**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
Josse Coen Armin Halilovic Jonas Vanden Branden Group 2

April 20, 2016

# 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.

**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.

**Question 3.a)**
How does TCP decide the maximum size of a TCP segment?

**Question 3.b)**
How does UDP decide the maximum size of a UDP datagram?

**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?

**Question 3.d)**
Explain why a TCP connection over an Ethernet segment never runs into problems with fragmentation.

**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?

**Question 5.a)**
Describe Nagle's algorithm and explain why it is used in TCP

**Question 5.b)**
Describe Karn's Algorithm and explain why it is used in TCP

**Question 6.a)**
What is a delayed acknowledgement in TCP?

**Question 6.b)**
What is a piggybacked acknowledgement in TCP?

**Question 7)**
Describe how the retransmission timeout (RTO) value is determined in TCP.

**Question 8.a)**
Describe the sliding window flow control mechanism used in TCP .

**Question 8.b)**
Describe the concepts of slow start and congestion avoidance in TCP.

**Question 8.c)**
Explain the concept of fast retransmit and fast recovery in TCP.

## 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 **??**. 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 **??**.

## 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.

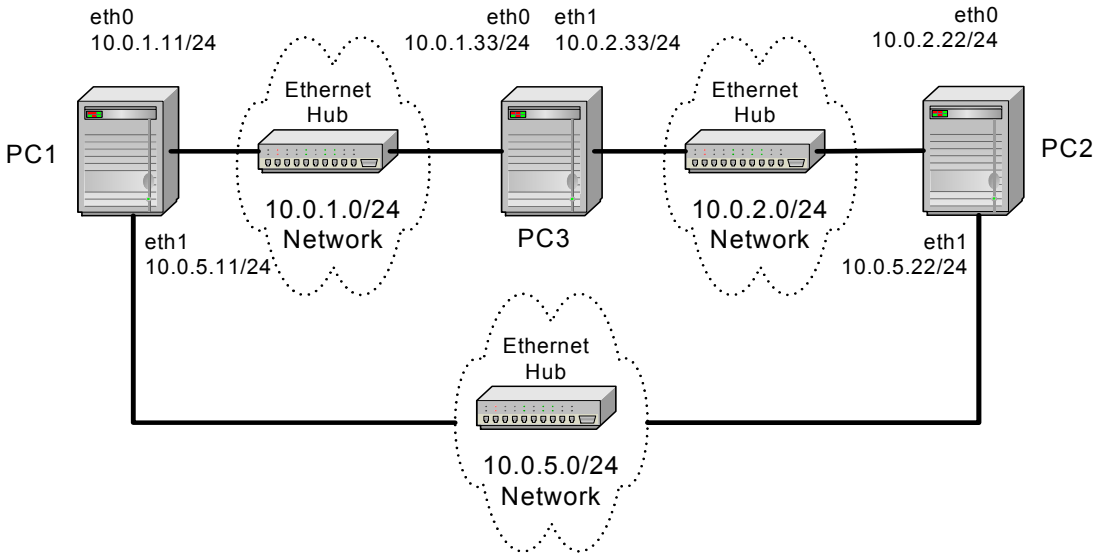


Figure 5.1: Network Topology for Parts 1-4.

| 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?
We can count 14 UDP packets in the data transfer. (We only consider UDP packets, since only they are part of the actual file transfer.) Only one packet per UDP datagram was transmitted; there was no fragmentation. We see three different UDP datagrams, their difference lies in the payload:

| Amount of packets | Payload length |
|---|---|
| 1 | 4 |
| 10 | 1024 |
| 3 | 4 |

**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.
We count 10676 bytes transmitted in total, 10256 bytes of UDP payload data. That means there was approximately 5% of overhead data.

**Question 1.B.1.c)**

Inspect the fields in the UDP headers. Which fields in the headers do not change in different packets?
The source port and destination port fields never change in different packets.

**Question 1.B.1.d)**

Observe the port numbers in the UDP header. How did the nttcp sender select the source port number?
source port: 40951, destination port: 5038

We notice that port 4444 is not in these packets. That port is used by the TCP packets sent by nttcp to set up the connection. For the transmitted UDP packets, we believe that a random source port is selected randomly from a range of ports that are not assinged to a particular application layer protocol. In fact, the port that was used, 40951, is indeed unassigned according to this web page: http://whatismyipaddress.com/port-list

### 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?

We count 16 packets in the data transfer.

| packet numbers | packet length |
|---|---|
| 10, 15 - 27 | 66 |
| 8, 9 | 74 |
| 11 | 1090 |
| 12 | 1514 |
| 13 | 2962 |
| 14 | 4938 |

**Question 1.C.1.b)**

What is the range of the sequence numbers?
Lowest sequence number: 0
Highest sequence number: 10242 (packet no. 27)

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

Packets transmitted by PC1: 7
Packets transmitted by PC2: 9

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

12 of the packets are control packets.
Their packet numbers are: 8, 9, 10, 15 - 21, 26, 27

**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.
From PC1 to PC2: 7 packets, total packet length: 10710 bytes, total application data length: 10240 bytes

From PC2 to PC1: 9 packets, total packet length: 602 bytes, total application data length: 0 bytes

We see that only PC1 has sent application data, and more specifically has sent 10 * 1024 bytes of data (flags -l1024 -n10 in nttcp).

**Question 1.C.1.f)**
Inspect the TCP headers. Which packets contain flags in the TCP header? Which types of flags do you observe?

| packet numbers | flag |
|---|---|
| 8, 9 | SYN |
| 9 - 21, 26, 27 | ACK |
| 11, 14 | PSH |
| 14, 26 | 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.
Total:
UDP transfer: 10676 B; TCP transfer: 11312 B
As is expected, the total amount of data transferred over TCP is greater than the amount of data transferred via UDP, since the TCP header is larger and the UDP received does not send any acknowledgements. Thus, using TCP, there was an increase of approximately 6%.
Payload data:
The TCP transfer had exactly the amount of payload data we'd expect: 10 * 1024 B = 10240 B. The UDP transfer had 16 B more, spread over four packets each carrying 4 B of payload data. We don't know why that's the case.

**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 had a length of 1066 bytes and a payload of 1024 bytes. The biggest TCP segment had a length of 4938 bytes and payload of 4872 bytes.

We see that the TCP relatively holds much more application data here.

## 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.

FTP: 28 -> 789 (22.95 -> 24.13) => 1.18 seconds

TFTP: 824 -> 4952 (406.71 -> 408.37) => 1.66 seconds
TFTP: 4955 -> 9083 (415.80 -> 417.44) => 1.64 seconds

We see that, indeed, FTP outperforms TFTP. The size of the data in FTP data packets ranges from 1448 bytes to 7240 bytes. In the TFTP packets this is 512 bytes (and 350 bytes for the last packet).

Since TCP operates a sliding window for acknowledgements, the FTP sender doesn't have to wait for every packet's ACK before sending the next packet. Only when the delay of ACKs is large enough (determined by the sliding window parameters) the sender will start retransmitting packets.
With TFTP, the sender *does* have to wait for the ACK of every packet, so, every time a packet is sent after the receiver has received the packet, no data is sent while the ACK is traveling back to the sender.

