Based on
Mastering Networks - An Internet Lab Manual
by Jörg Liebeherr and Magda Al Zarki

Adapted for
'Labo Computernetwerken'
by Johan Bergs, Nicolas Letor, Michael Voorhaen and Kurt Smolderen

Completed by
Sam Mylle Frederico Quin Stan Draulans

April 23, 2017

Lab 5

Transport Layer Protocols: UDP and TCP

What you will learn in this lab:

- The differences between data transfers with UDP and with TCP
- What effect IP Fragmentation has on TCP and UDP
- How to analyze measurements of a TCP connection
- The difference between interactive and bulk data transfers in TCP
- How TCP performs retransmissions
- How TCP congestion control works

5.1 Prelab 5

TCP and UDP

Use the following resources to prepare yourself for this lab session:

1. TCP and UDP: Read the overview of TCP and UDP available at http://en.wikipedia.org/wiki/Transmission_Control_Protocol and http://en.wikipedia.org/wiki/User_Datagram_Protocol.
2. IP Fragmentation: Refer to the website http://www.tcpipguide.com/free/t_IPMessageFragmentationProcess.htm for information on IP Fragmentation and Path MTU Discovery.
3. TCP Retransmissions: Refer to RFC 2988, which is available at http://tools.ietf.org/html/rfc2988, and read about TCP retransmissions.
4. TCP Congestion Control: Refer RFC 2001, which is available at http://tools.ietf.org/html/rfc2001, and read about TCP congestion control.

Prelab Questions

Question 1)

Explain the role of port numbers in TCP and UDP.
Server or daemon software may listen to a certain port (with a port number of course). It may provide a service when receiving a message on this port. For example, a webserver listens to port 80.

Question 2)

Provide the syntax of the `ttcp` command for both the sender and receiver, which executes the following scenario: A TCP server has IP address 10.0.2.6 and a TCP client has IP address 10.0.2.7. The TCP server is waiting on port number 2222 for a connection request. The client connects to the server and transmits 2000 bytes to the server, which are sent as 4 write operations of 500 bytes each.
Server-side: nttcp -i -l500 -n4 -p2222
Client-side: nttcp -ts -l500 -n4 -p2222 10.0.2.6

Question 3.a)

How does TCP decide the maximum size of a TCP segment?
It is specified as an option in the TCP header. Note that the header is not kept into account in this field.

Question 3.b)

How does UDP decide the maximum size of a UDP datagram?
It is specified in the UDP header. Note that the header is kept into account in this field.

Question 3.c)

What is the ICMP error generated by a router when it needs to fragment a datagram with the DF bit set? Is the MTU of the interface that caused the fragmentation also returned?
While issuing the following command: ping -c 1 -s 2000 -Mdo 192.168.0.133
We get the following output:
PING 192.168.0.133 (192.168.0.133) 2000(2028) bytes of data.
ping: local error: Message too long, mtu=1500

— 192.168.0.133 ping statistics —
1 packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms

As you can see, the router informs us that the packet cannot be sent due to its size and the DF bit. The MTU is also returned (here 1500).

Question 3.d)

Explain why a TCP connection over an Ethernet segment never runs into problems with fragmentation.
Because a TCP header has no DF bit.

Question 4)

Assume a TCP sender receives an acknowledgement (ACK), that is, a TCP segment with the ACK flag set, where the acknowledgement number is set to 34567 and the window size is set to 2048. Which sequence numbers can the sender transmit?
The sender can send sequence numbers 34568 to 36615.

Question 5.a)

Describe Nagle's algorithm and explain why it is used in TCP
Assume you have data to send.
If the window size allows it and if the available data is larger than the maximum segment size, then you shoud send a complete segment now (with a size of "maximum segment

size").
If the above does not hold and there is still unconfirmed data, then you have to enqueue the data in the buffer until an acknowledgment is received.
If the above does not hold, then you just have to send the data immediately.

The reason this algoritmh is used is because it sends small packets together in a full packet's worth of output. This reduces the overhead of acknowledgments.
For example: if you send 8 small packets, the receiver has to send 8 acknowledgments. If you send those 8 small messages in one packet, the sender only has to send one acknowledgment.

Question 5.b)

Describe Karn's Algorithm and explain why it is used in TCP
Karn's algoritmh is used to get an accurate estimate of the round-trip time for messages using TCP.
The round-trip time is estimated by calculating the difference between the time that a packet was sent and the time that the sender receives the corresponding aknowledgment.
The problem here are retransmissions: is the acknowledgment a response to the first message or its retransmission? Karne's algorithm simply ignores these ambiguous acknowledgments.
A new problem arises: if the delay is permanently increased, every packet will be retransmitted and the estimate will never be updated. The solution to this is to keep a backoff timer initialized using the round-trip time estimate . For each retransmission, this timer is doubled.

Question 6.a)

What is a delayed acknowledgement in TCP?
A delayed acknowledgment is a technique used to decrease the overhead of TCP acknowledgments. Instead of sending the acknowledgment right away, it waits and combines several acknowledgments to be sent in one packet.

Question 6.b)

What is a piggybacked acknowledgement in TCP?
When a sender sends a packet to a receiver (TCP), the receiver will have to send an acknowledgment. But when the receiver has data for the sender, he can send the data and the acknowledgment in one packet.
This is called a piggybacked acknowledgment.

Question 7)

Describe how the retransmission timeout (RTO) value is determined in TCP.
The initial value of a timer is calculated using some estimate of the round-trip time.
When this timer expires, the packet is assumed to be lost (potentially incorrect). The timeout value will be doubled, giving the retransmissions more time.

Question 8.a)

Describe the sliding window flow control mechanism used in TCP .
A sender and a receiver both have a window size N.
Whenever a sender has data to send, he checks the window size, the sequence number of the packet to be sent (n_s) and the highest sequence number that was acknowledged by the receiver (n_a. When $n_a < n_s <$ N + n_a, the sender may send the packet.
The receiver will only accept packets the same way the sender checks if he can send the packet.
Whenever a receiver receives and accepts a packet, he will update his n_a.
Whenever a sender receives an acknowledgment, he will update his n_a.

Question 8.b)

Describe the concepts of slow start and congestion avoidance in TCP.

Slow start begins with a congestion window size that will be incremented every time an ACK is received.

When a packet is lost or the receiver advertised a limiting window size or the slow start treshold is reached, the window will not be increased when receiving an ACK.

On packet loss, TCP takes steps to reduce the load.

When the slow start treshold is reached, the congestion window will increase by one every round-trip time.

Question 8.c)

Explain the concept of fast retransmit and fast recovery in TCP.

After receiving a packet, the receiver sends an acknowledgement to the sender.

Assume some packet has been lost. In the meantime the receiver receives the packets that followed the lost packet. After receiving these packets, the receiver sends an acknowledgement, but still only for the sequence number of the lost packet (indicating it still has to receive this packet). This results in duplicate acknowledgments.

Fast retransmit works as follows: if the sender receives a specified number (assume 3) of acknowledgements with the same acknowledge number, the sender will retransmit the packet that was lost before waiting for its timeout. Fast recovery works as follows: if the sender receives a specified number of acknowledgements (assume 3) with the same acknowledge number, it sets the slow start treshold to half of the congestion window. Then, it retransmits the lost packet, sets its congestion window (= cwnd) to cwnd + 3 and continues with slow start. Once you receive an ACK that acknowledges all packets sent between the lost packet and the first reception of a duplicate ACK, exit fast recovery (set cwnd to the slow start treshold).

5.2 Lab 5

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

The operation of TCP is more complex. First, TCP is a connection-oriented protocol, where 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 which 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, acknowledgements, 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. 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, 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. Parts 1 and 2 compare the performance of data transmissions in TCP and UDP. Part 3 explores how TCP and UDP deal with IP fragmentation. The remaining parts address important components of TCP. Part 4 explores connection management, Parts 5 and 6 look at flow control and acknowledgements, Part 7 explores retransmissions, and Part 8 is devoted to congestion control.

This lab has two different network topologies. The topology for Parts 1-4 is shown in Figure 5.1. In this configuration, PC1 and PC2 are used as hosts, and PC3 is set up as an IP router. The network configuration for Parts 5-8 is shown in Figure 5.2. Here, the four Cisco routers interconnected via Ethernet, where one of the links is configured to emulate a slow serial link, as show in Figure 5.2.

Part 1. Learning how to use nttcp

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

Some useful `nttcp` commands:

`nttcp -u -i -rs -lbuflen -nnumbufs -pport`
: *Start a nttcp receiver process*

`nttcp -u -ts -lbuflen -nnumbufs -pport -D IPaddress`
: *Start a nttcp sender process*

List of important `nttcp` options:

`-t`
: *Specifies the transmit mode.*

`-r`
: *Specifies the receive mode*

`-u`
: *Specifies to use UDP instead of TCP. By default, nttcp uses TCP to send data. Make sure this is the first option*

`-s`
: *Sends a character string as payload of the transmitted packets. Without the -s option, the default is to transmit data from the terminal window (stdin) of the sender and to print the received data to the terminal window (stdout) at the receiver*

`-nnumbufs`
: *Number of blocks of application data to be transmitted (Default value is 2048)*

`-lbuflen`
: *Length of the application data blocks that are passed to UDP or TCP in bytes (default 8192). 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. Only used 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*

By default, nttcp transmits data over a TCP connection. The nttcp sender opens a TCP connection to a nttcp receiver, transmits data and then closes the connection. The nttcp receiver must be running when the nttcp sender is started. UDP data transfer is specified with the `-u` option. Since UDP is a connectionless protocol, the nttcp sender starts immediately sending UDP datagrams, regardless if there is a nttcp receiver established or not.

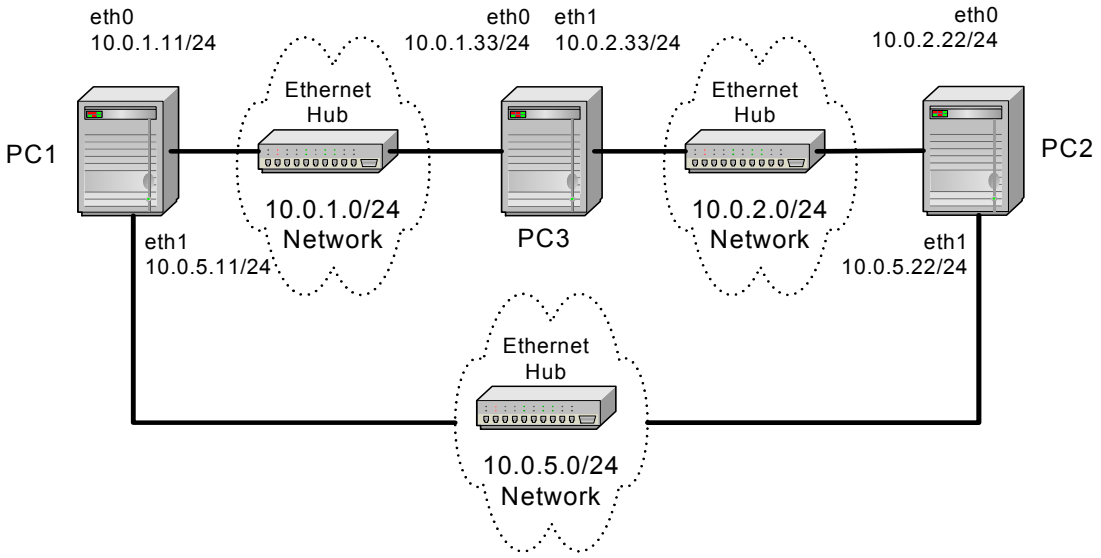


Figure 5.1: Network Topology for Parts 1-4.

Linux PC	Ethernet Interface eth0	Ethernet Interface eth1
PC1	10.0.1.11/24	10.0.5.11/24
PC2	10.0.2.22/24	10.0.5.22/24
PC3	10.0.1.33/24	10.0.2.33/24

Table 5.1: IP Addresses of the Linux PCs.

Exercise 1-a. Network setup

1. Connect the Ethernet interfaces of the Linux PCs as shown in Figure 5.1. Configure the IP addresses of the interfaces as given in Table 5.1.

2. PC1 and PC2 are set up as hosts and IP forwarding should be disabled. On PC1, 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. Add default routes to the routing tables of PC1 and PC2, so that PC3 is the default gateway. For PC1 the command is as follows:

```
PC1% route add default gw 10.0.1.33
```

5. Verify that the setup is correct by issuing a `ping` command from PC1 to PC2 over both paths:

```
PC1% ping 10.0.2.22
PC1% ping 10.0.5.22
```

Exercise 1-b. Transmitting data with UDP

This exercise consists of setting up a UDP data transfer between two hosts, PC1 and PC2, and observe the UDP traffic.

1. On PC1, start Wireshark to capture packets on interface *eth1* between PC1 to PC2.

   ```
   PC1% wireshark -i eth1 -f 'host 10.0.5.22'
   ```

   This sets a capture filter to packets that include IP address 10.0.5.22 in the IP header. Start to capture traffic.

2. On PC2, start a nttcp receiver that receives UDP traffic with the following command:

   ```
   PC2% nttcp -u -i -rs -l1024 -n10 -p4444
   ```

3. On PC1, start a nttcp sender that transmits UDP traffic by typing:

   ```
   PC1% nttcp -u -ts -l1024 -n10 -p4444 10.0.5.22
   ```

   Observe the captured traffic captured by Wireshark.

4. Stop the Wireshark capture on PC1 and save the captured traffic.

Use the data captured with Wireshark to answer the questions in Step 3. Support your answers with the saved Wireshark data.

Question 1.B.1.a)
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?
13 TCP packets are transferred, as well as 14 UDP packets. There is one packet being transmitted per UDP datagram. The UDP payload size is 1024 bytes, except for the first and last UDP packet which only have a payload of 4 bytes.

Question 1.B.1.b)
Compare the total number of bytes transmitted, in both directions, including Ethernet, IP, and UDP headers, to the amount of application data transmitted.
There were 13278 bytes transmitted (of which 102 ARP and 148 ICMP (destination unreachable)), for 10240 bytes of application data.

Question 1.B.1.c)
Inspect the fields in the UDP headers. Which fields in the headers do not change in different packets?
Source port, destination port, length and checksum, but length and checksum do change for the first and last udp packets.

Question 1.B.1.d)
Observe the port numbers in the UDP header. How did the nttcp sender select the source port number?
It picked a port from the range 49152-65535 range, which is the dynamic and/or private ports range.

Exercise 1-c. Transmitting data with TCP

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

1. On PC1, start Wireshark and capture packets on interface *eth1* between PC1 to PC2:

   ```
   PC1% wireshark -i eth1 -f 'host 10.0.5.22'
   ```

2. Start a nttcp receiver on PC2 that receives packets sent by PC1:

   ```
   PC2% nttcp -i -rs -l1024 -n10 -p4444
   ```

3. Start a nttcp sender on PC1o that transmits packets from PC1 to PC2:

   ```
   PC1% nttcp -ts -l1024 -n10 -p4444 -D 10.0.5.22
   ```

Use the data captured with Wireshark to answer the questions in Step 3. Support your answers by including data from the saved Wireshark data captures.

Question 1.C.1.a)
How many packets are exchanged in the data transfer? What are the sizes of the TCP segments?
There are 34 packets exchanged in the data transfer (36 with the arp request included). Of these, 9 (attempt to) transmit application data. The sizes of the TCP segments are maximum 1448 bytes of data, encapsulated by 66 bytes of header, forming a total of 1514 bytes.

Question 1.C.1.b)
What is the range of the sequence numbers?
The range of the sequence numbers goes from 1 to 9713.

Question 1.C.1.c)
How many packets are transmitted by PC1 and how many packets are transmitted by PC2?

PC1 transmitted 18 packets. PC2 transmitted 16 packets.

Question 1.C.1.d)
How many packets do not carry a payload, that is, how many packets are control packets?

Of the 36 packets transmitted total, 9 carry a payload (of which 1 is a retransmission), and 2 are ARP.

Question 1.C.1.e)
Compare the total number of bytes transmitted, in both directions, including Ethernet, IP, and UDP headers, to the amount of application data transmitted.
Of the 14638 bytes of total data (of which 102 are ARP), 10768 bytes are (unique) application data.

Question 1.C.1.f)
Inspect the TCP headers. Which packets contain flags in the TCP header? Which types of flags do you observe?
All TCP packets contain flags in their header. Flags observed in this transmission were SYN, ACK, PSH and FIN.

4. Stop the wireshark capture on PC1, and save the captured traffic to files. Save both the summary and detail output in the Print menu.

Question 1.C.2)

Compare the amount of data transmitted in the TCP and the UDP data transfers.
For UDP, there were 13278 bytes transmitted (of which 102 ARP and 148 ICMP (destination unreachable)), for 10240 bytes of application data. For TCP there were 14638 bytes transmitted (of which 102 are ARP) for 10768 bytes of (unique) application data.

Question 1.C.3)

Take the biggest UDP datagram and the biggest TCP segment that you observed, and compare the amount of application data that is transmitted in the UDP datagram and the TCP segment.
The biggest UDP datagram was 1066 bytes, containing 1024 bytes of application data: this results in 42 bytes of header. The biggest TCP segment was 1514 bytes, containing 1448 bytes of application data: this results in 66 bytes of header.

Part 2. File Transfers using TCP and UDP

Here you compare the throughput of a file transfer with TCP and UDP, using the application programs `ftp` and `tftp`.

The File Transfer Protocol (FTP) for copying files between hosts, as described in the Introduction, employs TCP as its transport protocol, thereby ensuring a reliable transfer of transmitted data. Two TCP connections are established for each FTP session: A control connection for exchanging commands, and a data connection for the file transfer.

