10.0.5.100
```

4. Terminate the traffic capture on PC1 and save the output.

Lab Questions:

- How often does the TCP client try to establish a connection? How much time elapses between repeated attempts to open a connection?
- Does the TCP client terminate or reset the connection when it gives up trying to establish a connection?
- Why does this experiment require setting a static ARP table entry?

Exercise 3(C). Requesting a connection to a non-existing port

When a host tries to establish a TCP connection to a port at a remote server, and no TCP server is listening on that port, the remote host terminates the TCP connection. This is observed in the following exercise.

1. Start a new traffic capture with Wireshark on PC1.

2. Establish a TCP connection to port 80 of PC2. Note that there is no TCP server running on PC2 that is listening on this port number.

```
PC1% telnet 10.0.5.22 80
```

3. Terminate Wireshark on PC1 and save the output.

Lab Questions:

- How does TCP at the remote host close this connection?
- How long does the process of ending the connection take?

PART 4. TCP Data Exchange

In parts 4 and 5, you study acknowledgements and flow control in TCP. The receiver of TCP data acknowledges the receipt of data in segments that have the ACK flag set. These segments are called **acknowledgments**, or ACKs. In TCP, each transmitted byte of application data has a sequence number. The sender of a segment writes the sequence number of the first byte of transmitted application data in the sequence number field of the TCP header. When a receiver sends an ACK, it writes a sequence number in the acknowledgement number field of the TCP header. The acknowledgment number is larger by 1 than the highest sequence number that the receiver wants to acknowledge. Whenever possible, a TCP receiver sends an ACK in a segment that carries a payload. This is called **piggybacking**. A TCP receiver can acknowledge multiple segments in a single ACK. This is called **cumulative acknowledgments**.

In this lab, you study acknowledgments separately for interactive applications, such as Telnet, and for bulk transfer applications, such as file transfers. You will observe that different TCP mechanisms play a role for these different types of applications. In this part, you study the data transfer of interactive applications.

Interactive applications typically generate a small volume of data. Since interactive applications are generally delay sensitive, a TCP sender does not wait until the application data fills a complete TCP segment, and instead, TCP sends data as soon as it arrives from the application. This, however, results in an inefficient use of bandwidth since small segments mainly consist of protocol headers. Here, you will observe TCP mechanisms that try to reduce the number of segments with a **small** payload from being transmitted, by consolidation. One such mechanism, called **delayed acknowledgments**, requires that the receiver of data wait for a certain amount of time before sending an ACK. If, during this delay, the receiver has data for the sender, the ACK can be piggybacked to the data, thereby saving the transmission of a segment. Another such mechanism, called **Nagle's algorithm**, limits the number of small segments that a TCP sender can transmit without waiting for an ACK. Forcing this wait, more data can accumulate in the buffer, thereby increasing the segment size.

Exercise 4(A). TCP Data Exchange - Interactive applications

Here you observe interactive data transfer in TCP, by establishing a TCP connection from PC1 to PC2 over the Ethernet link between the PCs. Depending on the type of hub, the Ethernet link has a maximum data rate of 10 Mbps or 100 Mbps.

1. Start Wireshark on PC1 for interface FastEthernet 0/1. Do not set any filters.

2. On PC1, establish a `Telnet` session to PC2 by typing

   ```
   PC1% telnet 10.0.5.22 23
   ```