**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.

Source ports used: 21 for ftp settings, 20 for sending ftp data
Destination ports used: 60889 for ftp settings, 42752 for sending ftp data

## 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).

Taken from “Lab 5/traces/3.A.PC1.pcap”:

**** WHY NO. 72? Lijkt erop dat 70 ook al fragmentatie ondervindt, dus moeten denk ik een datagram hebben van daarvoor. – Josse ****

the complete datagram before fragmentation:

```
No.     Time        Source                Destination           Protocol Length
    Info
     72 8.015696    10.0.1.11             10.0.2.22             UDP      35
         Source port: 39068  Destination port: 5038

Frame 72: 35 bytes on wire (280 bits), 35 bytes captured (280 bits)
    Encapsulation type: Ethernet (1)
    Arrival Time: Mar 15, 2016 11:35:52.642696000 CET
    [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: 1458038152.642696000 seconds
    [Time delta from previous captured frame: 0.000002000 seconds]
    [Time delta from previous displayed frame: 0.000002000 seconds]
    [Time since reference or first frame: 8.015696000 seconds]
    Frame Number: 72
    Frame Length: 35 bytes (280 bits)
    Capture Length: 35 bytes (280 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: eth:ip:udp:data]
    [Coloring Rule Name: UDP]
    [Coloring Rule String: udp]
Ethernet II, Src: IntelCor_1a:7c:70 (68:05:ca:1a:7c:70), Dst: IntelCor_1a:80:18
    (68:05:ca:1a:80:18)
    Destination: IntelCor_1a:80:18 (68:05:ca:1a:80:18)
        Address: IntelCor_1a:80:18 (68:05:ca:1a:80:18)
        .... ..0. .... .... .... .... = LG bit: Globally unique address (factory
            default)
        .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
    Source: IntelCor_1a:7c:70 (68:05:ca:1a:7c:70)
        Address: IntelCor_1a:7c:70 (68:05:ca:1a:7c:70)
        .... ..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: 21
    Identification: 0x07ef (2031)
    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: 0x5b10 [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]
    [2 IPv4 Fragments (1481 bytes): #71(1480), #72(1)]
        [Frame: 71, payload: 0-1479 (1480 bytes)]
        [Frame: 72, payload: 1480-1480 (1 byte)]
        [Fragment count: 2]
        [Reassembled IPv4 length: 1481]
        [Reassembled IPv4 data: 989c13ae05c9462e1472baa5935b4c7b3df37df0ae953907
            ...]
User Datagram Protocol, Src Port: 39068 (39068), Dst Port: 5038 (5038)
    Source port: 39068 (39068)
    Destination port: 5038 (5038)
    Length: 1481
    Checksum: 0x462e [validation disabled]
        [Good Checksum: False]
        [Bad Checksum: False]
Data (1473 bytes)

0000  14 72 ba a5 93 5b 4c 7b 3d f3 7d f0 ae 95 39 07   .r...[L{=.}...9.
0010  c7 80 76 11 3b 82 f6 e7 aa c5 75 64 d8 c6 ab 8f   ..v.;.....ud....
0020  3e 6f 74 2a ce 1d 02 10 3a f9 85 56 55 20 e4 0f   >ot*....:..VU ..
0030  e8 69 4f 0d 97 71 ba 8c 59 b7 25 9e 9c 67 b2 81   .iO..q..Y.%..g..
0040  9f 67 2d f1 d9 0d ac 8e 01 c2 6d cb 1d a0 e6 b3   .g-.......m.....
0050  15 9f 8b ad ad 01 0c d0 c8 ad 21 1e 95 30 dd 1b   ..........!..0..
0060  2f 7b 2a e9 64 76 56 28 b4 4c 51 5c ca e3 85 ab   /{*.dvV(.LQ\....
0070  f8 dd 84 6a bc 50 fd e1 57 c9 f6 74 cb 87 aa 56   ...j.P..W..t...V
0080  2c 6e fb 08 7a 14 ee ff 78 10 3e 3b 4b bb 24 18   ,n..z...x.>;K.$.
0090  bb e1 d1 0f b5 80 0f 72 22 b9 72 0e a2 40 fa bb   .......r".r..@..
00a0  4b 20 f3 99 bf 83 18 08 5e 30 9a cf 7a a2 c6 ba   K ......^0..z...
00b0  02 61 14 2b cf 14 37 29 30 f2 91 36 32 3f bb 6c   .a.+..7)0..62?.l
00c0  f6 84 12 0f 8c c5 8a 8b 0c 20 8a 52 04 06 a6 74   ......... .R...t
00d0  3e a2 e9 1a 82 a0 b5 c0 8c f0 cd 7a e5 3d a3 98   >..........z.=..
00e0  78 18 07 0c a6 3b 37 16 54 39 d8 c5 b5 34 8e 2c   x....;7.T9...4.,
00f0  9e f9 ec 95 8d 03 54 06 95 57 3c c6 f6 f4 cf 91   ......T..W<.....
0100  af fe 96 0b a0 74 c4 29 69 fa 26 d7 bf e7 4f 49   .....t.)i.&...OI
0110  71 4d fc e9 1b 9f d0 bc bf 61 c1 17 64 14 f5 56   qM.......a..d..V
0120  b9 34 22 42 64 87 5a 95 64 00 9a 4b f6 7f 99 d3   .4"Bd.Z.d..K....
0130  fa 76 e2 fc 05 6d 05 d8 31 99 97 39 1d 6b 16 77   .v...m..1..9.k.w
0140  0e 48 bb c0 0c df 36 6f f2 84 3c af 79 3c 9b 33   .H....6o..<.y<.3
0150  33 6b a4 5b a5 0c 9a bc 18 91 a4 77 0a 63 36 72   3k.[.......w.c6r
0160  50 8e c0 72 f3 b5 d0 61 25 1e 38 51 74 a7 c6 b5   P..r...a%.8Qt...
0170  c0 42 cf b1 ee 73 c7 21 91 61 3a 3a c2 d0 bf a1   .B...s.!.a::....
0180  da 71 54 8e c7 d7 de 36 27 4c bd 74 86 33 73 76   .qT....6'L.t.3sv
0190  fb bd b3 16 8c 98 a9 e0 8e 7f 62 49 09 bf 68 3c   ..........bI..h<
01a0  ff 47 17 41 d7 33 93 b9 59 6a be de 75 ea d8 70   .G.A.3..Yj..u..p
01b0  23 dc 66 4b a5 14 70 5b 3c 36 63 e0 12 95 c7 33   #.fK..p[<6c....3
01c0  f5 12 df da fc b0 f8 03 cb 71 e6 93 70 8d bb bd   .........q..p...
01d0  de d6 fb fe 85 c1 25 31 7c ae 17 ea 83 30 09 42   ......%1|....0.B
01e0  0b 8d 67 0a b3 61 ee 91 82 b6 22 e1 7d 31 fa 09   ..g..a...."."}1..
01f0  37 30 e7 07 b2 06 84 07 7a 41 ce db 88 33 f1 00   70......zA...3..
0200  63 ef dc 3a 64 18 0e 09 3c 8e 50 dc 9d d3 21 5d   c..:d...<.P...!]
0210  00 03 1e bb b0 43 ea d3 22 88 8f 13 6e ff 10 01   .....C.."...n...
0220  69 e6 12 fb 27 f8 e0 f3 01 7a 1a f8 9d 9e 4c 64   i...'....z....Ld
0230  ba b0 0d 66 d0 2e 5c bc 25 e4 b7 11 d1 7e d2 aa   ...f..\.%....~..
0240  dc b7 82 a7 e0 c9 32 99 8f 43 68 57 92 38 45 1e   ......2..ChW.8E.
0250  0f 7b 2b 64 b6 a8 d0 9f 86 8a 53 45 b1 94 60 c5   .{+d......SE..`.
0260  5d da 98 c0 ee aa 8a 22 07 90 f1 3e 61 a6 04 f0   ].......".. >a...
0270  f7 28 d3 6a b7 5d c8 63 de 4f 4b f3 e2 ac 26 aa   .(.j.].c.OK...&.
0280  ef cc 2e 7b 8d 93 81 81 75 f2 fc c7 f2 d8 57 10   ...{....u.....W.
0290  9a 11 8e 1c 8d 95 33 db 70 32 f6 7b 99 11 0b ae   ......3.p2.{....
02a0  40 98 04 6a bd c0 ce 5a 9b 9a f0 fe ee c4 b6 7a   @..j...Z.......z
02b0  8f 4b 3d 5f 85 f2 e9 b2 3f d3 06 13 a5 c0 04 2d   .K=_....?......-
02c0  66 4f af cb 56 9e 16 5d 95 e1 62 f3 1f 49 94 10   fO..V..]..b..I..
02d0  5b e3 d6 72 a0 68 24 d0 c0 70 92 02 24 62 e4 f3   [..r.h$..p..$b..
02e0  d6 d4 9c b9 59 ca 19 87 52 d9 a8 b6 6d 54 f7 b7   ....Y...R...mT..
```

```
    02f0  10 67 bf 1c 14 43 48 dd 5b 8a 41 e8 4c cb 5a 79   .g...CH.[.A.L.Zy
115 0300  a8 0d cc 79 78 0f f7 64 7c 95 63 dd 51 f0 43 a1   ...yx..d|.c.Q.C.
    0310  46 f5 88 67 43 3d 15 72 83 80 e8 93 e0 07 e0 0a   F..gC=.r........
117 0320  38 a4 11 e0 0b 76 81 13 d9 57 61 c0 3f 90 ab b7   8....v...Wa.?...
    0330  4d b7 8b a4 2f 15 bb c3 12 64 86 2c 18 57 1e 27   M.../....d.,.W.'
119 0340  f6 2f f5 cc 23 6a cd bd b7 f7 bd 44 f8 62 a5 0d   ./..#j.....D.b..
    0350  10 9b ab ec 1c ea 7a c2 fe d3 00 4f 79 a2 1d b5   ......z....Oy...
121 0360  c3 b6 17 cb 12 e8 13 06 6c d4 27 e8 a3 f8 b5 79   ........l.'....y
    0370  aa 87 2d 99 9f 6f 5b f5 ec b3 68 9e f7 d3 3b fe   ..-..o[...h...;.
123 0380  d5 76 c2 e2 9b 78 ee cc 14 58 47 4a 04 5d 65 d9   .v...x...XGJ.]e.
    0390  f8 81 09 9a fb b7 e0 69 04 59 24 e4 91 09 60 91   .......i.Y$...`.
125 03a0  3e 19 3a dc 05 76 89 87 8b 6f fd c8 bd b7 16 12   >.:..v...o......
    03b0  ee 52 35 eb a2 02 45 cb 72 ac 75 96 f8 aa 96 a1   .R5...E.r.u.....
127 03c0  86 f6 22 ac d5 78 bb 66 81 ab 98 f1 b2 f4 8c 70   ..".x.f.......p
    03d0  73 ce e7 86 bd f7 e8 41 8e 3e 6b 63 6b 04 1d 32   s......A.>kck..2
129 03e0  7b 22 5f c9 33 1d 81 48 1d cd d2 99 6c 1c 5b 3a   {"_.3..H....l.[:
    03f0  ea c0 ee 49 f7 ff 87 d9 b2 6a ae 61 18 5e 90 ac   ...I.....j.a.^..
131 0400  0f d1 05 3f e4 f0 c8 6f 2c fa b3 19 3b 0f dd ca   ...?...o,...;...
    0410  c6 49 7d 55 9e 51 02 9c 98 e4 f4 b0 49 a7 41 72   .I}U.Q......I.Ar
133 0420  dc 76 00 af e0 c9 a5 6b 1c 57 24 b3 d2 80 cb 84   .v.....k.W$.....
    0430  b2 a3 83 5a 9f 16 c4 52 68 ae 71 a4 9a 31 eb d7   ...Z...Rh.q..1..
135 0440  7a ab a5 19 ef 71 13 c0 a7 48 db e9 07 99 8a 12   z....q...H......
    0450  d0 54 5f 55 f6 6d d5 77 65 0e a5 37 bc 53 e3 5a   .T_U.m.we..7.S.Z
137 0460  e7 8a 1c a7 e9 0f 25 45 42 91 eb 09 13 f2 fc 04   ......%EB.......
    0470  5f e8 de c8 4a c1 3e e0 26 7f 9d 6b 0f 30 02 14   _...J.>.&..k.0..
139 0480  f2 97 2e b2 87 a4 4d aa 9a 5e 21 33 63 34 9a af   ......M..^!3c4..
    0490  be f9 a8 56 f5 e1 ae ed 90 59 51 35 c0 6f 1c 4d   ...V.....YQ5.o.M
141 04a0  86 5a 41 a9 b4 b5 e8 29 47 10 72 ca d1 1f 12 4b   .ZA....)G.r....K
    04b0  e2 8a 00 70 03 96 57 4a 5f 04 ed ea 75 88 66 aa   ...p..WJ_...u.f.
143 04c0  3b ad 05 19 3b 72 e8 6b 9a b4 21 a6 49 f7 19 6d   ;...;r.k..!.I..m
    04d0  d9 aa 59 44 c0 c6 72 eb 6c 21 bb 85 17 12 da 1a   ..YD..r.l!......
145 04e0  18 c2 0f 1d 86 c6 39 ea 36 ef 9d a5 e0 ec c0 43   ......9.6......C
    04f0  d4 3a e8 8d 24 7d bf b7 9f 89 4a 93 94 d4 ed e3   .:..$}....J.....
147 0500  3a 93 28 a5 5b c7 f3 45 1c f4 26 6f 02 c8 f5 5e   :.(.[..E..&o...^
    0510  32 9e 8c 16 63 9e fb 20 e5 3f 76 ec 5e 8e d3 aa   2...c.. .?v.^...
149 0520  91 f1 b3 e9 dd 1b 84 75 2a 2a 5f 33 da 0b aa d6   .......u**_3....
    0530  ab 4e 25 28 2b e5 33 85 5e 3f 53 ec b5 6b 25 a0   .N%(+.3.^?S..k%.
151 0540  0c d2 ec aa 51 e9 77 9a 1c 45 9a 0b 5b b8 1c 18   ....Q.w..E..[...
    0550  5d 11 cf 35 08 32 a5 68 91 b2 8c 10 a8 86 4f 46   ]..5.2.h......OF
153 0560  d2 5c 2f 48 41 50 57 32 3a 76 b4 84 ff 73 cf d6   .\/HAPW2:v...s..
    0570  2c 3a 3c 67 5f 64 12 b6 c4 2d 18 a1 c1 2f 22 44   ,:<g_d...-.../"D
155 0580  a1 6a d9 8a 1b db ee 10 e7 10 09 66 21 1c 66 75   .j.........f!.fu
    0590  7a 00 73 63 c2 1d 65 34 db be c3 46 b5 8f db b7   z.sc..e4...F....
157 05a0  20 dc 19 22 ec 46 c7 7c 8c b0 c2 6f 3f 06 ef c8    ..".F.|...o?...
    05b0  d9 99 f5 95 1c 59 9c 55 7d 2e 4e 87 2f 46 08 a8   .....Y.U}.N./F..
159 05c0  e3                                              .
        Data: 1472baa5935b4c7b3df37df0ae953907c78076113b82f6e7...
161     [Length: 1473]
```