The Trivial File Transfer Protocol (TFTP) is a minimal protocol for transferring files without authentication. TFTP employs UDP for data transport. A TFTP session is initiated when a TFTP client sends a request to upload or download a file to UDP port 69 of a TFTP server. When the request is received, the TFTP server picks a free UDP port and uses this port to communicate with the TFTP client. Since UDP does not recover lost or corrupted data, TFTP is responsible for maintaining the integrity of the data exchange. TFTP transfers data in blocks of 512 bytes. A block must be acknowledged before the next block can be sent. When an acknowledgment is not received before a timer expires, the block is retransmitted.

The purpose of the following exercises is to observe that, despite the overhead of maintaining a TCP connection, file transfers with FTP generally outperform those with UDP.

Exercise 2. Comparison of FTP and TFTP

Study the performance of FTP and TFTP file transfers for a large file.

1. Create a large file. On PC1, create a file with name `large.d` in directory `/tftpboot` by using the `dd` tool. Create the file and change its access permissions as follows:

   ```
   PC1% dd if=/dev/urandom of=/tftpboot/large.d bs=1024 count=1024
   PC1% chmod 644 /tftpboot/large.d
   ```

   Use the command `ls -l` to check the length of the file.

2. Start Wireshark: Invoke Wireshark on interface *eth1* of PC1 with a capture filter set for PC2, and start to capture traffic:

   ```
   PC1% wireshark -i eth1 -f âĂIJhost 10.0.5.22âĂİ
   ```

3. FTP file transfer: On PC2, perform the following steps:

   - Change the current directory to `/labdata`.
   - Start the FTP server on PC1 by typing

     ```
     PC1% service vsftpd start
     ```

   - Invoke an FTP session to PC1 by typing

     ```
     PC2% ftp 10.0.5.11
     ```

     Log in as the root user.

   - Transfer the file large.d from PC1 to directory `/labdata` on PC2 by typing

     ```
     ftp> cd /tftpboot
     ftp> get large.d
     ftp> quit
     ```

4. TFTP file transfer: On PC2 perform the following tasks:
   - Start the TFTP server on PC1 by typing
     ```
     PC1% service tftpd-hpa start
     ```
   - Start a TFTP session to PC1 by typing
     ```
     PC2% tftp 10.0.5.11
     ```
   - Transfer the file `large.d` from PC1 to PC2.
     ```
     tftp> get large.d
     tftp> quit
     ```
     By default, TFTP copies data from the directory `/tftpboot`.
   - Observe the output of the TFTP session and save the output to a file.
5. Analysis of outcome: On PC1, stop the Wireshark output.

Include the answers to the questions in Step 5.

Question 2.A.a)
From the timestamps recorded by wireshark, obtain the times it took to transfer the file with FTP and with TFTP? Use your knowledge of FTP, TFTP, TCP, and UDP to explain the outcome.
$19.516716574 - 18.285926392 = 1.230790182$ seconds for the FTP transfer, $138.673506602 - 136.988383918 = 1.685122684$ seconds for the TFTP transfer. For simplicity's sake we will define an $ACKtime$ as the $propagationdelay$ + an ACK packet's $transmissiondelay$. The results above can be explained as follows: FTP can transfer TCP segments without having to wait for acknowledgements (TCP delayed ACK), which leads to it reducing time spent on waiting for acknowledgements by $(n-1) * ACKtime$ when able to send n packets without acknowledgement. This leads to a total of $ceil(\frac{packetssent}{n}) * ACKtime$ when transfering data. For TFTP this n is 1, and therefore for every packet it sends it must receive an ACK, leading to $packetssent * ACKtime$ delay.

Question 2.A.b)
Identify the TCP connections that are created in the FTP session, and record the port numbers at the source and at the destination.
There are 2 connections in the FTP session: one for control, and one for data transfer. Ports for FTP control are: $Client : 33366$, $Server : 21$. Ports for FTP data & data ACK are: $Client : 34114$, $Server : 20$.

Part 3. 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 DonâĂŹt fragment (DF) flag is set in the IP header, the IP datagram is discarded.

When an IP datagram is fragmented, its payload is split into 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 higher layers, it incurs considerable overhead, and, therefore, should be avoided. TCP tries to avoid fragmentation with an Path MTU Discovery scheme that determines a Maximum Segment Size (MSS) which does not result in fragmentation.

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 3-a. UDP and Fragmentation

In this exercise you observe IP fragmentation of UDP traffic. In the following exercise, use `nttcp` to generate UDP traffic between PC1 and PC2, across IP router PC3, and gradually increase the size of UDP datagrams until fragmentation occurs. You can 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* interfaces of both PC1 and PC2, and start to capture traffic. Do not set any filters.
3. Use `nttcp` to generate UDP traffic between PC1 and PC2. The connection parameters are selected so that IP Fragmentation does not occur initially.
   - On PC2, execute the following command:
     ```
     PC2% nttcp -u -i -rs -l1024 -n12 -p4444
     ```
   - On PC1, execute the command:
     ```
     PC1% nttcp -u -ts -l1024 -n12 -p4444 10.0.2.22
     ```
4. Increment the size of the UDP datagrams, by increasing the argument given with the `-l` option.
   - Determine the exact UDP datagram size at which fragmentation occurs.
   - Determine the maximum size of the UDP datagram 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 for to be handled by IP.
5. Stop the traffic capture on PC1 and PC2, and save the Wireshark output.

Question 3.A.1)

From the saved Wireshark data, select one IP datagram that is fragmented. Include 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, DonâĂŹt Fragment Bit, More Fragments Bit).

Before fragmentation:

This is impossible to export in wireshark.

After fragmentation:

First packet:

```
No.     Time           Source                Destination           Protocol Length
    Info
      1 0.000000000    10.0.1.11             10.0.2.22             IPv4     1514
        Fragmented IP protocol (proto=UDP 17, off=0, ID=34af)

Frame 1: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on
    interface 0
    Interface id: 0 (eth0)
    Encapsulation type: Ethernet (1)
    Arrival Time: Mar 27, 2017 10:21:02.366463679 CEST
    [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: 1490602862.366463679 seconds
    [Time delta from previous captured frame: 0.000000000 seconds]
    [Time delta from previous displayed frame: 0.000000000 seconds]
    [Time since reference or first frame: 0.000000000 seconds]
    Frame Number: 1
    Frame Length: 1514 bytes (12112 bits)
    Capture Length: 1514 bytes (12112 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: eth:ethertype:ip:data]
Ethernet II, Src: IntelCor_36:33:a0 (68:05:ca:36:33:a0), Dst: IntelCor_36:39:c7
    (68:05:ca:36:39:c7)
    Destination: IntelCor_36:39:c7 (68:05:ca:36:39:c7)
        Address: IntelCor_36:39:c7 (68:05:ca:36:39:c7)
        .... ..0. .... .... .... .... = LG bit: Globally unique address (factory
            default)
        .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
    Source: IntelCor_36:33:a0 (68:05:ca:36:33:a0)
        Address: IntelCor_36:33:a0 (68:05:ca:36:33:a0)
        .... ..0. .... .... .... .... = LG bit: Globally unique address (factory
            default)
        .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
    Type: IP (0x0800)
Internet Protocol Version 4, Src: 10.0.1.11 (10.0.1.11), Dst: 10.0.2.22 (10.0.2.22)
    Version: 4
    Header Length: 20 bytes
    Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (
        Not ECN-Capable Transport))
        0000 00.. = Differentiated Services Codepoint: Default (0x00)
        .... ..00 = Explicit Congestion Notification: Not-ECT (Not ECN-Capable
            Transport) (0x00)
    Total Length: 1500
    Identification: 0x34af (13487)
    Flags: 0x01 (More Fragments)
        0... .... = Reserved bit: Not set
        .0.. .... = Don't fragment: Not set
        ..1. .... = More fragments: Set
    Fragment offset: 0
    Time to live: 64
    Protocol: UDP (17)
    Header checksum: 0x0942 [validation disabled]
        [Good: False]
        [Bad: False]
```

```
47      Source: 10.0.1.11 (10.0.1.11)
        Destination: 10.0.2.22 (10.0.2.22)
49      [Source GeoIP: Unknown]
        [Destination GeoIP: Unknown]
51  Data (1480 bytes)

53  0000  a4 2f 13 ae 06 08 39 36 14 72 ba a5 93 5b 4c 7b   ./....96.r...[L{
    0010  3d f3 7d f0 ae 95 39 07 c7 80 76 11 3b 82 f6 e7   =.}...9...v.;...
55  0020  aa c5 75 64 d8 c6 ab 8f 3e 6f 74 2a ce 1d 02 10   ..ud....>ot*....
    0030  3a f9 85 56 55 20 e4 0f e8 69 4f 0d 97 71 ba 8c   :..VU ...iO..q..
57  0040  59 b7 25 9e 9c 67 b2 81 9f 67 2d f1 d9 0d ac 8e   Y.%..g...g-.....
    0050  01 c2 6d cb 1d a0 e6 b3 15 9f 8b ad ad 01 0c d0   ..m.............
59  0060  c8 ad 21 1e 95 30 dd 1b 2f 7b 2a e9 64 76 56 28   ..!..0../{*.dvV(
    0070  b4 4c 51 5c ca e3 85 ab f8 dd 84 6a bc 50 fd e1   .LQ\.......j.P..
61  0080  57 c9 f6 74 cb 87 aa 56 2c 6e fb 08 7a 14 ee ff   W..t...V,n..z...
    0090  78 10 3e 3b 4b bb 24 18 bb e1 d1 0f b5 80 0f 72   x.>;K.$........r
63  00a0  22 b9 72 0e a2 40 fa bb 4b 20 f3 99 bf 83 18 08   ".r..@..K ......
    00b0  5e 30 9a cf 7a a2 c6 ba 02 61 14 2b cf 14 37 29   ^0..z....a.+..7)
65  00c0  30 f2 91 36 32 3f bb 6c f6 84 12 0f 8c c5 8a 8b   0..62?.l........
    00d0  0c 20 8a 52 04 06 a6 74 3e a2 e9 1a 82 a0 b5 c0   . .R...t>.......
67  00e0  8c f0 cd 7a e5 3d a3 98 78 18 07 0c a6 3b 37 16   ...z.=..x....;7.
    00f0  54 39 d8 c5 b5 34 8e 2c 9e f9 ec 95 8d 03 54 06   T9...4.,......T.
69  0100  95 57 3c c6 f6 f4 cf 91 af fe 96 0b a0 74 c4 29   .W<..........t.)
    0110  69 fa 26 d7 bf e7 4f 49 71 4d fc e9 1b 9f d0 bc   i.&...OIqM......
71  0120  bf 61 c1 17 64 14 f5 56 b9 34 22 42 64 87 5a 95   .a..d..V.4"Bd.Z.
    0130  64 00 9a 4b f6 7f 99 d3 fa 76 e2 fc 05 6d 05 d8   d..K.....v...m..
73  0140  31 99 97 39 1d 6b 16 77 0e 48 bb c0 0c df 36 6f   1..9.k.w.H....6o
    0150  f2 84 3c af 79 3c 9b 33 33 6b a4 5b a5 0c 9a bc   ..<.y<.33k.[....
75  0160  18 91 a4 77 0a 63 36 72 50 8e c0 72 f3 b5 d0 61   ...w.c6rP..r...a
    0170  25 1e 38 51 74 a7 c6 b5 c0 42 cf b1 ee 73 c7 21   %.8Qt....B...s.!
77  0180  91 61 3a 3a c2 d0 bf a1 da 71 54 8e c7 d7 de 36   .a::.....qT....6
    0190  27 4c bd 74 86 33 73 76 fb bd b3 16 8c 98 a9 e0   'L.t.3sv........
79  01a0  8e 7f 62 49 09 bf 68 3c ff 47 17 41 d7 33 93 b9   ..bI..h<.G.A.3..
    01b0  59 6a be de 75 ea d8 70 23 dc 66 4b a5 14 70 5b   Yj..u..p#.fK..p[
81  01c0  3c 36 63 e0 12 95 c7 33 f5 12 df da fc b0 f8 03   <6c....3........
    01d0  cb 71 e6 93 70 8d bb bd de d6 fb fe 85 c1 25 31   .q..p.........%1
83  01e0  7c ae 17 ea 83 30 09 42 0b 8d 67 0a b3 61 ee 91   |....0.B..g..a..
    01f0  82 b6 22 e1 7d 31 fa 09 37 30 e7 07 b2 06 84 07   ..".}1..70......
85  0200  7a 41 ce db 88 33 f1 00 63 ef dc 3a 64 18 0e 09   zA...3..c..:d...
    0210  3c 8e 50 dc 9d d3 21 5d 00 03 1e bb b0 43 ea d3   <.P...!].....C..
87  0220  22 88 8f 13 6e ff 10 01 69 e6 12 fb 27 f8 e0 f3   "...n...i...'...
    0230  01 7a 1a f8 9d 9e 4c 64 ba b0 0d 66 d0 2e 5c bc   .z....Ld...f..\.
89  0240  25 e4 b7 11 d1 7e d2 aa dc b7 82 a7 e0 c9 32 99   %....~........2.
    0250  8f 43 68 57 92 38 45 1e 0f 7b 2b 64 b6 a8 d0 9f   .ChW.8E..{+d....
91  0260  86 8a 53 45 b1 94 60 c5 5d da 98 c0 ee aa 8a 22   ..SE..`.]......"
    0270  07 90 f1 3e 61 a6 04 f0 f7 28 d3 6a b7 5d c8 63   ...>a....(.j.].c
93  0280  de 4f 4b f3 e2 ac 26 aa ef cc 2e 7b 8d 93 81 81   .OK...&....{....
    0290  75 f2 fc c7 f2 d8 57 10 9a 11 8e 1c 8d 95 33 db   u.....W.......3.
95  02a0  70 32 f6 7b 99 11 0b ae 40 98 04 6a bd c0 ce 5a   p2.{....@..j...Z
    02b0  9b 9a f0 fe ee c4 b6 7a 8f 4b 3d 5f 85 f2 e9 b2   .......z.K=_....
97  02c0  3f d3 06 13 a5 c0 04 2d 66 4f af cb 56 9e 16 5d   ?......-fO..V..]
    02d0  95 e1 62 f3 1f 49 94 10 5b e3 d6 72 a0 68 24 d0   ..b..I..[..r.h$.
99  02e0  c0 70 92 02 24 62 e4 f3 d6 d4 9c b9 59 ca 19 87   .p..$b......Y...
    02f0  52 d9 a8 b6 6d 54 f7 b7 10 67 bf 1c 14 43 48 dd   R...mT...g...CH.
101 0300  5b 8a 41 e8 4c cb 5a 79 a8 0d cc 79 78 0f f7 64   [.A.L.Zy...yx..d
    0310  7c 95 63 dd 51 f0 43 a1 46 f5 88 67 43 3d 15 72   |.c.Q.C.F..gC=.r
103 0320  83 80 e8 93 e0 07 e0 0a 38 a4 11 e0 0b 76 81 13   ........8....v..
    0330  d9 57 61 c0 3f 90 ab b7 4d b7 8b a4 2f 15 bb c3   .Wa.?...M.../...
105 0340  12 64 86 2c 18 57 1e 27 f6 2f f5 cc 23 6a cd bd   .d.,.W.'./..#j..
    0350  b7 f7 bd 44 f8 62 a5 0d 10 9b ab ec 1c ea 7a c2   ...D.b........z.
107 0360  fe d3 00 4f 79 a2 1d b5 c3 b6 17 cb 12 e8 13 06   ...Oy...........
    0370  6c d4 27 e8 a3 f8 b5 79 aa 87 2d 99 9f 6f 5b f5   l.'....y..-..o[.
109 0380  ec b3 68 9e f7 d3 3b fe d5 76 c2 e2 9b 78 ee cc   ..h...;..v...x..
    0390  14 58 47 4a 04 5d 65 d9 f8 81 09 9a fb b7 e0 69   .XGJ.]e........i
111 03a0  04 59 24 e4 91 09 60 91 3e 19 3a dc 05 76 89 87   .Y$...`.>.:..v..
    03b0  8b 6f fd c8 bd b7 16 12 ee 52 35 eb a2 02 45 cb   .o.......R5...E.
113 03c0  72 ac 75 96 f8 aa 96 a1 86 f6 22 ac d5 78 bb 66   r.u.......".x.f
```

```
    03d0  81 ab 98 f1 b2 f4 8c 70 73 ce e7 86 bd f7 e8 41   .......ps......A
115 03e0  8e 3e 6b 63 6b 04 1d 32 7b 22 5f c9 33 1d 81 48   .>kck..2{"_.3..H
    03f0  1d cd d2 99 6c 1c 5b 3a ea c0 ee 49 f7 ff 87 d9   ....l.[:...I....
117 0400  b2 6a ae 61 18 5e 90 ac 0f d1 05 3f e4 f0 c8 6f   .j.a.^.....?...o
    0410  2c fa b3 19 3b 0f dd ca c6 49 7d 55 9e 51 02 9c   ,...;....I}U.Q..
119 0420  98 e4 f4 b0 49 a7 41 72 dc 76 00 af e0 c9 a5 6b   ....I.Ar.v.....k
    0430  1c 57 24 b3 d2 80 cb 84 b2 a3 83 5a 9f 16 c4 52   .W$........Z...R
121 0440  68 ae 71 a4 9a 31 eb d7 7a ab a5 19 ef 71 13 c0   h.q..1..z....q..
    0450  a7 48 db e9 07 99 8a 12 d0 54 5f 55 f6 6d d5 77   .H.......T_U.m.w
123 0460  65 0e a5 37 bc 53 e3 5a e7 8a 1c a7 e9 0f 25 45   e..7.S.Z......%E
    0470  42 91 eb 09 13 f2 fc 04 5f e8 de c8 4a c1 3e e0   B........_...J.>.
125 0480  26 7f 9d 6b 0f 30 02 14 f2 97 2e b2 87 a4 4d aa   &..k.0........M.
    0490  9a 5e 21 33 63 34 9a af be f9 a8 56 f5 e1 ae ed   .^!3c4.....V....
127 04a0  90 59 51 35 c0 6f 1c 4d 86 5a 41 a9 b4 b5 e8 29   .YQ5.o.M.ZA....)
    04b0  47 10 72 ca d1 1f 12 4b e2 8a 00 70 03 96 57 4a   G.r....K...p..WJ
129 04c0  5f 04 ed ea 75 88 66 aa 3b ad 05 19 3b 72 e8 6b   _...u.f.;...;r.k
    04d0  9a b4 21 a6 49 f7 19 6d d9 aa 59 44 c0 c6 72 eb   ..!.I..m..YD..r.
131 04e0  6c 21 bb 85 17 12 da 1a 18 c2 0f 1d 86 c6 39 ea   l!............9.
    04f0  36 ef 9d a5 e0 ec c0 43 d4 3a e8 8d 24 7d bf b7   6......C.:..$}..
133 0500  9f 89 4a 93 94 d4 ed e3 3a 93 28 a5 5b c7 f3 45   ..J.....:.(.[..E
    0510  1c f4 26 6f 02 c8 f5 5e 32 9e 8c 16 63 9e fb 20   ..&o...^2...c..
135 0520  e5 3f 76 ec 5e 8e d3 aa 91 f1 b3 e9 dd 1b 84 75   .?v.^..........u
    0530  2a 2a 5f 33 da 0b aa d6 ab 4e 25 28 2b e5 33 85   **_3.....N%(+.3.
137 0540  5e 3f 53 ec b5 6b 25 a0 0c d2 ec aa 51 e9 77 9a   ^?S..k%.....Q.w.
    0550  1c 45 9a 0b 5b b8 1c 18 5d 11 cf 35 08 32 a5 68   .E..[...]..5.2.h
139 0560  91 b2 8c 10 a8 86 4f 46 d2 5c 2f 48 41 50 57 32   ......OF.\/HAPW2
    0570  3a 76 b4 84 ff 73 cf d6 2c 3a 3c 67 5f 64 12 b6   :v...s..,:<g_d..
141 0580  c4 2d 18 a1 c1 2f 22 44 a1 6a d9 8a 1b db ee 10   .-.../"D.j......
    0590  e7 10 09 66 21 1c 66 75 7a 00 73 63 c2 1d 65 34   ...f!.fuz.sc..e4
143 05a0  db be c3 46 b5 8f db b7 20 dc 19 22 ec 46 c7 7c   ...F.... ..".F.|
    05b0  8c b0 c2 6f 3f 06 ef c8 d9 99 f5 95 1c 59 9c 55   ...o?........Y.U
145 05c0  7d 2e 4e 87 2f 46 08 a8                         }.N./F..
        Data: a42f13ae060839361472baa5935b4c7b3df37df0ae953907...
147     [Length: 1480]
```