   Login with **username** *`user`* and **password** *`password`*. Please refer to Exercise 4(A) to on how to start a `Telnet` server on PC2.

3. Type a few characters in the Telnet window. The `Telnet` client sends each *typed* character in a separate TCP segment to the `Telnet` server, which, in turn, *echoes* the charter back to

the client. Including ACKs, one would expect to see four packets for each typed character. However, due to delayed acknowledgments, this is not the case.

4. Don't stop the Wireshark capture.

Lab Questions:

Analyze the Wireshark output.

- Observe the number of packets exchanged between PC1 and PC2 for each keystroke. Describe the payload of the packets. Use your knowledge of delayed acknowledgments to explain the sequence of segment transmissions. Explain why you should see 4 packets and explain why you do not see 4 packets per typed character.
- When the TCP client receives the echo of a character, it waits a certain time before sending the ACK. Why does the TCP client delay? How long is this delay? How much does the delay vary?
- What is the time delay associated with the transmission of ACKs from the `Telnet` server on PC2?
- Which flags, if any, are set in the TCP segments that carry *typed* characters as payload? Explain the meaning of these flags.
- Why do segments that have an empty payload carry a sequence number? Why does this not result in confusion at the TCP receiver?
- What is the window size that is advertised by the `Telnet` client and the `Telnet` server? How does the value of the window size field vary as the connection progresses?

5. Type characters in the `Telnet` client program as fast as you can (e.g., by pressing a key and holding it down).

6. Don't stop the Wireshark capture.

Lab Questions:

- Do you observe a difference in the transmission of segment payloads and ACKs?

7. Terminate the `Telnet` session by typing in the Telnet terminal mode: `Ctrl + [` and then typing `quit`.

8. Stop the traffic capture with Wireshark and save the captured packets.

Lab Questions:

- For one character typed at the `Telnet` client, include a drawing that shows the transmission of TCP segments between PC1 and PC2 due to this character.

Exercise 4(B). TCP Data Exchange – Bulk Data Transfer

The TCP receiver can use acknowledgments to control the transmission rate at the TCP sender. This is called *flow control*. Flow control is not an issue for interactive applications, since the traffic volume of these applications is small, but plays an important role in bulk transfer applications.

Bulk data transfers generally transmit full segments. In TCP, the receiver controls the amount of data that the sender can transmit using a *sliding window flow control* scheme. This prevents the

receiver from getting overwhelmed with data. The number of bytes that the receiver is willing to accept is written in the window size field. An ACK that has values (250, 100) for the acknowledgments number and the window size tells the TCP sender that it can transmit data with sequence number 250, 251…, 349. The TCP sender may have already transmitted some data in that range.

In this part of the lab, you observe acknowledgments and flow control for bulk data transfers, with traffic generated with the `iperf` tool.

All exercises are done with the network configuration from Figure 5.1 and Table 5.1.

1. On PC1 start Wireshark on eth1 to capture TCP packets.

2. For this exercise we need to use the command `iperf`.

NAME

iperf - perform network throughput tests

SYNOPSIS

iperf -s [options]
iperf -c server [options]
iperf -u -s [options]
iperf -u -c server [options]

DESCRIPTION

iperf is a tool for performing network throughput measurements. It can test either TCP or UDP throughput. To perform an iperf test the user must establish both a server (to discard traffic) and a client (to generate traffic).

SEE ALSO

`http://iperf.sourceforge.net/`

3. On PC2, set up the system to receive data from `iperf`.