all fragments after fragmentation:

```
1   No.     Time        Source                Destination           Protocol Length
        Info
        71 8.015694     10.0.1.11             10.0.2.22             IPv4     1514
            Fragmented IP protocol (proto=UDP 17, off=0, ID=07ef) [Reassembled in #72]
3
    Frame 71: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits)
5       Encapsulation type: Ethernet (1)
        Arrival Time: Mar 15, 2016 11:35:52.642694000 CET
7       [Time shift for this packet: 0.000000000 seconds]
        Epoch Time: 1458038152.642694000 seconds
9       [Time delta from previous captured frame: 0.000016000 seconds]
        [Time delta from previous displayed frame: 0.000016000 seconds]
11      [Time since reference or first frame: 8.015694000 seconds]
        Frame Number: 71
13      Frame Length: 1514 bytes (12112 bits)
```

```
    Capture Length: 1514 bytes (12112 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: eth:ip:data]
Ethernet II, Src: IntelCor_1a:7c:70 (68:05:ca:1a:7c:70), Dst: IntelCor_1a:80:18
    (68:05:ca:1a:80:18)
    Destination: IntelCor_1a:80:18 (68:05:ca:1a:80:18)
        Address: IntelCor_1a:80:18 (68:05:ca:1a:80:18)
        .... ..0. .... .... .... .... = LG bit: Globally unique address (factory
            default)
        .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
    Source: IntelCor_1a:7c:70 (68:05:ca:1a:7c:70)
        Address: IntelCor_1a:7c:70 (68:05:ca:1a:7c:70)
        .... ..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: 0x07ef (2031)
    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: 0x3602 [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]
    Reassembled IPv4 in frame: 72
Data (1480 bytes)

0000  98 9c 13 ae 05 c9 46 2e 14 72 ba a5 93 5b 4c 7b   ......F..r...[L{
0010  3d f3 7d f0 ae 95 39 07 c7 80 76 11 3b 82 f6 e7   =.}...9...v.;...
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..
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.............
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..
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
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)
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>.......
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.
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......
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..
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.[....
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: 989c13ae05c9462e1472baa5935b4c7b3df37df0ae953907...
147     [Length: 1480]
```