./traces/3–a.5–FRAG1.txt

Second packet:

```
  1 No.     Time           Source                Destination           Protocol Length
        Info
          1 0.000000000    10.0.1.11             10.0.2.22             IPv4     98
            Fragmented IP protocol (proto=UDP 17, off=1480, ID=34af)
  3
  Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0
  5     Interface id: 0 (eth0)
        Encapsulation type: Ethernet (1)
  7     Arrival Time: Mar 27, 2017 10:21:02.366466155 CEST
        [Time shift for this packet: 0.000000000 seconds]
  9     Epoch Time: 1490602862.366466155 seconds
        [Time delta from previous captured frame: 0.000000000 seconds]
 11     [Time delta from previous displayed frame: 0.000000000 seconds]
        [Time since reference or first frame: 0.000000000 seconds]
 13     Frame Number: 1
        Frame Length: 98 bytes (784 bits)
 15     Capture Length: 98 bytes (784 bits)
        [Frame is marked: False]
 17     [Frame is ignored: False]
        [Protocols in frame: eth:ethertype:ip:data]
 19 Ethernet II, Src: IntelCor_36:33:a0 (68:05:ca:36:33:a0), Dst: IntelCor_36:39:c7
        (68:05:ca:36:39:c7)
        Destination: IntelCor_36:39:c7 (68:05:ca:36:39:c7)
 21         Address: IntelCor_36:39:c7 (68:05:ca:36:39:c7)
            .... ..0. .... .... .... .... = LG bit: Globally unique address (factory
                default)
 23         .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
        Source: IntelCor_36:33:a0 (68:05:ca:36:33:a0)
 25         Address: IntelCor_36:33:a0 (68:05:ca:36:33:a0)
```

```
        .... ..0. .... .... .... .... = LG bit: Globally unique address (factory
            default)
        .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
    Type: IP (0x0800)
Internet Protocol Version 4, Src: 10.0.1.11 (10.0.1.11), Dst: 10.0.2.22 (10.0.2.22)
    Version: 4
    Header Length: 20 bytes
    Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (
        Not ECN-Capable Transport))
        0000 00.. = Differentiated Services Codepoint: Default (0x00)
        .... ..00 = Explicit Congestion Notification: Not-ECT (Not ECN-Capable
            Transport) (0x00)
    Total Length: 84
    Identification: 0x34af (13487)
    Flags: 0x00
        0... .... = Reserved bit: Not set
        .0.. .... = Don't fragment: Not set
        ..0. .... = More fragments: Not set
    Fragment offset: 1480
    Time to live: 64
    Protocol: UDP (17)
    Header checksum: 0x2e11 [validation disabled]
        [Good: False]
        [Bad: False]
    Source: 10.0.1.11 (10.0.1.11)
    Destination: 10.0.2.22 (10.0.2.22)
    [Source GeoIP: Unknown]
    [Destination GeoIP: Unknown]
Data (64 bytes)

0000  e3 c1 4c 1f 6f 83 61 dc fc 5e 5c f4 66 17 2c 73   ..L.o.a..^\.f.,s
0010  19 ce cc 2d 69 69 bd 43 33 6d 82 de 4a 87 8d 9d   ...-ii.C3m..J...
0020  81 f6 2e b2 8c 6a d5 e4 f2 e6 bc ab 5a 01 34 d5   .....j.......Z.4.
0030  39 f8 d9 5b 16 bc dc 95 bd b4 08 a9 5e 11 df 38   9..[........^..8
    Data: e3c14c1f6f8361dcfc5e5cf466172c7319cecc2d6969bd43...
    [Length: 64]
```

./traces/3–a.5–FRAG2.txt

As we can see in the IP header, there are flags which deal with fragmentation: the "don't fragment" flag and the "more fragments" flag, which in the first fragment is set to 1, as there will be a second fragment. The fragment offset for the first fragment is 0, because it is the first. For the second fragment it is 1480, as that is the byte where the previous fragment ended. In both packets the identification number is the same ($0x34af$) to indicate that it is the same packet, just fragmented.

Question 3.A.2)

Include the outcome of the experiment in Step 4. Indicate the UDP datagram size at which fragmentation occurs. Also, determine the maximum size of the UDP datagram that the system can send.

The UDP datagram size at which the fragmentation occurs is at 1472 bytes of data, with a UDP length of 1480.

Exercise 3-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). Both the TCP client and the TCP server send the MSS in an option that is attached to 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 only addresses MTU constraints at the hosts, but not at the intermediate routers. To determine the smallest MTU on the path from the sender to the receiver, TCP employs a method which is known as Path MTU Discovery, and 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 âĂIJDestination unreachable; Fragmentation neededâĂİ. Upon receiving such an ICMP error message, the TCP sender reduces the segment size. This continues until a segment size is determined which does not trigger an ICMP error message.

1. Modify the MTU of the interfaces with the values as shown in Table 5.2. In Linux, you can

Linux PC	MTU size of Ethernet Interface eth0	MTU size of Ethernet Interface eth1
PC1	1500	not used
PC2	500	not used
PC3	1500	1500

Table 5.2: MTU Sizes

view the MTU values of all interfaces in the output of 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 the *eth0* interfaces of both PC1 and PC3, and start to capture traffic with no filters set.

3. Start a nttcp receiver on PC2, and a nttcp sender on PC1 and generate TCP traffic with the following commands:

```
PC2% nttcp -i -rs -l1024 -n2 -p4444
PC1% nttcp -ts -l1024 -n2 -p4444 -D 10.0.2.22
```

Observe the output of Wireshark:

Question 3.B.1.3.a)
Do you observe fragmentation? If so, where does it occur? Explain your observation.
No IP fragmentation can be observed on the 10.0.1.0/24 network. This is because TCP negotiates maximum segment size based on MTU of the link. In this case the maximum segment size gets set to 460 bytes, because the MTU on the 10.0.2.0/24 for PC2 is 500 bytes and the connection runs across the 10.0.1.0/24 and 10.0.2.0/24 networks. The application data is segmented in TCP, and therefore no IP fragmentation occurs.

Question 3.B.1.3.b)
Explain why there is no ICMP error message generated in the first part of the experiment (Step 3). Is the DF bit set in the IP datagrams?
There is no ICMP error message because TCP first negotiated the segment size before sending any data. The don't fragment bit is set.

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

5. Repeat the nttcp transmission in Step 3.

Question 3.B.1.5.a)

Do you observe fragmentation? If so, where does it occur? Explain your observation.

Now TCP fragmentation occurs in the retransmissions because interface eth1 on PC3's MTU changed, and ICMP warns about this with a destination unreachable error (fragmentation needed).

Question 3.B.1.5.b)

If you observe ICMP error messages, describe how they are used for Path MTU Discovery.

ICMP error messages contain a field "MTU of next hop". These can be used to deduce the MTU for every link.

6. Save all Wireshark output (Select the Print details option).

Question 3.B.2)

If you observed ICMP error messages, include one such message in the report. Also include the first TCP segment that is sent after PC1 has received the ICMP error message.

```
No.     Time        Source                Destination           Protocol Length
    Info
  2     130 96.889700    10.0.1.33             10.0.1.11             ICMP     590
        Destination unreachable (Fragmentation needed)

  4 Frame 130: 590 bytes on wire (4720 bits), 590 bytes captured (4720 bits)
  Ethernet II, Src: IntelCor_36:39:c7 (68:05:ca:36:39:c7), Dst: IntelCor_36:33:a0
      (68:05:ca:36:33:a0)
  6 Internet Protocol Version 4, Src: 10.0.1.33 (10.0.1.33), Dst: 10.0.1.11 (10.0.1.11)
      Version: 4
  8   Header Length: 20 bytes
      Differentiated Services Field: 0xc0 (DSCP 0x30: Class Selector 6; ECN: 0x00:
          Not-ECT (Not ECN-Capable Transport))
 10   Total Length: 576
      Identification: 0x98c7 (39111)
 12   Flags: 0x00
      Fragment offset: 0
 14   Time to live: 64
      Protocol: ICMP (1)
 16   Header checksum: 0xc90a [validation disabled]
      Source: 10.0.1.33 (10.0.1.33)
 18   Destination: 10.0.1.11 (10.0.1.11)
      [Source GeoIP: Unknown]
 20   [Destination GeoIP: Unknown]
  Internet Control Message Protocol
 22   Type: 3 (Destination unreachable)
      Code: 4 (Fragmentation needed)
 24   Checksum: 0x164f [correct]
      MTU of next hop: 500
 26   Internet Protocol Version 4, Src: 10.0.1.11 (10.0.1.11), Dst: 10.0.2.22
          (10.0.2.22)
          Version: 4
 28       Header Length: 20 bytes
          Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT
              (Not ECN-Capable Transport))
 30       Total Length: 1076
          Identification: 0xa265 (41573)
```

```
32          Flags: 0x02 (Don't Fragment)
            Fragment offset: 0
34          Time to live: 64
            Protocol: TCP (6)
36          Header checksum: 0x7d3e [validation disabled]
            Source: 10.0.1.11 (10.0.1.11)
38          Destination: 10.0.2.22 (10.0.2.22)
            [Source GeoIP: Unknown]
40          [Destination GeoIP: Unknown]
        Transmission Control Protocol, Src Port: 54353 (54353), Dst Port: 5038 (5038),
            Seq: 3979089308, Ack: 1859776051
42          Source Port: 54353 (54353)
            Destination Port: 5038 (5038)
44          Sequence number: 3979089308
            [Stream index: 3]
46          Sequence number: 3979089308    (relative sequence number)
            Acknowledgment number: 1859776051    (relative ack number)
48          Header Length: 32 bytes
            .... 0000 0001 1001 = Flags: 0x019 (FIN, PSH, ACK)
50          Window size value: 229
            [Calculated window size: 229]
52          [Window size scaling factor: 128]
            Checksum: 0x9c6c [validation disabled]
54          Urgent pointer: 0
            Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
56              No-Operation (NOP)
                No-Operation (NOP)
58              Timestamps: TSval 1356793, TSecr 1356533
        Data (496 bytes)
60
    0000  88 af 34 68 f0 7f 00 00 88 af 34 68 f0 7f 00 00   ..4h......4h....
62  0010  40 12 9a 00 00 00 00 00 40 12 9a 00 00 00 00 00   @.......@.......
    0020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
64  0030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
    0040  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
66  0050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
    0060  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
68  0070  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
    0080  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
70  0090  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
    00a0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
72  00b0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
    00c0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
74  00d0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
    00e0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
76  00f0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
    0100  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
78  0110  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
    0120  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
80  0130  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
    0140  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
82  0150  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
    0160  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
84  0170  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
    0180  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
86  0190  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
    01a0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
88  01b0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
    01c0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
90  01d0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
    01e0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
92
    No.     Time        Source                Destination           Protocol Length
        Info
94      131 96.891108   10.0.1.11             10.0.2.22             TCP      566    [
            TCP Retransmission] 54353âĘŠ5038 [ACK] Seq=1 Ack=1 Win=29312 Len=500 TSval
            =1356794 TSecr=1356533
```

```
 96 Frame 131: 566 bytes on wire (4528 bits), 566 bytes captured (4528 bits)
    Ethernet II, Src: IntelCor_36:33:a0 (68:05:ca:36:33:a0), Dst: IntelCor_36:39:c7
        (68:05:ca:36:39:c7)
 98 Internet Protocol Version 4, Src: 10.0.1.11 (10.0.1.11), Dst: 10.0.2.22 (10.0.2.22)
        Version: 4
100     Header Length: 20 bytes
        Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (
            Not ECN-Capable Transport))
102     Total Length: 552
        Identification: 0xa266 (41574)
104     Flags: 0x02 (Don't Fragment)
        Fragment offset: 0
106     Time to live: 64
        Protocol: TCP (6)
108     Header checksum: 0x7f49 [validation disabled]
        Source: 10.0.1.11 (10.0.1.11)
110     Destination: 10.0.2.22 (10.0.2.22)
        [Source GeoIP: Unknown]
112     [Destination GeoIP: Unknown]
    Transmission Control Protocol, Src Port: 54353 (54353), Dst Port: 5038 (5038), Seq:
         1, Ack: 1, Len: 500
114     Source Port: 54353 (54353)
        Destination Port: 5038 (5038)
116     [Stream index: 3]
        [TCP Segment Len: 500]
118     Sequence number: 1    (relative sequence number)
        [Next sequence number: 501    (relative sequence number)]
120     Acknowledgment number: 1    (relative ack number)
        Header Length: 32 bytes
122     .... 0000 0001 0000 = Flags: 0x010 (ACK)
        Window size value: 229
124     [Calculated window size: 29312]
        [Window size scaling factor: 128]
126     Checksum: 0xa280 [validation disabled]
        Urgent pointer: 0
128     Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
            No-Operation (NOP)
130         No-Operation (NOP)
            Timestamps: TSval 1356794, TSecr 1356533
132     [SEQ/ACK analysis]
    Data (500 bytes)
134
    0000  88 af 34 68 f0 7f 00 00 88 af 34 68 f0 7f 00 00   ..4h......4h....
136 0010  40 12 9a 00 00 00 00 00 40 12 9a 00 00 00 00 00   @.......@.......
    0020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
138 0030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
    0040  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
140 0050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
    0060  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
142 0070  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
    0080  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
144 0090  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
    00a0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
146 00b0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
    00c0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
148 00d0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
    00e0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
150 00f0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
    0100  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
152 0110  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
    0120  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
154 0130  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
    0140  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
156 0150  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
    0160  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
158 0170  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
```

```
     0180  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
 160 0190  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
     01a0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
 162 01b0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
     01c0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
 164 01d0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
     01e0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
 166 01f0  00 00 00 00                                      ....
```

./traces/3.B.2.txt

Part 4. 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 is closed separately, and each part of the closing is called a half-close.

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 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.

