KTH Electrical Engineering

# EP2120 Internetworking

## Lab 2

## Transmission Control Protocol

September 2023

Division of Network and Systems Engineering
School of Electrical Engineering and Computer Science
KTH Royal Institute of Technology
Stockholm, Sweden

# Contents

**IMPORTANT:**
- Before coming to the lab session you MUST read Sections 1-3 of this lab manual.
- Before coming to the lab session you MUST do the preparation step in Section 4.1.
- Note that your lab report needs to cover only Sections 5 to 10.

**Throughout the lab:**
- **OPTIONAL:** means this step is not mandatory to perform during the lab.
- **AFTER THE LAB:** means this step should be done after finishing the lab session.

# 1 Introduction

This syllabus contains information about Lab 2. The topic of Lab 2 is the transmission control protocol (TCP). Please read the syllabus carefully.

## 1.1 Intended Learning Outcomes of the Lab

The learning outcomes of this lab are the following:

- Know how to configure simple network devices so as to enable host-to-host data delivery
- You would get an understanding of the difference between UDP and TCP data transmission.
- Understand the effect of fragmentation on TCP.
- Understand TCP connection establishment and termination.
- Understand TCP congestion and flow control.
- Understand the difference between TCP bulk and interactive data transfer.

## 1.2 Requirements and reporting

In order to pass the lab, you need to fulfill the following:

- Pass the preparation quiz on Canvas before the lab starts. The preparation quiz is a prerequisite for performing the lab.
- Perform the lab. You need to be present during the lab session.
- Write and submit one lab report per group. The report should be submitted no later than one week after the lab.
- For completing the report you will need to analyze output data and graphs from Wireshark, produced during the lab session.

### 1.2.1 Lab report

To pass the lab, one lab report shall be written per group. You will probably not have the time to write the report during the lab. Instead, save the logs from `Wireshark` and complete the report after the lab. The lab report shall be written in English and contain the following:

- Names and e-mail addresses of each participant.
- Answers to the questions in the exercises in Sections 5-10.
- `Wireshark` output from the exercises in Sections 5-10, including the data and the graphs. It is explicitly stated in the exercises what data you should save.

The Appendix gives an outline of the lab report. Each group shall submit one lab report via Canvas as a group assignment. Depending on the quality of the lab report, further revision might be required. The Canvas groups for labs will be created by the lab assistants during the lab sessions.

# 2 Preparation for the lab

In order to prepare for the lab, read the items in the reading list and answer the questions below. The lab is limited in time, so it is necessary that the preparations are made in advance to the scheduled lab. The schedule is tight.

## 2.1 Software tools

Please make sure that the TCP congestion control algorithm running on your computer is Reno (instead of cubic). To check the congestion control algorithm running on your computer, please run the following command.

```
sudo sysctl net.ipv4.tcp_congestion_control
```

If your computer is not running Reno as the congestion control algorithm, you can try to switch to Reno by running the following command as the root user.

```
echo reno > /proc/sys/net/ipv4/tcp_congestion_control
```

The following software tools will be used in the lab.

- `ttcp` - Test TCP and UDP performance.
- `tc` - show/manipulate traffic control settings
- `telnet` - An application and a protocol to communicate with another host using TCP.
- `slattach` - Attach a network interface to a serial line using SLIP.
- `ifconfig` - Used to configure a network Interface.
- `route` - Used to manipulate the routing table.

- `import` - Tool to dump a window to a bitmap.
- `wireshark` - An interactive and graphical network traffic analyzer. Read more about Wireshark in Section 2.3.

Manual pages of these tools are installed on all Linux computers (man command) and they can also be found on the web (for example http://linux.die.net/). Read the manual pages of these commands. It is even better if you install the above software on your own computer and learn to use them before the lab. More information about Linux traffic control can be found at http://lartc.org/howto.

## 2.2 Reading list

Forouzan, "TCP/IP Protocol Suite",
4th edition, Chapter 7.3, 8, 11, 12, 13, 14, 15.

- Fragmentation
- UDP
- TCP
- EP2120/IK2218 Lecture Notes: Transmission Control Protocol

Put emphasis on the sections about TCP error and congestion control.

## 2.3 Wireshark

`Wireshark` is a traffic analyzer with a graphical interface. It is the main tool used in this lab. In this section, you find the short description of most of the `Wireshark` functions that are necessary for the lab.

Figure 1 shows the main `Wireshark` window as seen after capturing traffic. The window consists of three frames. A list of the captured packets is shown in the top frame. When a packet in the list is marked, the content is shown in the middle and the lower frames. The middle frame shows the packet in symbolic form, while the lower frame shows the packet in raw, hexadecimal format. In the middle frame, the fields may be expanded to show a more detailed view.