Determine the values of the fields in the IP header that are used for fragmentation:

Packet 71 (IPv4):
Identification: 2031
Don't Fragment Bit: Not set
More Fragments Bit: Set
Fragment Offset: 0

Packet 72 (UDP):
Identification: 2031
Don't Fragment Bit: Not set
More Fragments Bit: Not set
Fragment Offset: 1480

In packet 71, the More Fragments bit is set to show that this packet belongs to a fragment of a datagram. The Identification is set to be the same as the UDP packet 72 to indicate that packet 71 is a fragment for packet 72. The fragment offset value is used to reassemble the data in the correct order.

**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.
At -l1473, fragmentation starts, and continues to happen until -l65507.
After -l65508, there is no fragmentation, as the packet is not sent at all anymore.
This happens because the limit for the data length for IPv4 is 65507 bytes ( = 64kB - 8 byte UDP header - 20 byte IP header).

In "Lab 5/traces/3.A.PC1.pcap", we captured the transmissions in following order: -l1024, -l1072. -l1073, -l65507, -l65508

### 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.
We do not observe fragmentation, TCP avoids fragmentation by setting the DF bit.

Additionally:
The MSS is negotiated in the three way handshake and determined by the minimum MTU value of the two pcs. We see in this case that the MSS is 460.
This size is respected by all of the packets in the nttcp transmission, so no fragmentation is needed.

**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?
Only when an intermediary hop is causing a “bottleneck” for a packet to be forwarded due to its MTU being exceeded will there be any ICMP error messages: the hosts themselves have knowledge of their own MTUs and based on those, the MSS is negotiated in the three way handshake. We see in this case that the MSS is 460.

The DF-bit is set in the IP datagrams.

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.
We do not observe fragmentation, TCP avoids fragmentation by setting the DF bit.

**Question 3.B.1.5.b)**
If you observe ICMP error messages, describe how they are used for Path MTU Discovery.

The ICMP error messages are used for Path MTU Discovery to reduce the TCP transmission's MSS value and send smaller packets.

We see this in packets 45, 47 and 48 in the file "Lab 5/traces/3.B.PC1.pcap":

- packet 45 is a TCP packet sent by PC1 with a payload of 1024 bytes, which exceeds PC3's MTU.
- PC1 then receives packet 47 from PC3, which is the ICMP error message.
- packet 48 is a TCP packet sent by PC1 with a smaller payload of 500 bytes.

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.

If you observed ICMP error messages, include one such message in the report:

```
No.     Time        Source                Destination           Protocol Length
    Info
     47 537.418549  10.0.1.33             10.0.1.11             ICMP     590
         Destination unreachable (Fragmentation needed)

Frame 47: 590 bytes on wire (4720 bits), 590 bytes captured (4720 bits)
    Encapsulation type: Ethernet (1)
    Arrival Time: Apr 11, 2016 10:13:49.771318000 CEST
    [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: 1460362429.771318000 seconds
    [Time delta from previous captured frame: 0.002373000 seconds]
    [Time delta from previous displayed frame: 0.002373000 seconds]
    [Time since reference or first frame: 537.418549000 seconds]
    Frame Number: 47
    Frame Length: 590 bytes (4720 bits)
    Capture Length: 590 bytes (4720 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: eth:ip:icmp:ip:tcp:data]
    [Coloring Rule Name: ICMP errors]
    [Coloring Rule String: icmp.type eq 3 || icmp.type eq 4 || icmp.type eq 5 ||
        icmp.type eq 11 || icmpv6.type eq 1 || icmpv6.type eq 2 || icmpv6.type eq 3
         || icmpv6.type eq 4]
```

```
Ethernet II, Src: IntelCor_36:39:c7 (68:05:ca:36:39:c7), Dst: IntelCor_36:33:a0
    (68:05:ca:36:33:a0)
    Destination: 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)
    Source: 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)
    Type: IP (0x0800)
Internet Protocol Version 4, Src: 10.0.1.33 (10.0.1.33), Dst: 10.0.1.11 (10.0.1.11)
    Version: 4
    Header length: 20 bytes
    Differentiated Services Field: 0xc0 (DSCP 0x30: Class Selector 6; ECN: 0x00:
        Not-ECT (Not ECN-Capable Transport))
        1100 00.. = Differentiated Services Codepoint: Class Selector 6 (0x30)
        .... ..00 = Explicit Congestion Notification: Not-ECT (Not ECN-Capable
            Transport) (0x00)
    Total Length: 576
    Identification: 0x8814 (34836)
    Flags: 0x00
        0... .... = Reserved bit: Not set
        .0.. .... = Don't fragment: Not set
        ..0. .... = More fragments: Not set
    Fragment offset: 0
    Time to live: 64
    Protocol: ICMP (1)
    Header checksum: 0xd9bd [validation disabled]
        [Good: False]
        [Bad: False]
    Source: 10.0.1.33 (10.0.1.33)
    Destination: 10.0.1.11 (10.0.1.11)
    [Source GeoIP: Unknown]
    [Destination GeoIP: Unknown]
Internet Control Message Protocol
    Type: 3 (Destination unreachable)
    Code: 4 (Fragmentation needed)
    Checksum: 0x164f [correct]
    MTU of next hop: 500
    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: 1076
        Identification: 0x985b (39003)
        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: 0x8748 [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]
```

```
    Transmission Control Protocol, Src Port: 43803 (43803), Dst Port: 5038 (5038),
        Seq: 3445317143, Ack: 468857916
        Source port: 43803 (43803)
        Destination port: 5038 (5038)
        Sequence number: 3445317143
        [Stream index: 3]
        Sequence number: 3445317143    (relative sequence number)
        Acknowledgment number: 468857916    (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
        Window size value: 229
        [Calculated window size: 229]
        [Window size scaling factor: 128]
        Checksum: 0x8f0f [validation disabled]
            [Good Checksum: False]
            [Bad Checksum: False]
        Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
            No-Operation (NOP)
                Type: 1
                    0... .... = Copy on fragmentation: No
                    .00. .... = Class: Control (0)
                    ...0 0001 = Number: No-Operation (NOP) (1)
            No-Operation (NOP)
                Type: 1
                    0... .... = Copy on fragmentation: No
                    .00. .... = Class: Control (0)
                    ...0 0001 = Number: No-Operation (NOP) (1)
            Timestamps: TSval 61392463, TSecr 61391765
                Kind: Timestamp (8)
                Length: 10
                Timestamp value: 61392463
                Timestamp echo reply: 61391765
    Data (496 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   ................
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   ................
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   ................
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   ................
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   ................
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   ................
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   ................
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    ................
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    ................
01a0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
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    ................
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    ................
        Data: 00000000000000000000000000000000000000000000000000...
        [Length: 496]
```