Answer the questions in Exercise 4-a (Steps 3, 4, and 5), Exercise 4-b (Step 3), and Exercise 4-c (Step 2). **For each answer, include Wireshark data to support your answer.**

Exercise 4-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.s

1. This part of the lab only uses PC1 and PC2 in the network configuration in Figure 5.1. If the network is not set up, follow the instructions of Exercise 1-a. Verify that the MTU values of all interfaces of PC1 and PC2 are set to 1500 bytes, which is the default MTU for Ethernet networks.
2. Start Wireshark on the *eth1* interface of PC1 to capture traffic of the Telnet connection. Do not set any filters.
3. Establishing a TCP connection: Establish a Telnet session from PC1 to PC2 as follows:

   ```
   PC1% telnet 10.0.5.22
   ```

   Observe the TCP segments of the packets that are transmitted:

Question 4.A.3.a)
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 or TCP client.

```
No.     Time           Source                Destination           Protocol Length
Info
1 0.000000000    10.0.5.11             10.0.5.22             TCP      74     33230âĘŠ23
[SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=1525233 TSecr=0 WS=128
```

```
Frame 1: 74 bytes on wire (592 bits),74 bytes captured (592 bits) on interface
0
...
Flags: 0x002 (SYN)
000. .... .... = Reserved: Not set
...0 .... .... = Nonce: Not set
.... 0... .... = Congestion Window Reduced (CWR): Not set
.... .0.. .... = ECN-Echo: Not set
.... ..0. .... = Urgent: Not set
.... ...0 .... = Acknowledgment: Not set
.... .... 0... = Push: Not set
.... .... .0.. = Reset: Not set
.... .... ..1. = Syn: Set
[Expert Info (Chat/Sequence): Connection establish request (SYN): server port
23]
[Connection establish request (SYN): server port 23]
[Severity level: Chat]
[Group: Sequence]
.... .... ...0 = Fin: Not set
[TCP Flags: ÂüÂüÂüÂüÂüÂüÂüÂüÂüSÂü]
...

No.     Time           Source                Destination           Protocol Length
Info
2 0.000388357    10.0.5.22             10.0.5.11             TCP      74     23âĘŠ33230
[SYN,ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=1524972 TSecr=1525233
WS=128

Frame 2: 74 bytes on wire (592 bits),74 bytes captured (592 bits) on interface
0
...
Flags: 0x012 (SYN,ACK)
000. .... .... = Reserved: Not set
...0 .... .... = Nonce: Not set
.... 0... .... = Congestion Window Reduced (CWR): Not set
.... .0.. .... = ECN-Echo: Not set
.... ..0. .... = Urgent: Not set
.... ...1 .... = Acknowledgment: Set
.... .... 0... = Push: Not set
.... .... .0.. = Reset: Not set
.... .... ..1. = Syn: Set
[Expert Info (Chat/Sequence): Connection establish acknowledge (SYN+ACK): server
port 23]
[Connection establish acknowledge (SYN+ACK): server port 23]
[Severity level: Chat]
[Group: Sequence]
.... .... ...0 = Fin: Not set
[TCP Flags: ÂüÂüÂüÂüÂüÂüÂüAÂüÂüSÂü]
...

No.     Time           Source                Destination           Protocol Length
Info
3 0.000417125    10.0.5.11             10.0.5.22             TCP      66     33230âĘŠ23
```

```
[ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=1525233 TSecr=1524972

Frame 3: 66 bytes on wire (528 bits),66 bytes captured (528 bits) on interface
0
...
Flags: 0x010 (ACK)
000. .... .... = Reserved: Not set
...0 .... .... = Nonce: Not set
.... 0... .... = Congestion Window Reduced (CWR): Not set
.... .0.. .... = ECN-Echo: Not set
.... ..0. .... = Urgent: Not set
.... ...1 .... = Acknowledgment: Set
.... .... 0... = Push: Not set
.... .... .0.. = Reset: Not set
.... .... ..0. = Syn: Not set
.... .... ...0 = Fin: Not set
[TCP Flags: ÂüÂüÂüÂüÂüÂüAÂüÂüÂüÂü]
...
```

As you can see, frame 1 has the SYN flag set, frame 2 has the SYN flag and the ACK flag and frame 3 only has the ACK flag.
What happens is the following: the client starts the conversation, sending a SYN message. The server then acknowledges this SYN message by responding with a SYN-ACK message. Upon receiving the response, the client sends an ACK.

Question 4.A.3.b)

During the connection setup, the TCP client and TCP server tell each other the first sequence number they will use for data transmission. What is the initial sequence number of the TCP client and the TCP server?

```
No.     Time           Source                Destination           Protocol Length
Info
1 0.000000000    10.0.5.11             10.0.5.22             TCP      74     33230âĘŠ23
[SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=1525233 TSecr=0 WS=128

Frame 1: 74 bytes on wire (592 bits),74 bytes captured (592 bits) on interface
0
...
Transmission Control Protocol,Src Port: 33230,Dst Port: 23,Seq: 0,Len: 0
Source Port: 33230
Destination Port: 23
[Stream index: 0]
[TCP Segment Len: 0]
Sequence number: 0    (relative sequence number)
Acknowledgment number: 0
Header Length: 40 bytes
...

No.     Time           Source                Destination           Protocol Length
Info
2 0.000388357    10.0.5.22             10.0.5.11             TCP      74     23âĘŠ33230
[SYN,ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=1524972 TSecr=1525233
WS=128
```

```
Frame 2: 74 bytes on wire (592 bits),74 bytes captured (592 bits) on interface
0
...
Transmission Control Protocol,Src Port: 23,Dst Port: 33230,Seq: 0,Ack: 1,Len:
0
Source Port: 23
Destination Port: 33230
[Stream index: 0]
[TCP Segment Len: 0]
Sequence number: 0    (relative sequence number)
...

No.     Time           Source                Destination           Protocol Length
Info
3 0.000417125    10.0.5.11             10.0.5.22             TCP      66     33230âĘŠ23
[ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=1525233 TSecr=1524972

Frame 3: 66 bytes on wire (528 bits),66 bytes captured (528 bits) on interface
0
...
Transmission Control Protocol,Src Port: 33230,Dst Port: 23,Seq: 1,Ack: 1,Len:
0
Source Port: 33230
Destination Port: 23
[Stream index: 0]
[TCP Segment Len: 0]
Sequence number: 1    (relative sequence number)
...
```

As you can see, the sequence number starts at 0.

Question 4.A.3.c)

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?

```
No.     Time           Source                Destination           Protocol Length
Info
4 0.000556972    10.0.5.11             10.0.5.22             TELNET   93     Telnet
Data ...

Frame 4: 93 bytes on wire (744 bits),93 bytes captured (744 bits) on interface
0
...
Sequence number: 1    (relative sequence number)
[Next sequence number: 28    (relative sequence number)]
Acknowledgment number: 1    (relative ack number)
...
```

As you can see, the first sequence number that is used is 1.

Question 4.A.3.d)

The TCP client and TCP server exchange window sizes to get the maximum amount of data

that the other side can sent at any time. Determine the values of the window sizes for the TCP client and the TCP server.

```
No.     Time           Source                Destination           Protocol Length
Info
2 0.000388357    10.0.5.22             10.0.5.11             TCP      74     23âĘŠ33230
[SYN,ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=1524972 TSecr=1525233
WS=128

Frame 2: 74 bytes on wire (592 bits),74 bytes captured (592 bits) on interface
0
...
Window size value: 28960
[Calculated window size: 28960]
...

No.     Time           Source                Destination           Protocol Length
Info
3 0.000417125    10.0.5.11             10.0.5.22             TCP      66     33230âĘŠ23
[ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=1525233 TSecr=1524972

Frame 3: 66 bytes on wire (528 bits),66 bytes captured (528 bits) on interface
0
...
Window size value: 229
[Calculated window size: 29312]
...
```

The client has a window size of 29312, the server has a window size of 28960. **4.A.3.e**)

What is the MSS value that is negotiated between the TCP client and the TCP server? **4.A.3.f**)

```
No.     Time           Source                Destination
Protocol Length Info
1 0.000000000    10.0.5.11             10.0.5.22             TCP
74     33230âĘŠ23 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1
TSval=1525233 TSecr=0 WS=128

No.     Time           Source                Destination
Protocol Length Info
2 0.000388357    10.0.5.22             10.0.5.11             TCP
74     23âĘŠ33230 [SYN,ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460
SACK_PERM=1 TSval=1524972 TSecr=1525233 WS=128
```

The negotiated MSS value is 1460.

How long does it take to open a TCP connection?

```
No.     Time           Source                Destination
Protocol Length Info
1 0.000000000    10.0.5.11             10.0.5.22             TCP
74     33230âĘŠ23 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1
TSval=1525233 TSecr=0 WS=128

No.     Time           Source                Destination
Protocol Length Info
4 0.000556972    10.0.5.11             10.0.5.22             TELNET
93     Telnet Data ...
```

The first application data is sent after 0.556972 ms. This means that the TCP connection setup takes 0.556972 ms.

4. Closing a TCP connection (initiated by client): On PC1, type `Ctrl-]` at the Telnet prompt and type quit, to terminate the connection. (If the Telnet session is no longer running, first create a new session). In the output of Wireshark, observe the TCP segments of the packets that are transmitted:

Question 4.A.4)

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.

The packets involved in closing the TCP connection are packets with both the FIN and ACK flags set, and a last ACK (packet 84 - 86 in 4-a.6.pcapng) The ACK flags acknowledge the receiving of previous packets, where the FIN flags denote the end of the TCP connection. The client's FIN flag gets replied to by the server's ACK with the FIN flag set, denoting the connection on the server's end is closing too. This then gets acknowledged by the client sending an ACK to the server.

5. , Closing a TCP connection (initiated by server): The closing of a connection can also be initiated by the server application, as seen next. Establish a Telnet session on PC1 to PC2 as follows:

```
PC1% telnet 10.0.5.22
```

Do not type anything. After a while, the connection will be closed by the TCP server, and a message is displayed at the Telnet client application.

Question 4.A.5.a)

Describe how the closing of the connection is different from Step 4.

The telnet server sends the client a message containing "Login timed out after 60 seconds.", which gets acknowledged. Afterwards the server sends a packet to the client containing the FIN and ACK flags, which gets responded to by the client with FIN and ACK, which then gets acknowledged by the server. It is now the server initiating the closing of the connection instead of the client.

Question 4.A.5.b)

How long does the Telnet server wait until it closes the TCP connection?

The server closes the connection after 60 seconds of inactivity.

6. Save the Wireshark output.

Exercise 4-b. Requesting a connection to non-existing host

Here you observe how often a TCP client tries to establish a connection to a host that does not exist, before it gives up.

1. Start a new traffic capture with Wireshark on interface *eth1* of PC1.
2. Set a static entry in the ARP table for the non-existing IP address 10.0.5.100. Note that the IP address does not exist.

   ```
   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
   ```

   Observe the TCP segments that are transmitted.

Question 4.B.3.a)
How often does the TCP client try to establish a connection? How much time elapses between repeated attempts to open a connection?
TCP tries to establish a connection a total of 6 times, with exponential backoff in time. The first waiting period is 1 second, then 2, then 4, 8, 16 and 32.

Question 4.B.3.b)
Does the TCP client terminate or reset the connection, when it gives up with trying to establish a connection?
The TCP client doesn't terminate a connection, as there was no connection that was set up. It just gives up its retransmissions.

Question 4.B.3.c)
Why does this experiment require to set a static ARP table entry?
So TCP thinks there is an actual host with the address 10.0.5.100. If it was up to dynamic routing protocols, if there is no (partial) match for that address, there is no such host and it would not even have to try to establish a connection to realize there is no such host.

4. Save the Wireshark output.

Exercise 4-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 interface *eth1* of PC1.
2. Establish a TCP connection to port 80 of PC2.

   ```
   PC1% telnet 10.0.5.22 80
   ```

There should not be a TCP server running on PC2 that is listening at this port number. Observe the TCP segments of the packets that are transmitted:

Question 4.C.2)

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

The remote host closes the connection by setting the RST (reset) flag of its acknowledgement. The connection therefore ends immediately when the acknowledgement is received.

3. Save the Wireshark output.

Part 5. TCP data exchange - Interactive applications

In Parts 5 and 6 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 acknowledgements 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 acknowledgement number is by one larger 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 acknowledgements.

In this lab, you study acknowledgements 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, TCP has mechanisms that keep the number of segments with a small payload small. One such mechanism, called delayed acknowledgements, requires that the receiver of data waits 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 NagleâĂŹs algorithm, limits the number of small segments that a TCP sender can transmit without waiting for an ACK.

The network configuration is shown in Figure 5.2. The network connects two Linux PCs, PC1 and PC2, such that there are three paths between the PCs. One route goes over three Ethernet links (with either 10 Mbps or 100 Mbps), and one route goes over a serial WAN link (which will be set to 125 kbps), and one route goes over a direct Ethernet link (also with 10 Mbps or 100 Mbps).

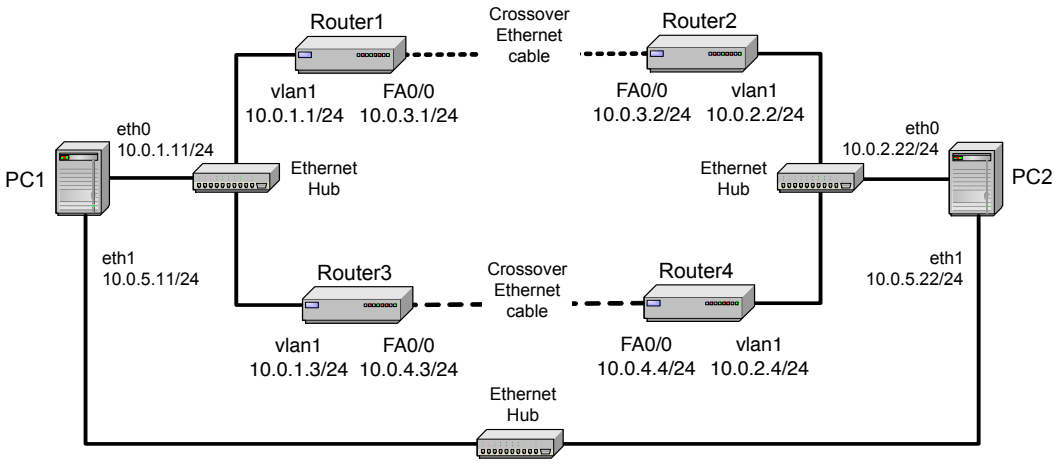


Figure 5.2: Network Topology for Parts 1-4.

Linux PC	Interface eth0	Interface eth1
PC1	10.0.1.11/24	10.0.5.11/24
PC2	10.0.2.22/24	10.0.5.22/24
Cisco Router	**Interface FastEthernet0/0**	**Interface vlan1**
Router1	10.0.3.1/24	10.0.1.1/24
Router2	10.0.3.2/24	10.0.2.2/24
Cisco Router	**Interface FastEthernet0/0**	**Interface vlan1**
Router3	10.0.4.3/24	10.0.1.3/24
Router4	10.0.4.4/24	10.0.2.4/24

Table 5.3: IP Addresses of Linux PCs and Cisco routers.

Linux PC	Routing Table Entries	
	Destination	**Next Hop**
PC1	default gateway	10.0.1.3
PC2	default gateway	10.0.2.4
Linux PC	**Routing Table Entries**	
	Destination	**Next Hop**
Router1	network 10.0.4.0/24	10.0.1.3
Router1	default gateway	10.0.3.2
Router2	network 10.0.4.0/24	10.0.2.4
Router2	default gateway	10.0.3.1
Router3	network 10.0.3.0/24	10.0.1.1
Router3	default gateway	10.0.4.4
Router4	network 10.0.3.0/24	10.0.2.2
Router4	default gateway	10.0.4.3

Table 5.4: Routing table entries for Part 5-8.

Exercise 5-a. Network Setup

The following network configuration is used in the remaining parts of the lab.

1. Set up the Ethernet connections as shown in Figure 5.2. *Note: connect two routers through a hub or switch if you don't have a crossover cable available.*

2. Configure the IP addresses and the routing tables of the PCs as shown in Tables 5.3 and 5.4.

   - IP forwarding should be disabled on both PC1 and PC2.
   - Set a new default gateways on PC1 and PC2. Remove the default gateway entry that was set in Part 1 of the lab. Changing the default gateway on PC1 is done with the following commands:

     ```
     PC1% route del default gw 10.0.1.33
     PC1% route add default gw 10.0.1.3
     ```

   Repeat the steps on PC2. Use the command `netstat -rn`, to verify that there are no other static routing entries.

3. Verify that the PCs are connected to the console ports of routers. PC1 should be connected to Router1, PC2 to Router 2, and so on. On each PC, establish a `minicom` session to the connected router.

4. Configure the IP addresses and routing table entries of the routers. The commands for Router4 are as follows:

```
Router4> enable
Password: <enable secret>
Router4# configure terminal
Router4(config)# no ip routing
Router4(config)# ip routing
Router4(config)# ip route 0.0.0.0 0.0.0.0 10.0.4.3
Router4(config)# ip route 10.0.3.0 255.255.255.0 10.0.2.2
Router4(config)# interface FastEthernet0/0
Router4(config-if)# no shutdown
Router4(config-if)# ip address 10.0.2.4 255.255.255.0
Router4(config-if)# interface FastEthernet0/1
Router4(config-if)# no shutdown
Router4(config-if)# interface vlan1
Router4(config-if)# no shutdown
Router4(config-if)# ip address 10.0.4.4 255.255.255.0
Router4(config-if)# end
```

   Use the commands `show ip route` and show interfaces to verify that the routing table and the interfaces are set correctly.

5. Emulate a slow serial link between Router3 and Router4 using traffic-shaping:

```
> interface FastEthernet0/0
> ip address 10.10.1.80 255.255.255.0
> fair-queue
> traffic-shape rate 64000 5000 5000 1000
```

   This will simulate a link of 64kbps on the packets going out of *FE0/0*. You can change the parameters of the “traffic-shape” command to control the rate of the link. DonâĂŹt forget that shaping only works on the outgoing packets of the interface, so you will have to configure this in both Router3 and Router4! Also note that you can not apply the traffic shaping to a virtual interface like *Vlan1*. You can use

```
> show running-config
```

   to check if the traffic shaping is correctly configured to 64kbps.

6. Test the network connectivity by issuing ping commands between PC1 and PC2. Verify the route taken by traffic between the PCs by issuing `traceroute` commands.

```
PC1% traceroute 10.0.2.22 PC1% traceroute 10.0.5.22
PC2% traceroute 10.0.1.11 PC2% traceroute 10.0.5.11
```

   Also, you should be able to issue ping commands between the routers. If the commands are not successful, use the commands `traceroute` (on Linux) or `trace` (on IOS) and the content of the routing tables to locate configuration problems.

If all commands are successful, then you are ready to continue.