```
PC2% iperf -s
```

4. On PC1 issue an `iperf` to PC2 with the following command

```
PC1% iperf -c 10.0.5.22 -t 1 -M 1000
```

5. When `iperf` is done, stop Wireshark and save the output.

Lab Questions:

From the output of Wireshark, observe the sliding window flow control scheme. The sender transmits data up to the window size advertised by the receiver and then waits for ACKs.

- Observe the transmission of TCP segments and ACKs. How frequently does the receiver send ACKs? Is there an ACK sent for each TCP segment or less often? Can you determine the rule used by TCP to send the ACKs? Can you explain this rule?
- How much data (measured in bytes) does the receiver acknowledge in a typical ACK? What is the most data that is acknowledged in a single ACK?
- What is the range of the window sizes advertised by the receiver? How does the window size vary during the lifetime of the TCP connection?
- Select an arbitrary ACK packet in Wireshark, sent by PC2 to PC1. Locate the ACK number in the TCP header. New relate this ACK to a segment sent by PC1. Identify this segment in the Wireshark output. How long did it take from the transmission of the segment until the ACK arrives at PC1.
- Determine whether or not the TCP sender generally transmits the maximum amount of data allowed by the advertised window. Explain your answer.
- When the `iperf` sender has transmitted all of its data, it closes the connection, but ACKs from PC2 still trickle in. What does PC2 do when it has sent all the ACKs?

PART 5. RETRANSMISSIONS IN TCP

Next you observe retransmissions in TCP. TCP uses ACKs and timers to trigger retransmissions of lost segments. A TCP sender retransmits a segment when it assumes that the segment has been lost. This occurs in two situations:

- **No ACK has been received for a segment:** Each TCP sender maintains one retransmission timer for the connection. When the timer expires, the TCP sender retransmits the earliest segment that has not been acknowledged. The time is started when a segment with payload is transmitted and the timer is not running, when an ACK arrives that acknowledges new data, and when a segment is retransmitted. The timer is stopped when all outstanding data has been acknowledged.

  The retransmission timer is set to a retransmission timeout (RTO) value, which adapts to the current network delays between the sender and the receiver. A TCP connection performs round-trip measurements by calculating the delay between the transmission of a segment and the receipt of the acknowledgment for that segment. The RTO value is calculated based on these round-trip measurements. Following a heuristic called *Karn's algorithm*, measurements are not taken for retransmitted segments. Instead, when a retransmission occurs, the current RTO value is simply doubled.

- **Multiple ACKs have been received for the same segment**: A duplicate acknowledgment for a segment can be caused by an out-of-order delivery of a segment or by a lost packet. A TCP sender takes multiple, in most cases, three, duplicates as in indication that a packet has been lost. In this case, the TCP sender expedites a fast retransmit by sending an ACK for each packet that is received out of order.

A disadvantage of cumulative acknowledgments in TCP is that a TCP receiver cannot request the retransmission of specific segments. For example, if the receiver has obtained segments 1, 2, 3, 5, 6, 7 with cumulative acknowledgments the receiver can send ACKs only for segments 1, 2, 3 but not for 5, 6, 7. The problem can be remedied with an optional feature of TCP, which is called *selective acknowledgments* (SACKs). Here, in addition to acknowledging the highest sequence number of contiguous data that has been received correctly, a receiver can acknowledge additional blocks of sequence numbers. The range of these blocks is included in TCP headers as an option. Whether SACKs are used is negotiated in TCP header options when the TCP connections are created.

The exercise in this part explores aspects of TCP retransmissions that do not require access to internal timers. Unfortunately, the round-trip time measurements and the RTO values are difficult to observe and are, therefore, not included in this lab.

The network configuration for this part is the network shown in Figure 5.1 and Table 5.1.

Exercise 5(A). TCP Retransmissions

The purpose of this exercise is to observe when TCP retransmissions occur. In this part of the lab, you will transmit data from PC1 to PC2 through PC3. When you logically disconnect the

connection to PC2, ACKs cannot reach the sending host PC1. As a result, a timeout occurs and the sender performs retransmissions.

1. On PC1 start Wireshark on eth0 to capture TCP packets (set TCP filter).

2. On PC3 enable IP forwarding with the following command:

```
PC3% echo "1" > /proc/sys/net/ipv4/ip_forward
```

   Note: Check to make sure that the default gateways for PC1 and PC2 are set as shown in Table 5.1.

3. For this exercise we will use the command `iperf`.

```
PC1% iperf -c 10.0.2.22 -t 20 -M 1000
```

4. Disable ip forwarding on PC3 with the following command:

```
PC3% echo "0" > /proc/sys/net/ipv4/ip_forward
```

5. In the Wireshark output, observe the retransmissions from PC1.

6. Re-enable ip forwarding on PC3. Now quickly disable and enable ip forwarding several times, varying the length of time that the ip forwarding is disabled.

7. Stop Wireshark and save the captured traffic.

Lab Questions:

Analyze the Wireshark output and answer the following questions:

- When the connection is created, do the TCP sender and TCP receiver negotiate to permit SACKs? Describe the process of negotiation.
- When you first disable ip forwarding, observe the time instants when retransmissions took place. How many packets were retransmitted at one time?
- Try to derive the algorithm that sets the time when a packet is retransmitted. Use data to backup your answer. Is there a maximum time interval between retransmissions?
- After how many retransmissions, if at all, does the TCP sender stop to retransmit the segment? Describe your observations.
- After you re-enable ip forwarding in Step 5, and disable and enable ip forwarding rapidly, do you notice any difference in the retransmissions from those observed in Step 3? Specifically, do you observe fast retransmits and/or SACKs?