The first TCP segment that is sent after PC1 has received the ICMP error message:

```
No.     Time        Source                Destination           Protocol Length
    Info
     48 537.418590  10.0.1.11             10.0.2.22             TCP      566    [
        TCP Out-Of-Order] 43803 > 5038 [ACK] Seq=1 Ack=1 Win=29312 Len=500 TSval
        =61392464 TSecr=61391765

Frame 48: 566 bytes on wire (4528 bits), 566 bytes captured (4528 bits)
    Encapsulation type: Ethernet (1)
    Arrival Time: Apr 11, 2016 10:13:49.771359000 CEST
    [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: 1460362429.771359000 seconds
    [Time delta from previous captured frame: 0.000041000 seconds]
    [Time delta from previous displayed frame: 0.000041000 seconds]
    [Time since reference or first frame: 537.418590000 seconds]
    Frame Number: 48
    Frame Length: 566 bytes (4528 bits)
    Capture Length: 566 bytes (4528 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: eth:ip:tcp:data]
    [Coloring Rule Name: Bad TCP]
    [Coloring Rule String: tcp.analysis.flags && !tcp.analysis.window_update]
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: 552
    Identification: 0x985d (39005)
    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: 0x8952 [validation disabled]
```

```
46          [Good: False]
            [Bad: False]
48      Source: 10.0.1.11 (10.0.1.11)
        Destination: 10.0.2.22 (10.0.2.22)
50      [Source GeoIP: Unknown]
        [Destination GeoIP: Unknown]
52  Transmission Control Protocol, Src Port: 43803 (43803), Dst Port: 5038 (5038), Seq:
        1, Ack: 1, Len: 500
        Source port: 43803 (43803)
54      Destination port: 5038 (5038)
        [Stream index: 3]
56      Sequence number: 1    (relative sequence number)
        [Next sequence number: 501    (relative sequence number)]
58      Acknowledgment number: 1    (relative ack number)
        Header length: 32 bytes
60      Flags: 0x010 (ACK)
            000. .... .... = Reserved: Not set
62          ...0 .... .... = Nonce: Not set
            .... 0... .... = Congestion Window Reduced (CWR): Not set
64          .... .0.. .... = ECN-Echo: Not set
            .... ..0. .... = Urgent: Not set
66          .... ...1 .... = Acknowledgment: Set
            .... .... 0... = Push: Not set
68          .... .... .0.. = Reset: Not set
            .... .... ..0. = Syn: Not set
70          .... .... ...0 = Fin: Not set
        Window size value: 229
72      [Calculated window size: 29312]
        [Window size scaling factor: 128]
74      Checksum: 0x193b [validation disabled]
            [Good Checksum: False]
76          [Bad Checksum: False]
        Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
78          No-Operation (NOP)
                Type: 1
80                  0... .... = Copy on fragmentation: No
                    .00. .... = Class: Control (0)
82                  ...0 0001 = Number: No-Operation (NOP) (1)
            No-Operation (NOP)
84              Type: 1
                    0... .... = Copy on fragmentation: No
86                  .00. .... = Class: Control (0)
                    ...0 0001 = Number: No-Operation (NOP) (1)
88          Timestamps: TSval 61392464, TSecr 61391765
                Kind: Timestamp (8)
90              Length: 10
                Timestamp value: 61392464
92              Timestamp echo reply: 61391765
        [SEQ/ACK analysis]
94          [Bytes in flight: 2049]
            [TCP Analysis Flags]
96              [This frame is a (suspected) out-of-order segment]
                    [Expert Info (Warn/Sequence): Out-Of-Order segment]
98                      [Message: Out-Of-Order segment]
                        [Severity level: Warn]
100                     [Group: Sequence]
    Data (500 bytes)
102
    0000  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
104 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   ................
106 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   ................
108 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   ................
110 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   ................
```

```
112 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  ................
114 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  ................
116 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  ................
118 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  ................
120 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  ................
122 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  ................
124 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  ................
126 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  ................
128 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  ................
130 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  ................
132 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  ................
134 01f0  00 00 00 00                                      ....
        Data: 0000000000000000000000000000000000000000000000000...
136     [Length: 500]
```

## 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.
The three-way handshake packets are packets no. 1, 2 and 3 in “Lab 5/traces/4.A.PC1.pcap”. The first packet (from client (PC1) to server (PC2)) has the SYN flag set. PC2 recognizes this as the first packet of the three-way handshake. SYN is for synchronizing the sequence numbers. Say PC1 sent sequence number i, then PC2 knows that the next packet from PC1 will have sequence number i + 1. In this case, because the SYN flag is set, even though no application data was sent, the sequence number increases by 1.

The next packet (from server to client) has the SYN and ACK flags set. PC1 interprets this as the second packet in the three-way handshake. Say PC2 sent sequence number j, then PC1 knows the next packet from PC2 will have sequence number j + 1. This packet also acknowledges PC1's initial packet by sending acknowledgement number i + 1 (an ACK always has the sequence number of the first byte the host expects to receive next).
Then PC1 concludes the three-way handshake by sending one more packet with sequence number i + 1 and an acknowledgement of PC2's packet with sequence number j + 1.

```
1 0.000000   10.0.5.11  10.0.5.22  TCP 74   48613âĘŠ23 [SYN] Seq=2495102103 Win=29200
    Len=0 MSS=1460 SACK_PERM=1 TSval=4215509 TSecr=0 WS=128
2 2 0.000621   10.0.5.22  10.0.5.11  TCP 74   23âĘŠ48613 [SYN, ACK] Seq=3213249774 Ack
    =2495102104 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=4214383 TSecr=4215509 WS
    =128
3 0.000660   10.0.5.11  10.0.5.22  TCP 66   48613âĘŠ23 [ACK] Seq=2495102104 Ack
    =3213249775 Win=29312 Len=0 TSval=4215509 TSecr=4214383
```

**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?
PC1 started with sequence number 2495102103 and PC2 started with sequence number 3213249774.

```
1 1 0.000000   10.0.5.11  10.0.5.22  TCP 74   48613âĘŠ23 [SYN] Seq=2495102103 Win=29200
    Len=0 MSS=1460 SACK_PERM=1 TSval=4215509 TSecr=0 WS=128
2 0.000621   10.0.5.22  10.0.5.11  TCP 74   23âĘŠ48613 [SYN, ACK] Seq=3213249774 Ack
    =2495102104 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=4214383 TSecr=4215509 WS
    =128
```