Exercise 5-b. TCP Data Transfer - Interactive Applications over a fast link

Here you observe interactive data transfer in TCP, by establishing a TCP connection from PC1 to PC2 over the Ethernet link between the PCs. Dependent 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 *eth1*, and start to capture traffic. Do not set any filters.
2. On PC1, establish a Telnet session to PC2 by typing:

   ```
   PC1% telnet 10.0.5.22
   ```

   Log in as root user.
3. Now start to type a few characters in the window which contains the Telnet session. The Telnet client sends each typed character in a separate TCP segment to the Telnet server, which, in turn, echoes the character 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. Observe the output of Wireshark:

Include your answers to the following questions. Include examples from the saved Wireshark data to support your answers.

Question 5.B.1.a)

Observe the number of packets exchanged between the Linux PCs for each keystroke? Describe the payload of the packets. Use your knowledge of delayed acknowledgements to explain the sequence of segment transmissions. Explain why you do not see four packets per typed character.

```
22 10.891039493   10.0.5.11             10.0.5.22             TELNET   67     Telnet
Data ...

Frame 22: 67 bytes on wire (536 bits),67 bytes captured (536 bits) on interface
0
Ethernet II,Src: 68:05:ca:39:cc:79,Dst: 68:05:ca:39:e1:36
Internet Protocol Version 4,Src: 10.0.5.11,Dst: 10.0.5.22
Transmission Control Protocol,Src Port: 33240 (33240),Dst Port: 23 (23),Seq: 115,Ack:
91,Len: 1
Telnet

23 10.891487340   10.0.5.22             10.0.5.11             TELNET   67     Telnet
Data ...

Frame 23: 67 bytes on wire (536 bits),67 bytes captured (536 bits) on interface
0
Ethernet II,Src: 68:05:ca:39:e1:36,Dst: 68:05:ca:39:cc:79
Internet Protocol Version 4,Src: 10.0.5.22,Dst: 10.0.5.11
Transmission Control Protocol,Src Port: 23 (23),Dst Port: 33240 (33240),Seq: 91,Ack:
116,Len: 1
Telnet

24 10.891507546   10.0.5.11             10.0.5.22             TCP      66     33240
âĘŠ 23 [ACK] Seq=116 Ack=92 Win=29312 Len=0 TSval=2012244 TSecr=2011981

Frame 24: 66 bytes on wire (528 bits),66 bytes captured (528 bits) on interface
0
Ethernet II,Src: 68:05:ca:39:cc:79,Dst: 68:05:ca:39:e1:36
Internet Protocol Version 4,Src: 10.0.5.11,Dst: 10.0.5.22
Transmission Control Protocol,Src Port: 33240 (33240),Dst Port: 23 (23),Seq: 116,Ack:
92,Len: 0
```

```
25 11.026831823   10.0.5.11            10.0.5.22            TELNET   67     Telnet
Data ...

Frame 25: 67 bytes on wire (536 bits),67 bytes captured (536 bits) on interface
0
Ethernet II,Src: 68:05:ca:39:cc:79,Dst: 68:05:ca:39:e1:36
Internet Protocol Version 4,Src: 10.0.5.11,Dst: 10.0.5.22
Transmission Control Protocol,Src Port: 33240 (33240),Dst Port: 23 (23),Seq: 116,Ack:
92,Len: 1
Telnet
```

For the username, there are 3 packets per typed character. An example of this are packets 22-24, which indicate that the "s" key has been pressed. PC1 sends a character to PC2. According to the protocol, PC2 has to send the same character back to PC1. Now according to Nagle's algoritmh, PC2 piggybacks this character on the ACK he had to send to PC1. After that, PC1 sends an ACK back to PC2. You might wonder why PC1 doesn't piggyback his next character on that ACK. But that is because the time between keystrokes was too large. As you can see in packets 22 and 25, the time between keystrokes was approximately 135 ms.

For the password, there's only two packets per keystroke. The reason for this is the same as for the username, except now PC2 doesn't have to send the characters back to PC1 (according to TELNET) and thus PC2 can't piggyback acknowledgments.

For the exit command, the mechanism is the same as for the username.

Question 5.B.1.b)

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?

```
48 13.442543645   10.0.5.11            10.0.5.22            TELNET   67     Telnet
Data ...

Frame 48: 67 bytes on wire (536 bits),67 bytes captured (536 bits) on interface
0
Ethernet II,Src: 68:05:ca:39:cc:79,Dst: 68:05:ca:39:e1:36
Internet Protocol Version 4,Src: 10.0.5.11,Dst: 10.0.5.22
Transmission Control Protocol,Src Port: 33240 (33240),Dst Port: 23 (23),Seq: 124,Ack:
110,Len: 1
Telnet

49 13.480247436   10.0.5.22            10.0.5.11            TCP      66     23
âĘŠ 33240 [ACK] Seq=110 Ack=125 Win=29056 Len=0 TSval=2012629 TSecr=2012882

Frame 49: 66 bytes on wire (528 bits),66 bytes captured (528 bits) on interface
0
Ethernet II,Src: 68:05:ca:39:e1:36,Dst: 68:05:ca:39:cc:79
Internet Protocol Version 4,Src: 10.0.5.22,Dst: 10.0.5.11
Transmission Control Protocol,Src Port: 23 (23),Dst Port: 33240 (33240),Seq: 110,Ack:
125,Len: 0
```

```
60 15.547661297   10.0.5.11            10.0.5.22            TELNET   67     Telnet
Data ...

Frame 60: 67 bytes on wire (536 bits),67 bytes captured (536 bits) on interface
0
Ethernet II,Src: 68:05:ca:39:cc:79,Dst: 68:05:ca:39:e1:36
Internet Protocol Version 4,Src: 10.0.5.11,Dst: 10.0.5.22
Transmission Control Protocol,Src Port: 33240 (33240),Dst Port: 23 (23),Seq: 130,Ack:
110,Len: 1
Telnet

61 15.547992610   10.0.5.22            10.0.5.11            TCP      66     23
âĘŠ 33240 [ACK] Seq=110 Ack=131 Win=29056 Len=0 TSval=2013145 TSecr=2013408

Frame 61: 66 bytes on wire (528 bits),66 bytes captured (528 bits) on interface
0
Ethernet II,Src: 68:05:ca:39:e1:36,Dst: 68:05:ca:39:cc:79
Internet Protocol Version 4,Src: 10.0.5.22,Dst: 10.0.5.11
Transmission Control Protocol,Src Port: 23 (23),Dst Port: 33240 (33240),Seq: 110,Ack:
131,Len: 0
```

TCP delays ACKs so that they may be sent together with other ACKs or so that they can be piggybacked. This reduces the overhead of TCP.
Because the ACKs get piggybacked here, we can't calculate the maximum delay, but we can get a lower and upper bound for the delays. According to RFC 1122, this delay may be at most 500 ms.
The delay time ranges (in this example) from 0.33 ms (as seen in packets 60 and 61) to 42 ms (as seen in packets 48 and 49). As we can see in the other packets, the delays are rather close to 0.33 ms and rather far from 42 ms.

Question 5.B.1.c)

What is the time delay associated with the transmission of ACKs from the Telnet server on PC3?

Question 5.B.1.d)

Which flags, if any, are set in the TCP segments that carry typed characters as payload? Explain the meaning of these flags.

```
No.     Time           Source                Destination           Protocol Length
Info
9 3.896053141    10.0.1.11            10.0.2.22            TELNET   93     Telnet
Data ...

Frame 9: 93 bytes on wire (744 bits),93 bytes captured (744 bits) on interface
0
Interface id: 0 (eth0)
Encapsulation type: Ethernet (1)
Arrival Time: Mar 27,2017 11:28:35.552353333 CEST
[Time shift for this packet: 0.000000000 seconds]
Epoch Time: 1490606915.552353333 seconds
[Time delta from previous captured frame: 0.000124951 seconds]
[Time delta from previous displayed frame: 0.000124951 seconds]
[Time since reference or first frame: 3.896053141 seconds]
```

```
Frame Number: 9
Frame Length: 93 bytes (744 bits)
Capture Length: 93 bytes (744 bits)
[Frame is marked: False]
[Frame is ignored: False]
[Protocols in frame: eth:ethertype:ip:tcp:telnet]
[Coloring Rule Name: TCP]
[Coloring Rule String: tcp]
Ethernet II,Src: IntelCor_36:33:a0 (68:05:ca:36:33:a0),Dst: CiscoInc_ef:eb:24
(00:0d:bc:ef:eb:24)
Destination: CiscoInc_ef:eb:24 (00:0d:bc:ef:eb:24)
Address: CiscoInc_ef:eb:24 (00:0d:bc:ef:eb:24)
.... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
.... ...0 .... .... .... .... = IG bit: Individual address (unicast)
Source: IntelCor_36:33:a0 (68:05:ca:36:33:a0)
Address: IntelCor_36:33:a0 (68:05:ca:36:33:a0)
.... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
.... ...0 .... .... .... .... = IG bit: Individual address (unicast)
Type: IPv4 (0x0800)
Internet Protocol Version 4,Src: 10.0.1.11,Dst: 10.0.2.22
0100 .... = Version: 4
.... 0101 = Header Length: 20 bytes (5)
Differentiated Services Field: 0x10 (DSCP: Unknown,ECN: Not-ECT)
0001 00.. = Differentiated Services Codepoint: Unknown (4)
.... ..00 = Explicit Congestion Notification: Not ECN-Capable Transport (0)
Total Length: 79
Identification: 0x99b1 (39345)
Flags: 0x02 (Don't Fragment)
0... .... = Reserved bit: Not set
.1.. .... = Don't fragment: Set
..0. .... = More fragments: Not set
Fragment offset: 0
Time to live: 64
Protocol: TCP (6)
Header checksum: 0x89c7 [validation disabled]
[Header checksum status: Unverified]
Source: 10.0.1.11
Destination: 10.0.2.22
[Source GeoIP: Unknown]
[Destination GeoIP: Unknown]
Transmission Control Protocol,Src Port: 45030,Dst Port: 23,Seq: 1,Ack: 1,Len:
27
Source Port: 45030
Destination Port: 23
[Stream index: 0]
[TCP Segment Len: 27]
Sequence number: 1    (relative sequence number)
[Next sequence number: 28    (relative sequence number)]
Acknowledgment number: 1    (relative ack number)
Header Length: 32 bytes
Flags: 0x018 (PSH,ACK)
000. .... .... = Reserved: Not set
...0 .... .... = Nonce: Not set
.... 0... .... = Congestion Window Reduced (CWR): Not set
```

```
.... .0.. .... = ECN-Echo: Not set
.... ..0. .... = Urgent: Not set
.... ...1 .... = Acknowledgment: Set
.... .... 1... = Push: Set
.... .... .0.. = Reset: Not set
.... .... ..0. = Syn: Not set
.... .... ...0 = Fin: Not set
[TCP Flags: ÂůÂůÂůÂůÂůÂůÂůAPÂůÂůÂů]
...
```

Acknowledgment flag: This flag indicates that the Acknowledgment field is significant. All packets after the first packet of the session sent should have this flag set. Push flag: This flag indicates that TCP must send the packets immediately instead of waiting and combining the packets later.

Question 5.B.1.e)

Why do segments that have an empty payload carry a sequence number? Why does this not result in confusion at the TCP receiver?

Question 5.B.1.f)

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?

```
No.     Time           Source                Destination           Protocol Length
Info
1 0.000000000    10.0.5.11             10.0.5.22             TCP      74     33240âĘŠ23
[SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=2009521 TSecr=0 WS=128

Frame 1: 74 bytes on wire (592 bits),74 bytes captured (592 bits) on interface
0
...
Window size value: 29200
[Calculated window size: 29200]
...

No.     Time           Source                Destination           Protocol Length
Info
2 0.000392779    10.0.5.22             10.0.5.11             TCP      74     23âĘŠ33240
[SYN,ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=2009259 TSecr=2009521
WS=128

Frame 2: 74 bytes on wire (592 bits),74 bytes captured (592 bits) on interface
0
...
Window size value: 28960
[Calculated window size: 28960]
...

No.     Time           Source                Destination           Protocol Length
Info
69 16.150448897   10.0.5.11             10.0.5.22             TCP      66     33240âĘŠ23
[ACK] Seq=133 Ack=707 Win=30336 Len=0 TSval=2013559 TSecr=2013296
```

```
Frame 69: 66 bytes on wire (528 bits),66 bytes captured (528 bits) on interface
0
...
Window size value: 237
[Calculated window size: 30336]
...
```

The initial window size at the client is 29200. After the first packet, it becomes 29312.
The initial window size at the server is 28960. After the first packet, it becomes 29056.
Starting from packet 69, the window size of the client changes, it is now 30336.

Question 5.B.1.g)

Type characters in the Telnet client program as fast as you can, e.g., by pressing a key and holding it down. Do you observe a difference in the transmission of segment payloads and ACKs?

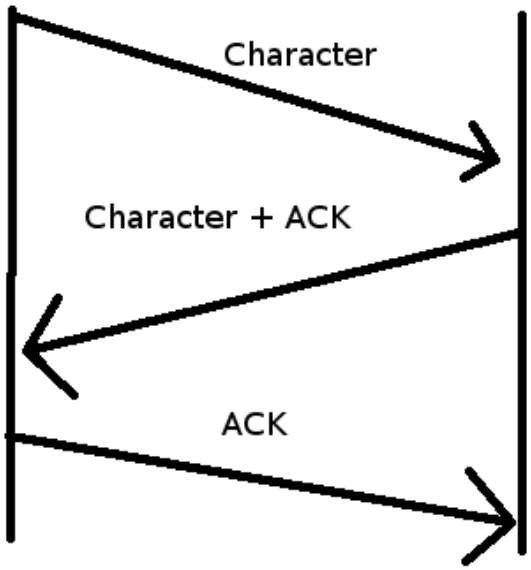
We usually don't pay attention to the exercises during the lab. This exercise requires additional actions on the lab computers. Because we noticed this too late, we couldn't make this exercise.

4. Terminate the Telnet session by typing exit.

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

Question 5.B.2)

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 5-c. TCP Data Transfer - Interactive Applications over a slow link

This exercise repeats the previous exercise, but establishes a data connection over the emulated slow link. The rate of this link should be set to 9600bps. This low rate introduces significant delays between PC1 and PC2. Due to the long delay, one would expect that the TCP sender transmits multiple segments, each carrying a payload of one typed character. However, this is not the case. A heuristic in TCP, called NagleâĂŹs algorithm, forces the sender to wait for an ACK after transmitting a small segment, even if the window size would allow the transmission of multiple segments. Therefore, no matter how slow or fast you type, you should only observe one TCP segment in transmission at a time, when the TCP segments are small.

1. Start to capture traffic with Wireshark on interface *eth0* of PC1. Do not set any display filters.
2. On PC1, establish a Telnet session to PC2 by typing:

   ```
   PC1% telnet 10.0.2.22
   ```

   Log in as root user. Note With the above IP address the route between PC1 to PC2 passes through the emulated serial link between Router3 and Router4.
3. As, in the previous exercise, type a few characters in the window that contains the Telnet session. Vary the rate at which you type characters in the Telnet client program. Observe the output of Wireshark:

Include your answers to the following questions. For each answer, include Wireshark data to support your answer.

Question 5.C.1.a)

Observe the number of packets that are exchanged between the Linux PCs for each keystroke? Observe how the transmission of packets changes when you type characters more quickly.

```
153 51.154268129   10.0.1.11            10.0.2.22            TELNET   67     Telnet
Data ...

Frame 153: 67 bytes on wire (536 bits),67 bytes captured (536 bits) on interface
0
Ethernet II,Src: 68:05:ca:36:33:a0,Dst: 00:0d:bc:ef:eb:24
Internet Protocol Version 4,Src: 10.0.1.11,Dst: 10.0.2.22
Transmission Control Protocol,Src Port: 45030 (45030),Dst Port: 23 (23),Seq: 143,Ack:
790,Len: 1
Telnet

154 51.155874871   10.0.2.22            10.0.1.11            TELNET   67     Telnet
Data ...

Frame 154: 67 bytes on wire (536 bits),67 bytes captured (536 bits) on interface
0
Ethernet II,Src: 00:0d:bc:ef:eb:24,Dst: 68:05:ca:36:33:a0
Internet Protocol Version 4,Src: 10.0.2.22,Dst: 10.0.1.11
```

```
Transmission Control Protocol,Src Port: 23 (23),Dst Port: 45030 (45030),Seq: 790,Ack:
144,Len: 1
Telnet

155 51.155910736   10.0.1.11            10.0.2.22            TELNET   67     Telnet
Data ...

Frame 155: 67 bytes on wire (536 bits),67 bytes captured (536 bits) on interface
0
Ethernet II,Src: 68:05:ca:36:33:a0,Dst: 00:0d:bc:ef:eb:24
Internet Protocol Version 4,Src: 10.0.1.11,Dst: 10.0.2.22
Transmission Control Protocol,Src Port: 45030 (45030),Dst Port: 23 (23),Seq: 144,Ack:
791,Len: 1
Telnet

156 51.157254672   10.0.2.22            10.0.1.11            TELNET   67     Telnet
Data ...

Frame 156: 67 bytes on wire (536 bits),67 bytes captured (536 bits) on interface
0
Ethernet II,Src: 00:0d:bc:ef:eb:24,Dst: 68:05:ca:36:33:a0
Internet Protocol Version 4,Src: 10.0.2.22,Dst: 10.0.1.11
Transmission Control Protocol,Src Port: 23 (23),Dst Port: 45030 (45030),Seq: 791,Ack:
145,Len: 1
Telnet

157 51.186483346   10.0.1.11            10.0.2.22            TELNET   67     Telnet
Data ...

Frame 157: 67 bytes on wire (536 bits),67 bytes captured (536 bits) on interface
0
Ethernet II,Src: 68:05:ca:36:33:a0,Dst: 00:0d:bc:ef:eb:24
Internet Protocol Version 4,Src: 10.0.1.11,Dst: 10.0.2.22
Transmission Control Protocol,Src Port: 45030 (45030),Dst Port: 23 (23),Seq: 145,Ack:
792,Len: 1
Telnet

158 51.187870097   10.0.2.22            10.0.1.11            TELNET   67     Telnet
Data ...

Frame 158: 67 bytes on wire (536 bits),67 bytes captured (536 bits) on interface
0
Ethernet II,Src: 00:0d:bc:ef:eb:24,Dst: 68:05:ca:36:33:a0
Internet Protocol Version 4,Src: 10.0.2.22,Dst: 10.0.1.11
Transmission Control Protocol,Src Port: 23 (23),Dst Port: 45030 (45030),Seq: 792,Ack:
146,Len: 1
Telnet

159 51.187890200   10.0.1.11            10.0.2.22            TELNET   67     Telnet
Data ...

Frame 159: 67 bytes on wire (536 bits),67 bytes captured (536 bits) on interface
0
Ethernet II,Src: 68:05:ca:36:33:a0,Dst: 00:0d:bc:ef:eb:24
```

```
Internet Protocol Version 4,Src: 10.0.1.11,Dst: 10.0.2.22
Transmission Control Protocol,Src Port: 45030 (45030),Dst Port: 23 (23),Seq: 146,Ack:
793,Len: 1
Telnet

160 51.189228372   10.0.2.22            10.0.1.11            TELNET   67     Telnet
Data ...

Frame 160: 67 bytes on wire (536 bits),67 bytes captured (536 bits) on interface
0
Ethernet II,Src: 00:0d:bc:ef:eb:24,Dst: 68:05:ca:36:33:a0
Internet Protocol Version 4,Src: 10.0.2.22,Dst: 10.0.1.11
Transmission Control Protocol,Src Port: 23 (23),Dst Port: 45030 (45030),Seq: 793,Ack:
147,Len: 1
Telnet

183 51.531557677   10.0.1.11            10.0.2.22            TELNET   70     Telnet
Data ...

Frame 183: 70 bytes on wire (560 bits),70 bytes captured (560 bits) on interface
0
Ethernet II,Src: 68:05:ca:36:33:a0,Dst: 00:0d:bc:ef:eb:24
Internet Protocol Version 4,Src: 10.0.1.11,Dst: 10.0.2.22
Transmission Control Protocol,Src Port: 45030 (45030),Dst Port: 23 (23),Seq: 156,Ack:
803,Len: 4
Telnet
```

As you can see in packets 153-160 there are no longer 3 packets per character, only 2. These packets have piggybacked ACKs.
In packet 183, you can even see that there are multiple characters in one packet.