**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?
Packet no. 4, the first packet after the three-way handshake, contains Telnet application data. The sequence number of the first byte from client to server is 2495102104, which is the same sequence number PC1 used in the final message of the three-way handshake.

```
Frame 4: 93 bytes on wire (744 bits), 93 bytes captured (744 bits)
2 Ethernet II, Src: IntelCor_1a:80:15 (68:05:ca:1a:80:15), Dst: IntelCor_1a:7c:75
    (68:05:ca:1a:7c:75)
Internet Protocol Version 4, Src: 10.0.5.11 (10.0.5.11), Dst: 10.0.5.22 (10.0.5.22)
4 Transmission Control Protocol, Src Port: 48613 (48613), Dst Port: 23 (23), Seq:
    2495102104, Ack: 3213249775, Len: 27
    Source Port: 48613 (48613)
6   Destination Port: 23 (23)
    [Stream index: 0]
8   [TCP Segment Len: 27]
    Sequence number: 2495102104
10  [Next sequence number: 2495102131]
    Acknowledgment number: 3213249775
12  Header Length: 32 bytes
    .... 0000 0001 1000 = Flags: 0x018 (PSH, ACK)
14  Window size value: 229
    [Calculated window size: 29312]
16  [Window size scaling factor: 128]
    Checksum: 0x1e62 [validation disabled]
18  Urgent pointer: 0
    Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
20      No-Operation (NOP)
```

```
        No-Operation (NOP)
22      Timestamps: TSval 4215509, TSecr 4214383
    [SEQ/ACK analysis]
24 Telnet
    Do Suppress Go Ahead
26  Will Terminal Type
    Will Negotiate About Window Size
28  Will Terminal Speed
    Will Remote Flow Control
30  Will Linemode
    Will New Environment Option
32  Do Status
    Will X Display Location
```

**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.

In the first two packets of the three-way handshake, the window sizes are rather large: 29200 for PC1 and 28960 for PC2. In subsequent packets, it drops down to about 227-229.

```
1  Frame 1: 74 bytes on wire (592 bits), 74 bytes captured (592 bits)
   Ethernet II, Src: IntelCor_1a:80:15 (68:05:ca:1a:80:15), Dst: IntelCor_1a:7c:75
       (68:05:ca:1a:7c:75)
3  Internet Protocol Version 4, Src: 10.0.5.11 (10.0.5.11), Dst: 10.0.5.22 (10.0.5.22)
   Transmission Control Protocol, Src Port: 48613 (48613), Dst Port: 23 (23), Seq:
       2495102103, Len: 0
5      Source Port: 48613 (48613)
       Destination Port: 23 (23)
7      [Stream index: 0]
       [TCP Segment Len: 0]
9      Sequence number: 2495102103
       Acknowledgment number: 0
11     Header Length: 40 bytes
       .... 0000 0000 0010 = Flags: 0x002 (SYN)
13     Window size value: 29200
       [Calculated window size: 29200]
15     Checksum: 0x1e4f [validation disabled]
       Urgent pointer: 0
17     Options: (20 bytes), Maximum segment size, SACK permitted, Timestamps, No-
           Operation (NOP), Window scale
   Frame 2: 74 bytes on wire (592 bits), 74 bytes captured (592 bits)
19 Ethernet II, Src: IntelCor_1a:7c:75 (68:05:ca:1a:7c:75), Dst: IntelCor_1a:80:15
       (68:05:ca:1a:80:15)
   Internet Protocol Version 4, Src: 10.0.5.22 (10.0.5.22), Dst: 10.0.5.11 (10.0.5.11)
21 Transmission Control Protocol, Src Port: 23 (23), Dst Port: 48613 (48613), Seq:
       3213249774, Ack: 2495102104, Len: 0
       Source Port: 23 (23)
23     Destination Port: 48613 (48613)
       [Stream index: 0]
25     [TCP Segment Len: 0]
       Sequence number: 3213249774
27     Acknowledgment number: 2495102104
       Header Length: 40 bytes
29     .... 0000 0001 0010 = Flags: 0x012 (SYN, ACK)
       Window size value: 28960
31     [Calculated window size: 28960]
       Checksum: 0x7b28 [validation disabled]
33     Urgent pointer: 0
       Options: (20 bytes), Maximum segment size, SACK permitted, Timestamps, No-
           Operation (NOP), Window scale
35     [SEQ/ACK analysis]
   Frame 3: 66 bytes on wire (528 bits), 66 bytes captured (528 bits)
```

```
Ethernet II, Src: IntelCor_1a:80:15 (68:05:ca:1a:80:15), Dst: IntelCor_1a:7c:75
    (68:05:ca:1a:7c:75)
Internet Protocol Version 4, Src: 10.0.5.11 (10.0.5.11), Dst: 10.0.5.22 (10.0.5.22)
Transmission Control Protocol, Src Port: 48613 (48613), Dst Port: 23 (23), Seq:
    2495102104, Ack: 3213249775, Len: 0
    Source Port: 48613 (48613)
    Destination Port: 23 (23)
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence number: 2495102104
    Acknowledgment number: 3213249775
    Header Length: 32 bytes
    .... 0000 0001 0000 = Flags: 0x010 (ACK)
    Window size value: 229
    [Calculated window size: 29312]
    [Window size scaling factor: 128]
    Checksum: 0x1e47 [validation disabled]
    Urgent pointer: 0
    Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
    [SEQ/ACK analysis]
Frame 4: 93 bytes on wire (744 bits), 93 bytes captured (744 bits)
Ethernet II, Src: IntelCor_1a:80:15 (68:05:ca:1a:80:15), Dst: IntelCor_1a:7c:75
    (68:05:ca:1a:7c:75)
Internet Protocol Version 4, Src: 10.0.5.11 (10.0.5.11), Dst: 10.0.5.22 (10.0.5.22)
Transmission Control Protocol, Src Port: 48613 (48613), Dst Port: 23 (23), Seq:
    2495102104, Ack: 3213249775, Len: 27
    Source Port: 48613 (48613)
    Destination Port: 23 (23)
    [Stream index: 0]
    [TCP Segment Len: 27]
    Sequence number: 2495102104
    [Next sequence number: 2495102131]
    Acknowledgment number: 3213249775
    Header Length: 32 bytes
    .... 0000 0001 1000 = Flags: 0x018 (PSH, ACK)
    Window size value: 229
    [Calculated window size: 29312]
    [Window size scaling factor: 128]
    Checksum: 0x1e62 [validation disabled]
    Urgent pointer: 0
    Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
    [SEQ/ACK analysis]
Telnet
```

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

PC1's initial packet includes a MSS of 1460, as does PC2's initial packet. They happen to both send the same values, although both directions could have different values for MSS. Only the packets where SYN is set include values for MSS.

```
Frame 1: 74 bytes on wire (592 bits), 74 bytes captured (592 bits)
Ethernet II, Src: IntelCor_1a:80:15 (68:05:ca:1a:80:15), Dst: IntelCor_1a:7
    c:75 (68:05:ca:1a:7c:75)
Internet Protocol Version 4, Src: 10.0.5.11 (10.0.5.11), Dst: 10.0.5.22
    (10.0.5.22)
Transmission Control Protocol, Src Port: 48613 (48613), Dst Port: 23 (23),
    Seq: 2495102103, Len: 0
    Source Port: 48613 (48613)
    Destination Port: 23 (23)
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence number: 2495102103
    Acknowledgment number: 0
    Header Length: 40 bytes
    .... 0000 0000 0010 = Flags: 0x002 (SYN)
    Window size value: 29200
    [Calculated window size: 29200]
    Checksum: 0x1e4f [validation disabled]
    Urgent pointer: 0
    Options: (20 bytes), Maximum segment size, SACK permitted, Timestamps,
        No-Operation (NOP), Window scale
        Maximum segment size: 1460 bytes
            Kind: Maximum Segment Size (2)
            Length: 4
            MSS Value: 1460
        TCP SACK Permitted Option: True
        Timestamps: TSval 4215509, TSecr 0
        No-Operation (NOP)
        Window scale: 7 (multiply by 128)
            Kind: Window Scale (3)
            Length: 3
            Shift count: 7
            [Multiplier: 128]
Frame 2: 74 bytes on wire (592 bits), 74 bytes captured (592 bits)
Ethernet II, Src: IntelCor_1a:7c:75 (68:05:ca:1a:7c:75), Dst: IntelCor_1a
    :80:15 (68:05:ca:1a:80:15)
Internet Protocol Version 4, Src: 10.0.5.22 (10.0.5.22), Dst: 10.0.5.11
    (10.0.5.11)
Transmission Control Protocol, Src Port: 23 (23), Dst Port: 48613 (48613),
    Seq: 3213249774, Ack: 2495102104, Len: 0
    Source Port: 23 (23)
    Destination Port: 48613 (48613)
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence number: 3213249774
    Acknowledgment number: 2495102104
    Header Length: 40 bytes
    .... 0000 0001 0010 = Flags: 0x012 (SYN, ACK)
    Window size value: 28960
    [Calculated window size: 28960]
    Checksum: 0x7b28 [validation disabled]
    Urgent pointer: 0
    Options: (20 bytes), Maximum segment size, SACK permitted, Timestamps,
        No-Operation (NOP), Window scale
        Maximum segment size: 1460 bytes
            Kind: Maximum Segment Size (2)
            Length: 4
            MSS Value: 1460
        TCP SACK Permitted Option: True
        Timestamps: TSval 4214383, TSecr 4215509
        No-Operation (NOP)
        Window scale: 7 (multiply by 128)
            Kind: Window Scale (3)
            Length: 3
            Shift count: 7
            [Multiplier: 128]
    [SEQ/ACK analysis]
```

**4.A.3.f**) How long does it take to open a TCP connection?
If we consider the connection to be “open” as soon as the first packet with application data is sent, in this case it took 0.843 ms.

```
1 1 0.000000   10.0.5.11 10.0.5.22 TCP 74   48613âĘŠ23 [SYN] Seq=2495102103 Win
    =29200 Len=0 MSS=1460 SACK_PERM=1 TSval=4215509 TSecr=0 WS=128
  2 0.000621   10.0.5.22 10.0.5.11 TCP 74   23âĘŠ48613 [SYN, ACK] Seq
    =3213249774 Ack=2495102104 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval
    =4214383 TSecr=4215509 WS=128
3 3 0.000660   10.0.5.11 10.0.5.22 TCP 66   48613âĘŠ23 [ACK] Seq=2495102104 Ack
    =3213249775 Win=29312 Len=0 TSval=4215509 TSecr=4214383
  4 0.000843   10.0.5.11 10.0.5.22 TELNET  93   Telnet Data ...
```

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.
In “Lab 5/traces/4.A.PC1.pcap” the packets that are involved in closing the TCP connection are packets with no. 76, 77, 78. They both have the FIN flag set. When a sender sets the FIN flag, it indicates that they won’t send any more data. As we see in the trace however, that doesn’t include ACKs, as PC1 (client) sends one last ACK after sending a packet with FIN set, acknowledging the receipt of PC2’s FIN message.

```
76  76.193351 10.0.5.11 10.0.5.22 TCP 66   48613âĘŠ23 [FIN, ACK] Seq=2495102305 Ack
    =3213250523 Win=30336 Len=0 TSval=4234558 TSecr=4216062
2 77  76.194236 10.0.5.22 10.0.5.11 TCP 66   23âĘŠ48613 [FIN, ACK] Seq=3213250523 Ack
    =2495102306 Win=29056 Len=0 TSval=4233431 TSecr=4234558
78  76.194264 10.0.5.11 10.0.5.22 TCP 66   48613âĘŠ23 [ACK] Seq=2495102306 Ack
    =3213250524 Win=30336 Len=0 TSval=4234558 TSecr=4233431
```

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 first packet with the FIN flag set is from PC2 (server), contrary to when the client closes the connection. Also, this time the server sends one last ACK after sending a packet with the FIN flag set, instead of the server.

```
1 348 209.621366  10.0.5.22 10.0.5.11 TCP 66  23âĘŠ48615 [FIN, ACK] Seq=419429485 Ack
      =3956140201 Win=29056 Len=0 TSval=4266788 TSecr=4252924
  349 209.621460  10.0.5.11 10.0.5.22 TCP 66  48615âĘŠ23 [FIN, ACK] Seq=3956140201
      Ack=419429486 Win=29312 Len=0 TSval=4267915 TSecr=4266788
3 350 209.621939  10.0.5.22 10.0.5.11 TCP 66  23âĘŠ48615 [ACK] Seq=419429486 Ack
      =3956140202 Win=29056 Len=0 TSval=4266788 TSecr=4267915
```

**Question 4.A.5.b)**

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

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?
We observed 7 attempts. The first retransmission occurs after a 1 second interval, then 2 seconds, then 4 seconds, then 8 seconds, then 16 seconds, and finally 32 seconds. Every attempt, the waiting time is doubled.

```
1 1 0.000000  10.0.5.11 10.0.5.100  TCP 74  33347âĘŠ23 [SYN] Seq=2556468869 Win=29200
      Len=0 MSS=1460 SACK_PERM=1 TSval=4441845 TSecr=0 WS=128
  2 0.999586  10.0.5.11 10.0.5.100  TCP 74  [TCP Retransmission] 33347âĘŠ23 [SYN] Seq
      =2556468869 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=4442095 TSecr=0 WS=128
3 3 3.003578  10.0.5.11 10.0.5.100  TCP 74  [TCP Retransmission] 33347âĘŠ23 [SYN] Seq
      =2556468869 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=4442596 TSecr=0 WS=128
  4 7.011578  10.0.5.11 10.0.5.100  TCP 74  [TCP Retransmission] 33347âĘŠ23 [SYN] Seq
      =2556468869 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=4443598 TSecr=0 WS=128
5 5 15.019585 10.0.5.11 10.0.5.100  TCP 74  [TCP Retransmission] 33347âĘŠ23 [SYN] Seq
      =2556468869 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=4445600 TSecr=0 WS=128
  6 31.051586 10.0.5.11 10.0.5.100  TCP 74  [TCP Retransmission] 33347âĘŠ23 [SYN] Seq
      =2556468869 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=4449608 TSecr=0 WS=128
7 7 63.147583 10.0.5.11 10.0.5.100  TCP 74  [TCP Retransmission] 33347âĘŠ23 [SYN] Seq
      =2556468869 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=4457632 TSecr=0 WS=128
```

**Question 4.B.3.b)**

Does the TCP client terminate or reset the connection, when it gives up with trying to establish a connection?
There are only retransmissions of the first packet in the trace. Since there was never a reply, no connection was made, so there is no need to terminate or reset anything.

**Question 4.B.3.c)**

Why does this experiment require to set a static ARP table entry?
If no ARP entry was set for the non-existing IP, an ARP request would first be sent out. The ARP request would fail, since no-one has that IP, and consequently no TCP packets would be sent out.

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?
A TCP packet with [RST,ACK] is returned by the receiving server. This means that the port is closed.

```
1 1 0.000000  10.0.5.11  10.0.5.22  TCP 74  49306âĘŠ80 [SYN] Seq=4067677396 Win=29200
    Len=0 MSS=1460 SACK_PERM=1 TSval=4578023 TSecr=0 WS=128
  2 0.000546  10.0.5.22  10.0.5.11  TCP 60  80âĘŠ49306 [RST, ACK] Seq=0 Ack=4067677397
    Win=0 Len=0
```

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 **??**. 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).