Question 5.C.1.b)
Do you observe delayed acknowledgements? Why is the outcome expected?
There are no delayed ACKs because the ACKS will be piggybacked.

Question 5.C.1.c)
If you type very quickly, i.e., if you hold a key down, you should observe that multiple characters are transmitted in the payload of a segment. Explain this outcome.
This is a part of Nagle's algorithm. Multiple small messages may be combined into one bigger message. This is done to the overhead of TCP.

4. Terminate the Telnet session by typing exit.
5. Stop the traffic capture with Wireshark and save the captured packets.

Question 5.C.2)
Include an example from the saved Wireshark data, which shows that NagleâĂŹs algorithm is used by the TCP sender.

```
No.     Time           Source                Destination           Protocol Length
Info
183 51.531557677   10.0.1.11            10.0.2.22            TELNET   70     Telnet
Data ...
```

```
Frame 183: 70 bytes on wire (560 bits),70 bytes captured (560 bits) on interface
0
...
Telnet
Data: cflz
```

As you can see, 4 characters are sent in one packet. This means that Nagle's algorithm was active.

Part 6. TCP data exchange - Bulk data transfer

The TCP receiver can use acknowledgements 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 that the receiver gets 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 acknowledgement number and the window size is interpreted by the TCP sender, that the sender is allowed to transmit data with sequence numbers 250, 251,..., 359. The TCP sender may have already transmitted some data in that range.

In this part of the lab, you observe acknowledgements and flow control for bulk data transfers, where traffic is generated with the nttcp tool. To observe the bulk data transfer, we introduce a feature of Wireshark that allows you to view the data of a TCP connection in a graph. This is done in Exercise 6-c. We also show how to save the graphs to a file.

All exercises are done with the network configuration from Figure 5.2.

Exercise 6-a. TCP Data Transfer - Bulk transfer (Fast Link)

The purpose of this exercise is to observe the operation of the sliding window flow control scheme in a bulk data transfer, where PC1 sends a large number of segments to PC2, using the nttcp traffic generation tool.

1. The network configuration is the same as in Part 5. If the network is not set up accordingly, then follow the instructions in Exercise 5-a.
2. Start Wireshark on PC1 for interface *eth1*, and start to capture traffic. Do not set any display filters.
3. Use nttcp to generate TCP traffic between PC1 and PC2.
   - On PC2, start a nttcp receiving process by typing:
     ```
     PC2% nttcp -i -rs -l1000 -n500 -p4444
     ```
   - On PC1, start a nttcp sender process that sends 500 blocks of application data by typing:
     ```
     PC1% nttcp -ts -l1000 -n500 -p4444 -D 10.0.5.22
     ```

   By using 10.0.5.22 as destination address, traffic will go over through the direct Ethernet link between PC1 and PC2.
4. From the output of Wireshark on PC1, 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.

The outcome of this experiment is dependent on the data rate of the Ethernet link between PC1 and PC2. If PC1 and PC2 are connected directly by an Ethernet . crossover cable or by a dual-speed hub, they will most likely exchange traffic at a data rate of 100 Mbps. If the Linux PCs are connected by a 10 Mbps Ethernet hub, the data rate is limited accordingly. The rate of the connection has a big impact on the outcome of the experiment.

Include your answers to the following questions. Include captured traffic to support your answers.

Question 6.A.a)

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 ACKs? Can you explain this rule?

In the beginning, the receiver sends ACKS every other packets. After a while, the receiver starts to send an ACK every other packet. The point at which this happens, is when the windows size exceeds the length of the packets sent. After this, not every packets is acknowledged, but rather every other packet is acknowledged.

This can be varified by looking at frame 144 (traces/6-a.5.pcapng). The window size becomes 1452 here, with the packet length being 1448. Shortly after this, the receiver only sends an ACK every other packet instead of every packet like before.

Question 6.A.b)

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?

Typically the receiver acknowledges the maximum segment size or less. The most data that is acknowledged is determined by the window size.

Question 6.A.c)

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?

The window size ranges from 227 to 2082 at the receiver. During the transmission of the data the window size increases.

Question 6.A.d)

Select an arbitrary ACK packet in Wireshark sent by PC2 to PC1. Locate the acknowledgement number in the TCP header. Now 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?

```
No.     Time           Source                Destination           Protocol Length
    Info
  285 10.264549120    10.0.5.11             10.0.5.22             TCP      1514
      42171 âĘŠ 5038 [ACK] Seq=235577 Ack=1 Win=29312 Len=1448 TSval=2090329
      TSecr=2090054

Frame 285: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on
    interface 0
Ethernet II, Src: 68:05:ca:39:cc:79, Dst: 68:05:ca:39:e1:36
Internet Protocol Version 4, Src: 10.0.5.11, Dst: 10.0.5.22
Transmission Control Protocol, Src Port: 42171 (42171), Dst Port: 5038 (5038), Seq:
     235577, Ack: 1, Len: 1448
Data (1448 bytes)

0000  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0040  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0060  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0070  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0080  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
```

```
    0090  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
20  00a0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
    00b0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
22  00c0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
    00d0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
24  00e0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
    00f0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
26  0100  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
    0110  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
28  0120  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
    0130  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
30  0140  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
    0150  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
32  0160  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
    0170  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
34  0180  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
    0190  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
36  01a0  00 00 00 00 00 00 00 00 88 5f 83 45 fa 7f 00 00  .........._.E....
    01b0  88 5f 83 45 fa 7f 00 00 40 d2 91 01 00 00 00 00  ._.E....@.......
38  01c0  40 d2 91 01 00 00 00 00 00 00 00 00 00 00 00 00  @...............
    01d0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
40  01e0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
    01f0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
42  0200  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
    0210  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
44  0220  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
    0230  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
46  0240  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
    0250  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
48  0260  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
    0270  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
50  0280  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
    0290  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
52  02a0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
    02b0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
54  02c0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
    02d0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
56  02e0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
    02f0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
58  0300  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
    0310  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
60  0320  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
    0330  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
62  0340  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
    0350  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
64  0360  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
    0370  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
66  0380  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
    0390  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
68  03a0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
    03b0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
70  03c0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
    03d0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
72  03e0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
    03f0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
74  0400  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
    0410  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
76  0420  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
    0430  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
78  0440  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
    0450  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
80  0460  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
    0470  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
82  0480  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
    0490  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
84  04a0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
    04b0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

```
 86 04c0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
    04d0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
 88 04e0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
    04f0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
 90 0500  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
    0510  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
 92 0520  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
    0530  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
 94 0540  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
    0550  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
 96 0560  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
    0570  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
 98 0580  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
    0590  88 5f 83 45 fa 7f 00 00 88 5f 83 45 fa 7f 00 00   ._.E....._.E....
100 05a0  40 d2 91 01 00 00 00 00                           @.......

102 No.     Time           Source                Destination           Protocol Length
        Info
    323 10.289250815   10.0.5.22             10.0.5.11             TCP      66
        5038 âĘŠ 42171 [ACK] Seq=1 Ack=237025 Win=266496 Len=0 TSval=2090069 TSecr
        =2090329
104
    Frame 323: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
106 Ethernet II, Src: 68:05:ca:39:e1:36, Dst: 68:05:ca:39:cc:79
    Internet Protocol Version 4, Src: 10.0.5.22, Dst: 10.0.5.11
108 Transmission Control Protocol, Src Port: 5038 (5038), Dst Port: 42171 (42171), Seq:
        1, Ack: 237025, Len: 0
```

The time between the sending of the data packet and the reception of the ACK packet was 0.0247 seconds.

Question 6.A.e)

Determine whether, or not, the TCP sender generally transmits the maximum amount of data allowed by the advertised window. Explain your answer.

The sender will always try to send the maximum amount of data allowed. As long as the advertised window limit has not been reached by the sender, it will keep sending data. Wether or not the sender actually reaches the maximum amount of allowed data, depends on the receiver.

Question 6.A.f)

When the nttcp sender has transmitted all its data, it closes the connection, but acknowledgements from PC2 still trickle in. What does PC2 do when it has sent all ACKs?

PC2 sends a TCP packet to the sender with the FIN flag set. This flag is used to close the TCP connection cleanly.

5. Stop the traffic capture with Wireshark and save the Wireshark output.

Exercise 6-b. TCP Data Transfer - Bulk transfer (Slow Link)

This exercise repeats the previous experiment, with the exception that traffic is sent over the emulated serial link.

1. Set the data rate of the serial link to 125 kbps. As in Exercixe-5b, this is done by using the traffic-shaping option.
2. Create a new Wireshark session on PC1 for interface *eth0*, and start to capture traffic. Do not set any display filters.

3. Use nttcp to generate TCP traffic between PC1 and PC2.
   - On PC2, start a nttcp receiving process by typing:
     ```
     PC2% nttcp -i -rs -l1000 -n500 -p4444
     ```
   - On PC1, start a nttcp sender process that sends 500 blocks of application data by typing:
     ```
     PC1% nttcp -ts -l1000 -n500 -p4444 -D 10.0.2.22
     ```

   With the given destination address, traffic will go through the slow serial link.
4. Observe the differences to the data transmission in the previous exercise.
5. Stop the traffic capture with wireshark and save the wireshark output.

Include your answers to the following questions. Emphasize the differences to the observations made in Exercise 6-a.

Question 6.B.a)
How does the pattern of data segments and ACK change, as compared to the fast Ethernet link?
After a while, some ACK's and data packets get retransmitted. The serial link could not keep up with the traffic. The sender didn't receive the ACK in time, and therefor retransmits his data packets. On the other hand, the receiver doesn't receive data for some time and assumes the ACK has been lost and resends it.

Question 6.B.b)
Does the frequency of ACKs change?
Before the duplicate ACK's and retransmissions, ACK's were sent for every packet. After the retransmissions, ACK's were sent every other packet, or even rarely every few packets.

Question 6.B.c)
Is the range of window sizes advertised by the receiver different from those in 6-a?
It is, the advertised window size ranges from 227 to 3428.

Question 6.B.d)
Does the TCP sender generally transmit the maximum amount of data allowed by the advertised window? Explain your answer.
In this exercise, the sender will generally transmit the maximum amount of allowed data. This happens because the serial link in between the sender and receiver is slow, meaning that the sender has ample time to fill the advertised window.

Exercise 6-c. View a graph of TCP data transfer

Wireshark can generate graphs that illustrate the transmissions of segments on a PC connection. This exercise familiarizes you with the graphing capabilities of Wireshark, and shows how you can extract information from the graphs.

1. **Select a TCP connection:** In the Wireshark main window, select a packet from the TCP connection for which you want to build a graph.
   - Here, select a TCP packet sent from PC1 to PC2 in Exercise 6-b.

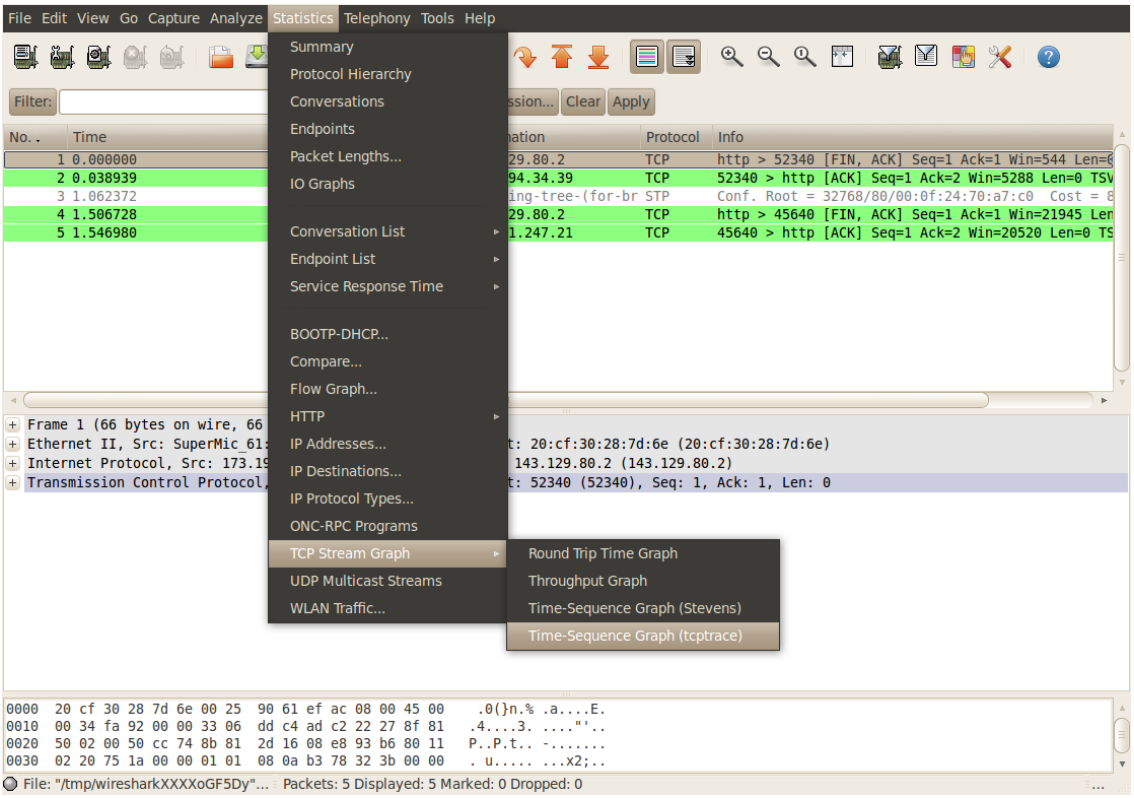


Figure 5.3: Selecting the type of graph for a TCP connection.

2. **Select the type of graph:** Select the Statistics menu from the Wireshark main window, and then select TCP Stream Analysis in the pull down menu, as shown in Figure 5.3. This displays the plotting functions available in Wireshark:

   Time-Sequence Graph (Stevens) Plots the transmission of sequence numbers as a function of time. There is one data point for each transmission of a TCP segment.

   Time-Sequence Graph (tcptrace) Generates a plot as shown in Figure 5.4. The graph is similar to the previous one, but additional information is included on the state of the sliding window.

   Throughput Graph Shows the rate of data transmission as a function of time.

   RTT Graph Shows the roundtrip time (RTT) as a function of time.

   Try out each of the graphs for the TCP connection from Exercise 6-b. Make sure that you select a packet with TCP payload from PC1 to PC2 in the Wireshark main window, before you generate a graph.

3. **Navigating the graphs:** It is possible to navigate the graphs generated by Wireshark. For example, you can zoom into a graph to display an area of interest at a greater level of detail. Here is the complete set of available options:

4. **Interpreting the Time-Sequence Graph (tcptrace):** A lot of information can be extracted from the Time-Sequence Graph (tcptrace). Refer to Figure 5.4 of a TCP connection. This graph shows the transmission of TCP segments and acknowledgements. The short vertical bars indicate the transmission of TCP segments. Each short bar represents one TCP segment, and the length of a bar corresponds to the length of a segment. The figure also shows two step curves. The top step curve shows the highest allowed sequence number,

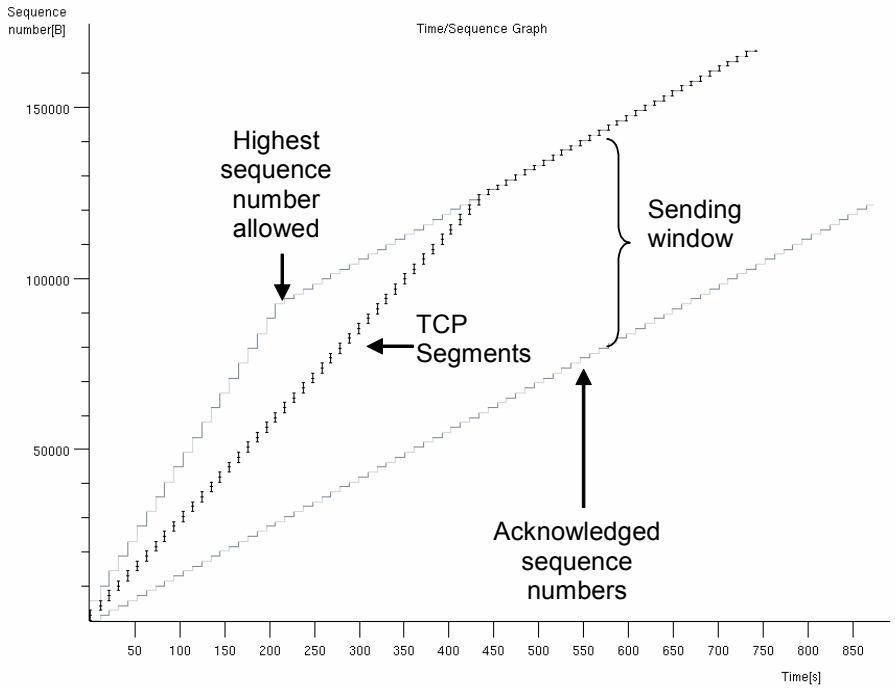


Figure 5.4: Time-Sequence Graph (tcptrace).

Left mouse button	Selecting a data point highlights the corresponding segment in the Wireshark main window.
Middle mouse button	Zooms to a selected part of the graph.
Shift + Middle mouse button	Zooms out.
Right mouse button	Holding this button down and moving the mouse, moves the displayed section of the graph. This only works if the graph is zoomed in.
Ctrl + Right Button	Magnifies a small portion of the graph.
Space	Toggles between showing and not showing crosshairs.
s	Toggles between relative and absolute sequence numbers.
t	Toggles the display of the x-axis.

Table 5.5: **Navigating graphs of TCP connection in Wireshark:**

and the bottom step curve shows the acknowledged sequence numbers. These functions are determined from the acknowledgement number and the window size fields of received ACKs. The vertical distance between the two step curves shows the open part of the sliding window, that is, the sequence numbers that the TCP sender may transmit.
From Figure 5.4, you can see that most of the time, TCP segments are transmitted in groups of two segments. An inspection of the vertical plot shows that no segments are retransmitted. The figure shows that the sequence numbers of transmitted segments are close to the upper step curve. This indicates that the TCP sender utilizes the entire sliding window, and that the transmissions by the TCP sender are triggered by arrivals of ACKs from the TCP receiver.

- Study the Time-Sequence Graph (tcptrace) of the TCP connections in Exercise 6-a and Exercise 6-b. Review the questions in Step 4 of Exercise 6-a and Step 3 in Exer-

cise 6-b, and try to determine the answers to the questions directly from the graphs.

5. **Saving graphs to a file:** Unfortunately, Wireshark does not allow you to save the graphs for a TCP connection. However, there is a simple method in Linux to save a window on the desktop to a file. Suppose you have constructed a TCP graph, similar to that of Figure 5.4, on PC1 and want to save it as a TIFF file. This is done by typing

```
PC1% import lab6c.tif
```

and then clicking on the window with the TCP graph. This saves the graph to a TIFF file with name `lab6c.tif`. If you use a different file extension, the file is saved to a different image format. Select a file format that you can use in your lab report and that has sufficient image quality. The import command supports numerous file formats, including those in Table 5.5. We recommend that you use the TIFF file format, which offers the highest quality image. The size of the file can be reduced to less than 20 kB if you compress the file following the instructions below.

File extension	Format	approximate size of resulting file
.jpeg	JPEG	30kB
.eps	Encapsulated Postscript	3.5 MB
.gif	GIF	300kB
.tif	TIFF	3.5MB

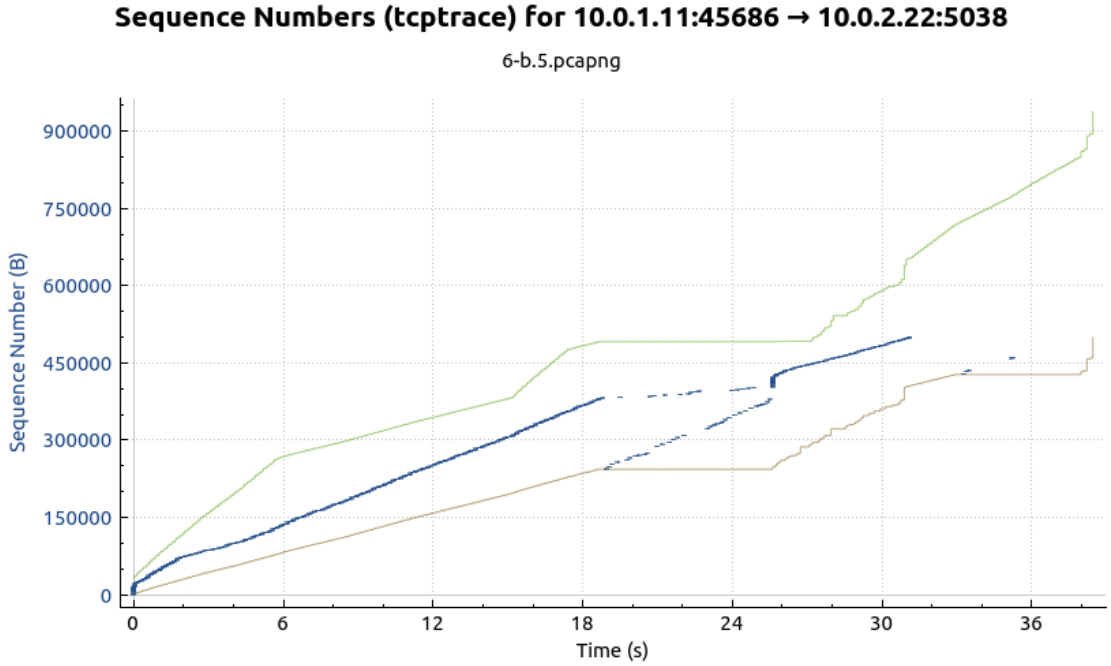
Table 5.6: File formats for import command.

- Save the Time-Sequence Graph (`tcptrace`) that you created for the TCP connections in Exercise 6-a and Exercise 6-b. Select a file format that you can use in your lab report. If you want to include detailed areas from the graphs, you may want to save multiple files for each graph.

- Verify the file has been correctly saved.

Question 6.C)

Include the Time-Sequence Graph (`tcptrace`) graphs that you saved. You may also use these graphs for your answers to the lab report questions for Exercise 6-a and Exercise 6-b.


Sequence Numbers (tcptrace) for 10.0.1.11:45686 → 10.0.2.22:5038
6-b.5.pcapng
Sequence Number (B)
900000
750000
600000
450000
300000
150000
0
0
6
12
18
24
30
36
Time (s)

Part 7. 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:

1. 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 timer 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 acknowledgement for that segment. The RTO value is calculated based on these round-trip measurements (see RFC 2988 from the prelab). Following a heuristic which is called KarnâĂŹs algorithm, measurements are not taken for retransmitted segments. Instead, when a retransmission occurs, the current RTO value is simply doubled.
2. Multiple ACKs have been received for the same segment. A duplicate acknowledgement 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 an indication that a packet has been lost. In this case, the TCP sender does not wait until the timer expires, but immediately retransmits the segment that is presumed lost. This mechanism is known as fast retransmit. The TCP receiver expedites a fast retransmit by sending an ACK for each packet that is received out-of-order.
   A disadvantage of cumulative acknowledgements 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 cumulative acknowledgements only permit to send ACK for segments 1, 2, 3 but not for the other correctly received segments. This may result in an unnecessary retransmission of segments 5, 6, and 7. The problem can be remedied with an optional feature of TCP, which is known as selective acknowledgement (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 or not, is negotiated in TCP header options when the TCP connection is created.

The exercises in this part explore aspects of TCP retransmissions that do not require access to internal timers. Unfortunately, the roundtrip 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 5.2.

Exercise 7-a. TCP Retransmissions

The purpose of this exercise is to observe when TCP retransmissions occur. As before, you transmit data from PC1 to PC2. Here, data is sent over the serial link, which is set to 125 kbps. When you disconnect one of the cables of the network, ACKs cannot reach the sending host. As a result, a timeout occurs and the sender performs retransmissions.

1. The network configuration is the same as in Part 5. If the network is not setup accordingly, then follow the instructions in Exercise 5-a.
2. Set the data rate of the emulated serial link to 125 kbps. If you continue from Part 6, this is the current value of the data rate of the link. If not, you proceed as in Exercise 5-a by setting the the traffic-shaping.
3. Start Wireshark on PC1 and capture traffic on interface *eth0*. Set a display filter to TCP traffic. This is done by typing `tcp` in the window at the bottom of the main window of Wireshark, next to the label Filter.
4. Start a nttcp receiving process on PC2:

   ```
   PC2% nttcp -i -rs -l1000 -n500 -p4444
   ```

5. Start a nttcp sending process on PC1:

   ```
   PC1% nttcp -ts -l1000 -n500 -p4444 -D 10.0.2.22
   ```

Question 7.A.5)
When the connection is created do the TCP sender and TCP receiver negotiate to permit SACKs? Describe the process of the negotiation.
When the sender wants to setup the connection, it sends a TCP packet with the SYN flag set. In the options, the TCP SACK Permitted Option is set to true. After receiving the SYN packet, the receiver transmits the same connection options back to the sender, in order to confirm it received the correct options.

6. Once Wireshark has transmitted at least one hundred packets, disconnect the cable that connects *FastEthernet0/0* of Router3 to the Ethernet hub. Disconnect the cable at the hub. Wait at least five minutes before you reconnect the cable. Observe TCP retransmissions from PC1 in the output of Wireshark.

Question 7.A.6.a)
Observe the time instants when retransmissions take place. How many packets retransmitted at one time?
During the time that the cable was disconnected, the same packet was retransmitted 57 times.

Question 7.A.6.b)
Try to derive the algorithm that sets the time when a packet is retransmitted. (Repeat the experiment, if necessary). Is there a maximum time interval between retransmissions?
The retransmissions we observed were sent just under every 5 seconds. This interval did not seem to change however.

Question 7.A.6.c)
After how many retransmissions, if at all, does the TCP sender give up with retransmitting the segment? Describe your observations.
The sender kept retransmitting packets all the time the cable was disconnected, it did not stop.

7. Now reconnect the cable, and wait until the transmission resumes (This may take some time). Now quickly disconnect and reconnect the cable that connects the interface *FastEthernet0/0* of Router3 to the Ethernet hub. Repeat this procedure a number of times, by varying the length of time that the cable is disconnected. Now observe the retransmissions from PC1.

Question 7.A.7)

Are the retransmissions different from those in Step 6? Specifically, do you observe fast retransmits and/or SACKs?

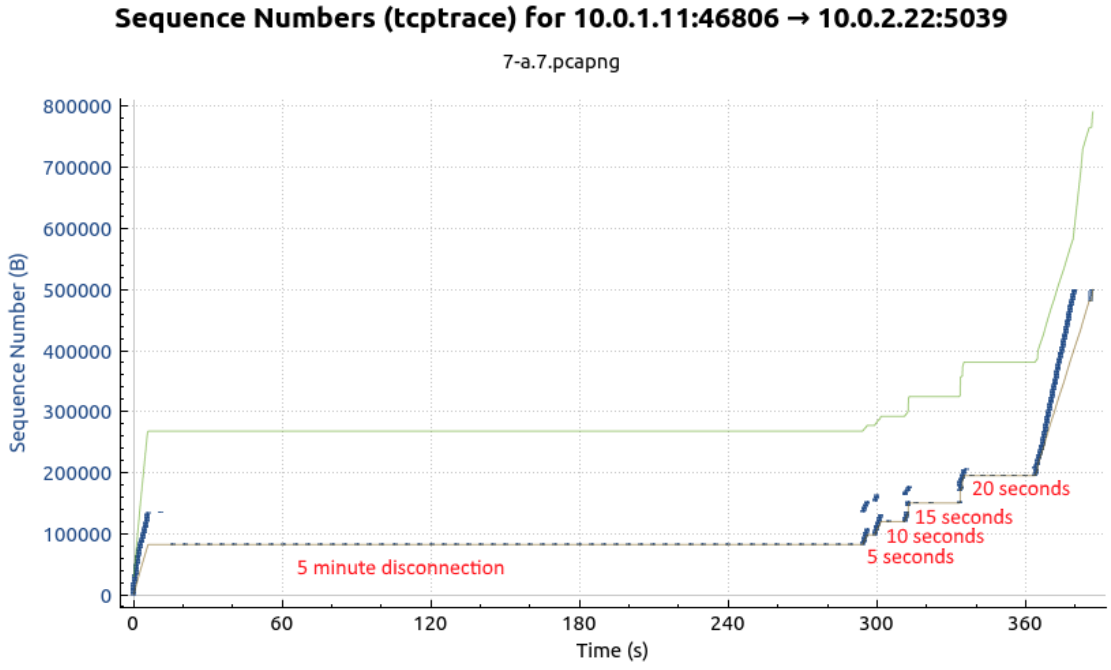
We did not observe fast retransmits when disconnecting the cables. We did, however, observe SACKs. The SACK's are included in the duplicate ACK packets, and specify a range of data that is has received. The other data that is missing has to be resent by the sender (as opposed to all the data following the lost packet).

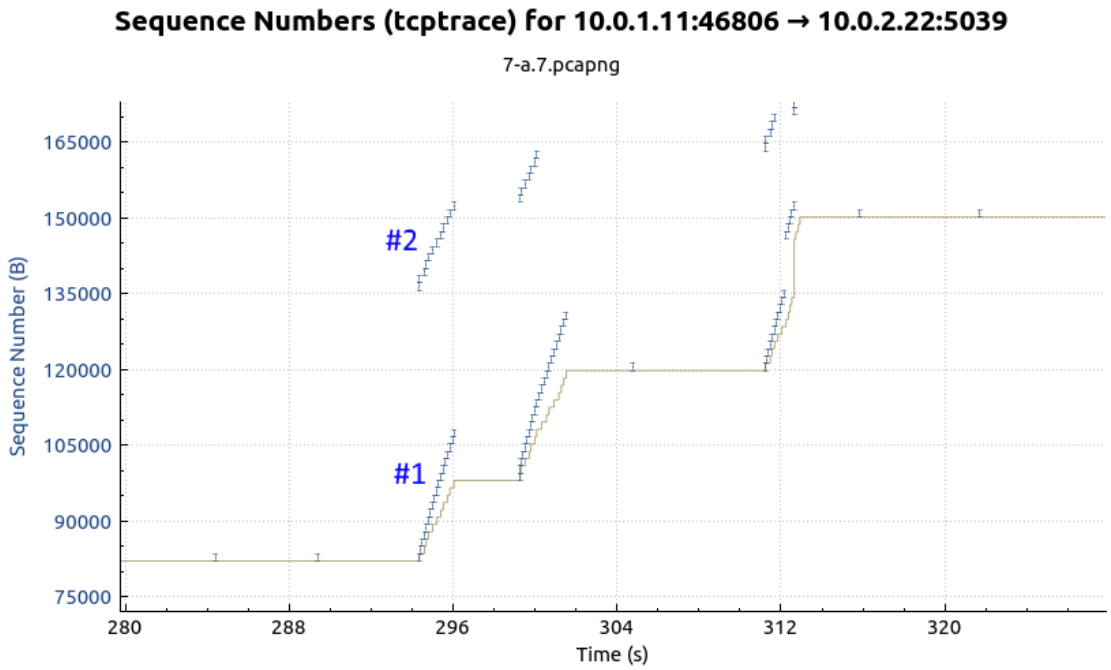
8. Use the instructions from Exercise 6-c to build a Time-Sequence Graph (tcptrace) in Wireshark for the TCP connection. Study the output of the graph and use the graph to provide answers to the questions above. Use the navigating features to zoom in to parts of the graph that are of interest.

9. Follow the instructions from Exercise 6-c to generate image files for the Time- Sequence Graph (tcptrace). Save enough images so that you can use the graphs to answer the above questions in your lab report. Generate images that show clearly the retransmission attempts in Step 6 and Step 7.

Question 7.A.9)

Include the image files saved in Step 9, and use them to support your answers. Annotate the graphs as necessary.





In the zoomed in version, you can see that (at 296 seconds) the cable gets disconnected. After the cable got connected again, the sender did not resend all the packets from the lowest sequence number onwards. This can be explained by the SACK's sent by the receiver. The packets with a high sequence number (blue line #2) were not resent.

Exercise 7-b. TCP performance at an overloaded link

Next you perform an experiment, where you overload the emulated serial link between Router3 and Router4, and cause losses and retransmissions due to buffer overflows at Router3.

As in Exercise 7-a, you set up a TCP connection from PC1 to PC2. Here, however, you flood Router 3 with ICMP Echo Request messages. The purpose of this exercise is to observe how a TCP connection performs when a router is overloaded.

1. Set the data rate of the serial link to 64 kbps.
2. Start Wireshark on PC1 for interface *eth0* interface, and start to capture traffic. Set a display filter to TCP traffic.
3. Start a nttcp receiving process on PC2:

   ```
   PC2% nttcp -i -rs -l1000 -n500 -p4444
   ```

4. Start a nttcp sending process on PC1:

   ```
   PC1% nttcp -ts -l1000 -n500 -p4444 -D 10.0.2.22
   ```

5. Once Wireshark has transmitted at least one hundred TCP packets, start to flood ICMP Echo Request messages by typing on PC1

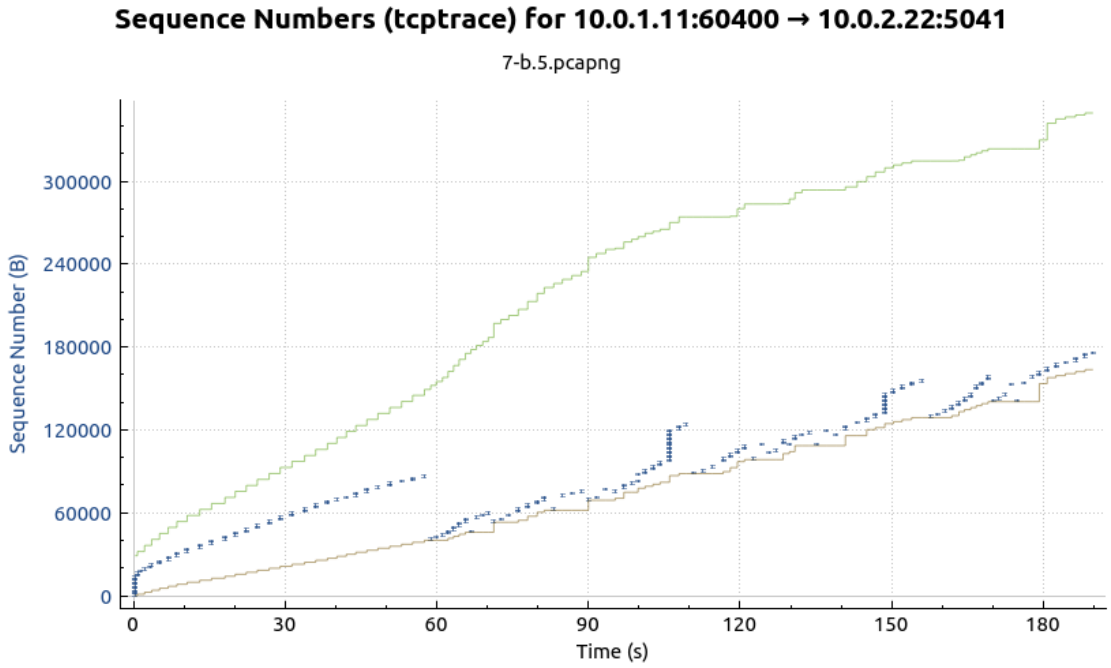
   ```
   PC1% ping -f 10.0.2.22
   ```

   Recall that with the `-f` option, PC2 sends ICMP Echo Request packets as fast as possible. The ICMP traffic sent from PC1 to PC2 will overflow the buffers of Router3 at the serial link.

6. Follow the instructions from Exercise 6-c to build a Time-Sequence Graph (tcptrace) in Wireshark for the TCP connection. If you do not observe any retransmissions, reduce the data rate of the serial link. As long as the nttcp sender is transmitting packets, rerun the construction of the graph to observe how the graph changes as time progresses. In the graph, observe the progress of the TCP connection:
7. Follow the instructions from Exercise 6-c to generate image files for the Time- Sequence Graph (tcptrace) and Throughput Graph. Save enough images so that you can use the graphs to answer the above questions in your lab report. Generate an image that shows in detail the loss events that occur right after the ping command is started.

Question 7.B)

Include your answers to the questions in Step 6. Included the saved image files from step 7. Annotate and describe the plots to support your answers.



Question 7.B.6.a)

Describe the losses that occur in the graph when the ping command is started. Do losses occur in regular intervals or irregularly?

The ping command is issued at 23 seconds. The packets sent at that point have to be partially resent later, because some of them are lost due to the flooding. The losses seem to occur in regular intervals, by looking at the graph. Some intervals packets get ACK'ed, in the other intervals no packets get ACK'ed.

Question 7.B.6.b)

From the graph, describe the size of the advertised window changes when the flooding ping is started.

The size of the advertised window decreases more and more. This can be seen by analyzing the height difference between the blue lines/points (TCP frames), and the line below (ACK'ed packets). The further we advance in time, the smaller the height difference gets.

Question 7.B.6.c)

Try to determine if retransmissions occur due to fast retransmit or due to timeouts of the

timers. How can you determine which type of retransmissions you observe?