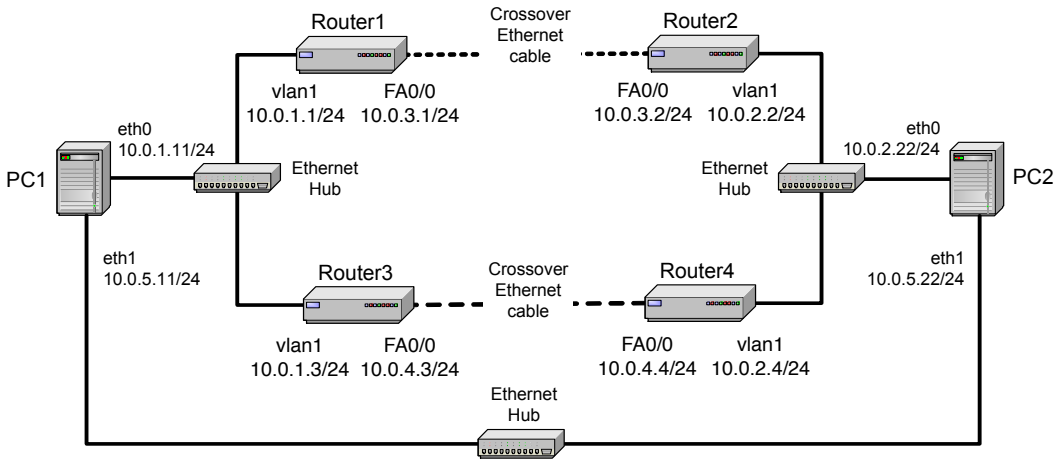


Figure 5.2: Network Topology for Parts 1-4.

| 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 **??**. *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 **??** and **??**.

   - 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.
3 packets per keystroke.

**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?

**Question 5.B.1.c)**
What is the time delay associated with the transmission of ACKs from the Telnet server on PC3?
In 5.B.pcap packet nr. 586, we see that the full contents of a 'paste' are sent within /textbf1 telnet packet.

**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.
*** TODO ***

**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?
*** TODO ***

**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?
*** TODO ***

**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?
*** TODO ***

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.

*** TODO ***

**Question 5.C.1.b)**
Do you observe delayed acknowledgements? Why is the outcome expected?
*** TODO ***

**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.
*** TODO ***

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.
*** TODO ***

## 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 **??**.

### 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?

**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?

**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?

**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?

**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.

**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?

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?

**Question 6.B.b)**
Does the frequency of ACKs change?

**Question 6.B.c)**
Is the range of window sizes advertised by the receiver different from those in 6-a?

**Question 6.B.d)**
Does the TCP sender generally transmit the maximum amount of data allowed by the advertised window? Explain your answer.

### 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.
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.

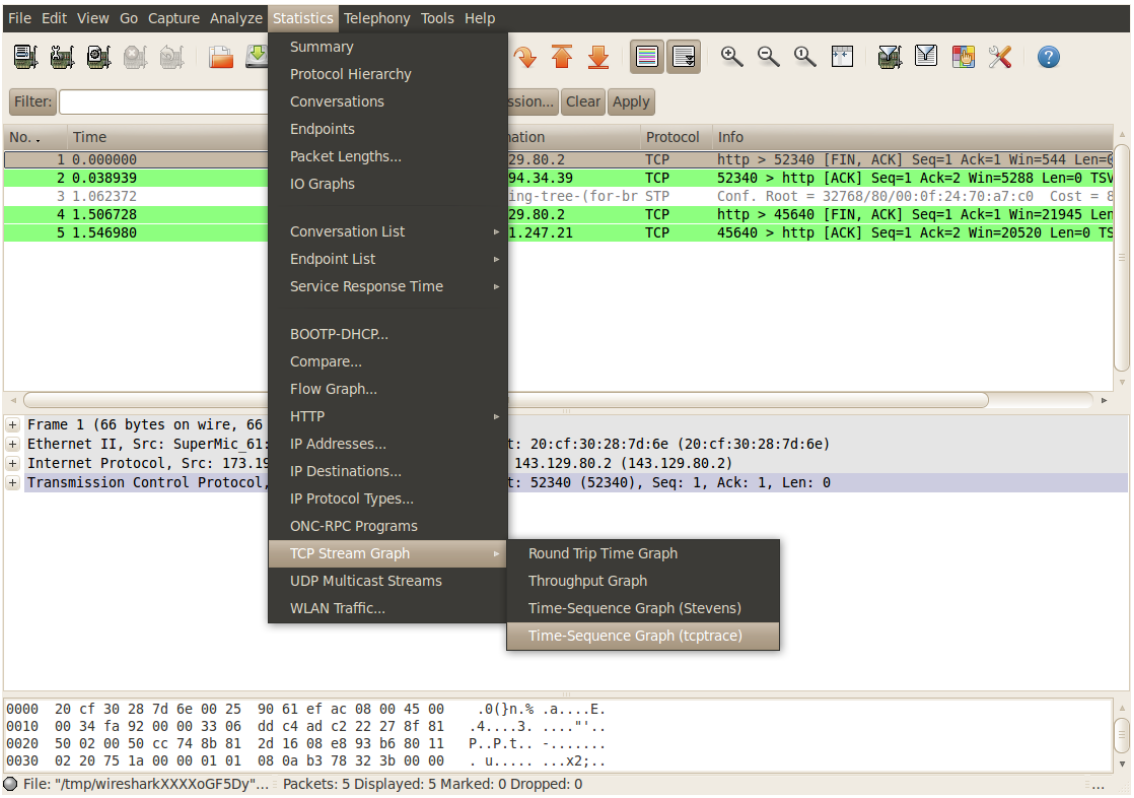


Figure 5.3: Selecting the type of graph for a TCP connection.

**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:

| | |
|---|---|
| **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:**

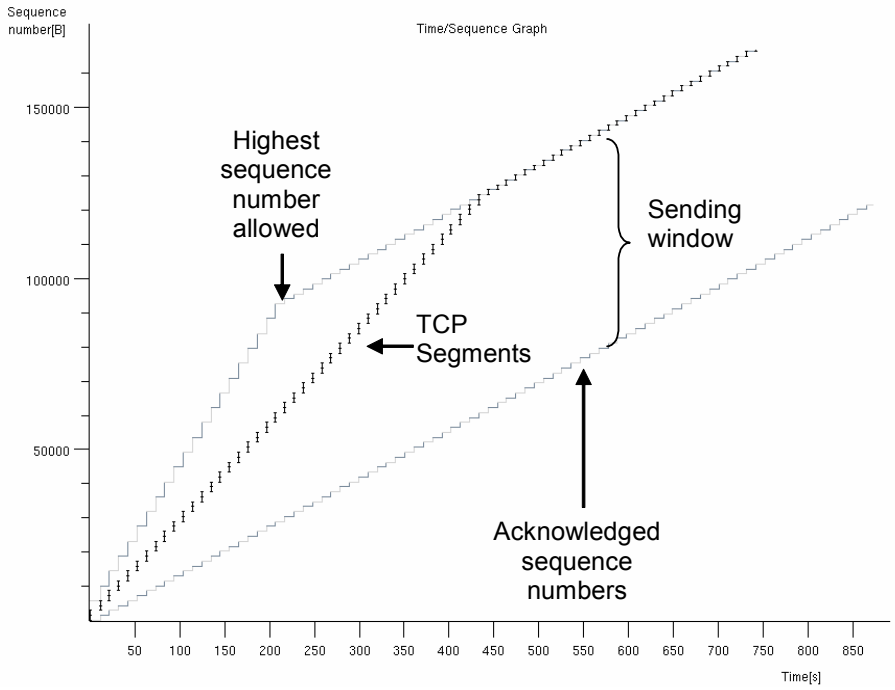


Figure 5.4: Time-Sequence Graph (tcptrace).

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, 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 Exercise 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.

## 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 **??**.

### 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.

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?

**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?

**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.

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?

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.

### 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?

**Question 7.B.6.b)**

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

**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?

**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?

## 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 **??**. 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 **??**. 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.

**Question 8.B.5.b)**

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

**Question 8.B.5.c)**

Can you find occurrences of fast recovery?