```
No.     Time           Source                Destination           Protocol Length
    Info
   8093 108.264114958  10.0.2.22             10.0.1.11             TCP      86
       5041 âĘŠ 60400 [ACK] Seq=1 Ack=70505 Win=180992 Len=0 TSval=2696368 TSecr
       =2695307 SLE=71953 SRE=74849 SLE=83537 SRE=86897

Frame 8093: 86 bytes on wire (688 bits), 86 bytes captured (688 bits) on interface
    0
Ethernet II, Src: 00:0d:bc:ef:eb:24, Dst: 68:05:ca:36:33:a0
Internet Protocol Version 4, Src: 10.0.2.22, Dst: 10.0.1.11
Transmission Control Protocol, Src Port: 5041 (5041), Dst Port: 60400 (60400), Seq:
     1, Ack: 70505, Len: 0
    Source Port: 5041
    Destination Port: 60400
    [Stream index: 3]
    [TCP Segment Len: 0]
    Sequence number: 1    (relative sequence number)
    Acknowledgment number: 70505    (relative ack number)
    Header Length: 52 bytes
    Flags: 0x010 (ACK)
    Window size value: 1414
    [Calculated window size: 180992]
    [Window size scaling factor: 128]
    Checksum: 0x89c5 [validation disabled]
    Urgent pointer: 0
    Options: (32 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps, No-
        Operation (NOP), No-Operation (NOP), SACK
    [SEQ/ACK analysis]

No.     Time           Source                Destination           Protocol Length
    Info
   8094 108.264126328  10.0.1.11             10.0.2.22             TCP      1514
       [TCP Retransmission] 60400 âĘŠ 5041 [ACK] Seq=74849 Ack=1 Win=29312 Len=1448
        TSval=2696633 TSecr=2696368

Frame 8094: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on
    interface 0
Ethernet II, Src: 68:05:ca:36:33:a0, Dst: 00:0d:bc:ef:eb:24
Internet Protocol Version 4, Src: 10.0.1.11, Dst: 10.0.2.22
Transmission Control Protocol, Src Port: 60400 (60400), Dst Port: 5041 (5041), Seq:
     74849, Ack: 1, Len: 1448
    Source Port: 60400
    Destination Port: 5041
    [Stream index: 3]
    [TCP Segment Len: 1448]
    Sequence number: 74849    (relative sequence number)
    [Next sequence number: 76297    (relative sequence number)]
    Acknowledgment number: 1    (relative ack number)
    Header Length: 32 bytes
    Flags: 0x010 (ACK)
    Window size value: 229
    [Calculated window size: 29312]
    [Window size scaling factor: 128]
    Checksum: 0x1cef [validation disabled]
    Urgent pointer: 0
    Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
    [SEQ/ACK analysis]
    Retransmitted TCP segment data (1448 bytes)
```

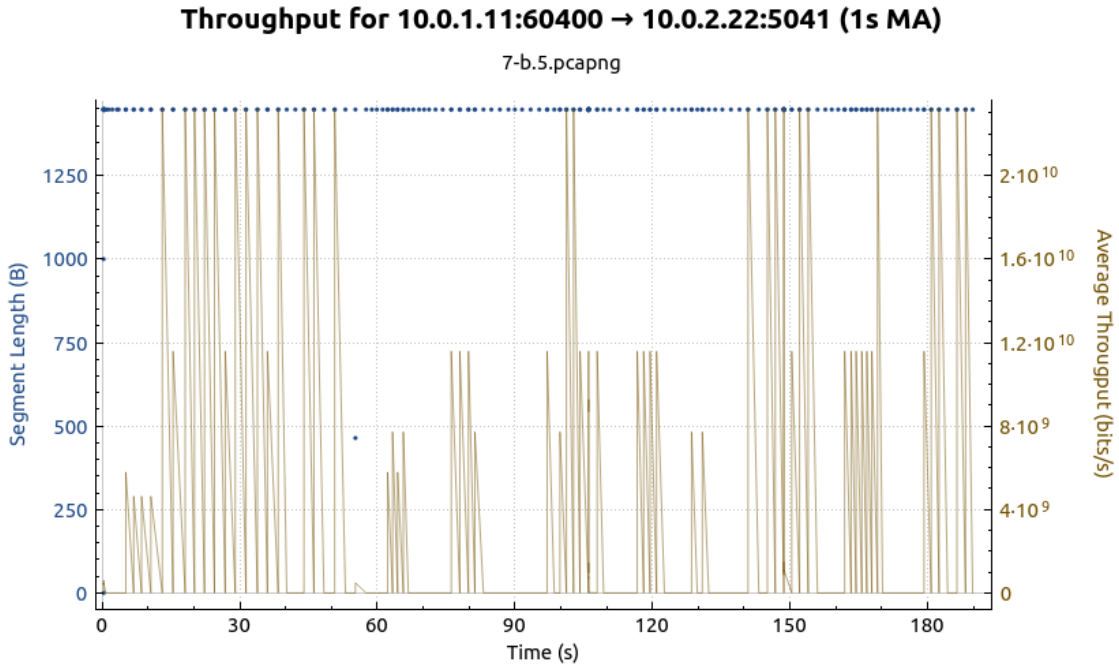
traces/7–b.5.timeout.txt

Judging by the packets, the retransmissions occur because of a timeout. The receiver sends an ACK for 70505, while the sender sends a retransmission for 74849. If the retransmission

didn't occur by timeout, the retransmission of 74849 would not happen before receiving the concerning information from the receiver.

Question 7.B.6.d)

Generate a Throughput Graph to view the data rate of the TCP connection. How does the throughput change after the flood of pings is started?



After the flood of pings starts (23 seconds), the throughput in general lowers. No data is sent at certain intervals and results in the throughput being 0. Other intervals there is some throughput, but less than before the flooding.

Part 8. TCP Congestion Control

TCP congestion control consists of a set of algorithms that adapt the sending rate of a TCP sender to the current conditions in the network. When the network is not congested, the TCP sender is allowed to increase its sending rate, and when the network is congested, the TCP sender reduces its rate. The TCP sender maintains a congestion window which limits the number of segments that can be sent without waiting for an acknowledgement. The actual number of segments that can be sent is the minimum of the congestion window and the window size sent by the receiver.

For congestion control, each TCP sender keeps two variables, the congestion window (cwnd) and the slow-start threshold (ssthresh). The initial values are set to one segment for cwnd and 65535 bytes for ssthresh. TCP congestion control operates in two phases, called slow start and congestion avoidance. The sender is in the slow start phase when cwnd $\leq$ ssthresh. Here, cwnd is increased by one for each arrived ACK. This results in a doubling of cwnd for each roundtrip time. When cwnd >ssthresh, the TCP sender is in the congestion avoidance phase. Here, the cwnd is incremented by one only after cwnd ACKs. This is done by incrementing cwnd by a fraction of a segment when an ACK arrives.

The TCP sender assumes that the network is congested when a segment is lost, that is, when the retransmission timer has a timeout or when three duplicate ACKs arrive. When a timeout occurs, the TCP sender sets ssthresh to half the current value of cwnd and then sets cwnd to one. This puts the TCP sender in slow start mode. When a third duplicate ACK arrives, the TCP sender performs what is called a fast recovery. Here, ssthresh is set to half the current value of cwnd, and cwnd is set to the new value of ssthresh.

The goal of this part of the lab is to observe the development of the congestion window. Since the number of the segments that can be transmitted by a TCP sender is the result of the congestion window as well as the advertised window, and since data segments and returning ACKs interleave, the size of the congestion window is not derived by observing traffic.

Exercise 8-a. Network Setup

The network configuration used is that in Figure 5.2. To observe the slow start features, change the routing table entries so that traffic from PC1 to PC2 traverses the path PC1 $\rightarrow$ R1 $\rightarrow$ R2 $\rightarrow$ PC2, and the reverse path is PC2 $\rightarrow$ R4 $\rightarrow$ R3 $\rightarrow$ PC1. When PC1 sends data to PC2, data segments can be transmitted quickly to PC2, but ACKs only slowly return to PC1. The sender will therefore transmit a full window of packets up to the threshold of the congestion window, and then be forced to wait to receive the ACKs before transmitting the next batch of packets.

1. The network configuration is similar to that in Parts 5-7. If the network is not setup accordingly, then follow the instructions in Exercise 5-a. The following two steps are modifications to the setup of Exercise 5-a.

2. Set a new default gateway of PC1 to Router1. If the default gateway from Table 5.4 is still set, you must first delete the existing entry. Use the command `netstat -rn` to see if a default gateway is configured. Assuming that the configuration from Table 5.4, you must enter the following commands:

```
PC1% route del default gw 10.0.1.3
PC1% route add default gw 10.0.1.1
```

The default gateway of PC2 remains unchanged and should be as shown in Table 5.4. With this modification, traffic from PC1 to PC2 passes through Router1 and Router2, and traffic from PC2 to PC1 passes through Router4 and Router3. Verify that this is the case.

3. Set the data rate of the emulated serial link to 1 Mbps.

Exercise 8-b. Observing TCP congestion control

This exercise is similar to Exercise 6-a, that is, PC1 transmits TCP segments to PC2.

1. Start Wireshark for interface *eth0* on PC1, and start to capture traffic. Set a display filter to TCP traffic.

2. Start a nttcp receiving process on PC2:

```
PC2% nttcp -i -rs -l1000 -n5000 -p4444
```

3. Start a nttcp sending process on PC1 that transmits 5000 blocks of data, each with 1000 Bytes:

```
PC1% nttcp -ts -l1000 -n5000 -p4444 -D 10.0.2.22
```

4. Once Wireshark has transmitted at least one hundred TCP packets, disconnect the cable that connects *FastEthernet0/0* of Router1 to the Ethernet hub. Disconnect the cable at the hub. Now reconnect the cable, and wait until the transmission resumes. Repeat this for a few times, varying the durations when the cable is disconnected.

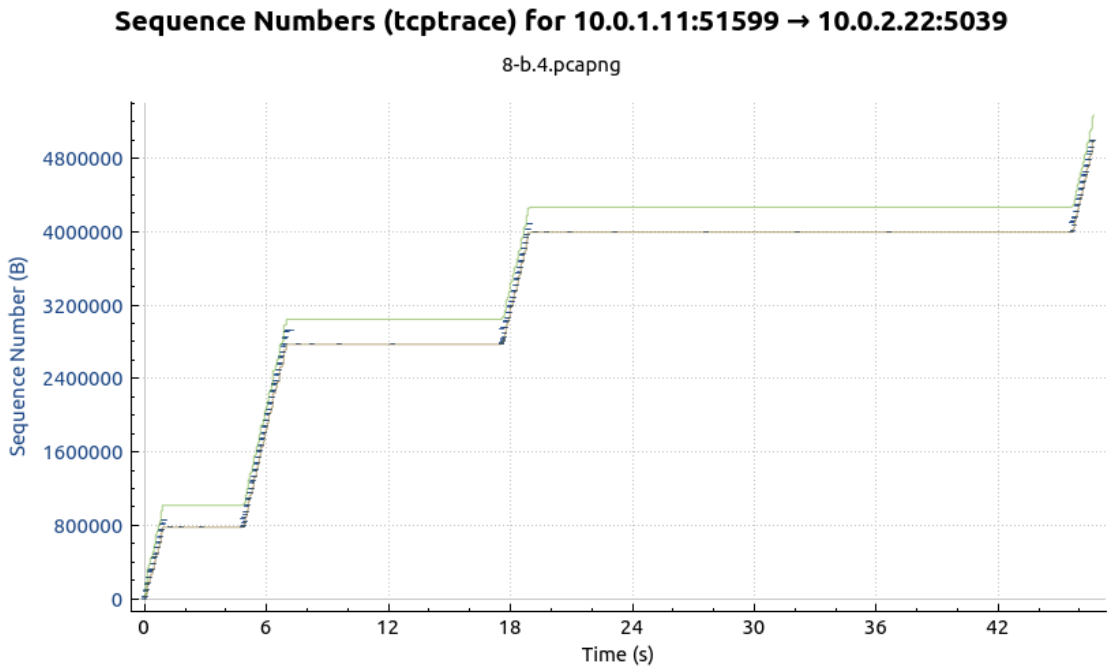
5. Use the instructions from Exercise 6-c to build a Time-Sequence Graph (tcptrace) in Wireshark for the TCP connection. Study the graph at the time instants when the cable is reconnected and TCP sender resumes transmission. Use the navigating features to zoom in to parts of the graph that are of interest.

The outcome of this experiment is dependent on the data rate of the link between Router1 and Router2, and Router3 and Router4, respectively. The outcome of the experiment is different when the Ethernet link between Router1 and Router2 is running at 10 Mbps or at 100 Mbps. The above settings are optimized for a 10 Mbps Ethernet link between Router1 and Router2.

6. Follow the instructions from Exercise 6-c to generate image files for the Time- Sequence Graph (tcptrace). Save enough images so that you can use the graphs to answer the above questions in your lab report. Make sure you include an image that shows a portion of the graph that illustrates the slow start phase. Compress the files and copy the files to a floppy disk.

Question 8.B)

Include the answers to the questions from Step 5. Use the saved image files to support your answers. Annotate the events in this graph, and explain the events that you observe, e.g. segments dropped, retransmission, congestion window, slow start, congestion avoidance, fast recovery, etc.



Question 8.B.5.a)

Try to observe periods when the TCP sender is in a slow start phase and when the sender switches to congestion avoidance. Verify that the congestion window follows the rules of the slow start phase.

We couldn't solve this question because our output had incorrect data and we noticed this too late.

Question 8.B.5.b)

Can you deduct the size of the ssthresh parameter during the times when the congestion window is small?

We couldn't solve this question because our output had incorrect data and we noticed this too late.

Question 8.B.5.c)

Can you find occurrences of fast recovery?

We couldn't solve this question because our output had incorrect data and we noticed this too late.