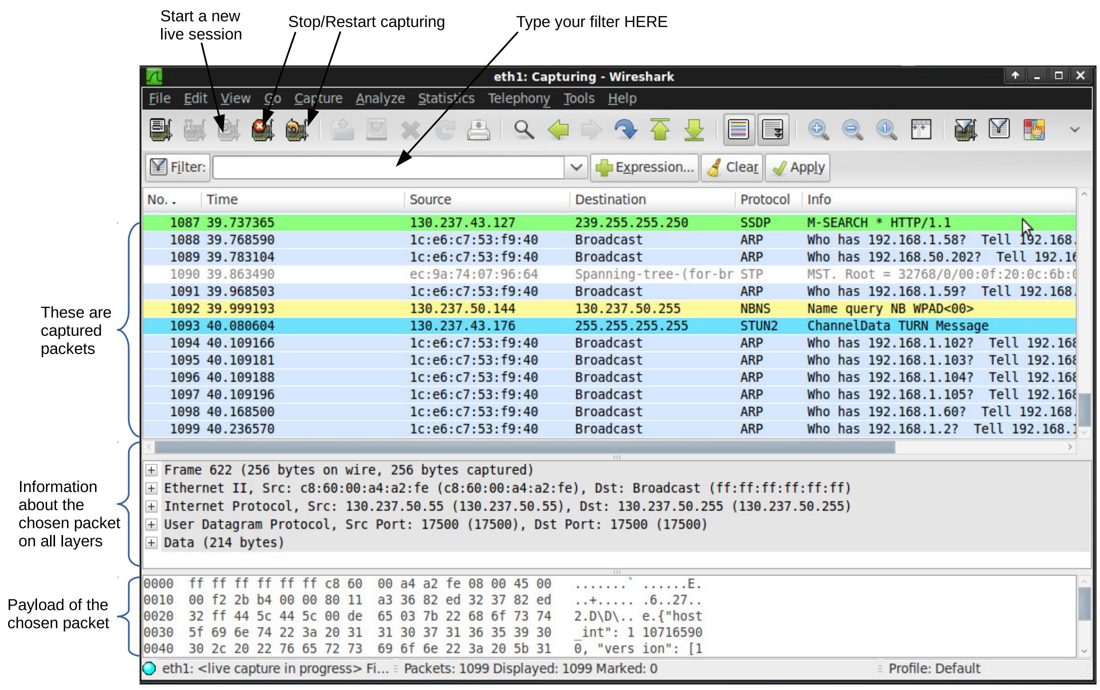
A session is started by choosing *Capture»Start*. This brings up a capture options window. Typically, the display options “Update list of packets in real time” and “Automatic Scrolling in live capture” should be selected. When capturing is started, a capture window is shown and the captured packets are shown in the top frame. To stop capturing, click on the "stop" button in the capture window.

Some statistics can be seen by choosing *Tools»Summary*.

### 2.3.1 Saving `Wireshark` output

Data can be saved as follows:

- Stop capturing the packets.


Start a new live session
Stop/Restart capturing
Type your filter HERE
eth1: Capturing - Wireshark
File Edit View Go Capture Analyze Statistics Telephony Tools Help
Filter:
Expression...
Clear
Apply
These are captured packets
Information about the chosen packet on all layers
Payload of the chosen packet

```
No. .  Time        Source              Destination         Protocol Info
 1087 39.737365  130.237.43.127      239.255.255.250     SSDP     M-SEARCH * HTTP/1.1
 1088 39.768590  1c:e6:c7:53:f9:40   Broadcast           ARP      Who has 192.168.1.58?  Tell 192.168.
 1089 39.783104  1c:e6:c7:53:f9:40   Broadcast           ARP      Who has 192.168.50.202?  Tell 192.16
 1090 39.863490  ec:9a:74:07:96:64   Spanning-tree-(for-br STP    MST. Root = 32768/0/00:0f:20:0c:6b:
 1091 39.968503  1c:e6:c7:53:f9:40   Broadcast           ARP      Who has 192.168.1.59?  Tell 192.168.
 1092 39.999193  130.237.50.144      130.237.50.255      NBNS     Name query NB WPAD<00>
 1093 40.080604  130.237.43.176      255.255.255.255     STUN2    ChannelData TURN Message
 1094 40.109166  1c:e6:c7:53:f9:40   Broadcast           ARP      Who has 192.168.1.102?  Tell 192.168
 1095 40.109181  1c:e6:c7:53:f9:40   Broadcast           ARP      Who has 192.168.1.103?  Tell 192.168
 1096 40.109188  1c:e6:c7:53:f9:40   Broadcast           ARP      Who has 192.168.1.104?  Tell 192.168
 1097 40.109196  1c:e6:c7:53:f9:40   Broadcast           ARP      Who has 192.168.1.105?  Tell 192.168
 1098 40.168500  1c:e6:c7:53:f9:40   Broadcast           ARP      Who has 192.168.1.60?  Tell 192.168.
 1099 40.236570  1c:e6:c7:53:f9:40   Broadcast           ARP      Who has 192.168.1.2?  Tell 192.168.1

+ Frame 622 (256 bytes on wire, 256 bytes captured)
+ Ethernet II, Src: c8:60:00:a4:a2:fe (c8:60:00:a4:a2:fe), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
+ Internet Protocol, Src: 130.237.50.55 (130.237.50.55), Dst: 130.237.50.255 (130.237.50.255)
+ User Datagram Protocol, Src Port: 17500 (17500), Dst Port: 17500 (17500)
+ Data (214 bytes)

0000  ff ff ff ff ff ff c8 60  00 a4 a2 fe 08 00 45 00   .......` ......E.
0010  00 f2 2b b4 00 00 80 11  a3 36 82 ed 32 37 82 ed   ..+..... .6..27..
0020  32 ff 44 5c 44 5c 00 de  65 03 7b 22 68 6f 73 74   2.D\D\.. e.{"host
0030  5f 69 6e 74 22 3a 20 31  31 30 37 31 36 35 39 30   _int": 1 10716590
0040  30 2c 20 22 76 65 72 73  69 6f 6e 22 3a 20 5b 31   0, "vers ion": [1

eth1: <live capture in progress> Fi...  Packets: 1099 Displayed: 1099 Marked: 0      Profile: Default
```

Figure 1: Main `Wireshark` window containing three frames.

- Click the *File»Save As* on the menu bar.
- Choose the *Wireshark/tcpdump/...-pcap* format.
- Name the file.
- Click the *Save* button.

### 2.3.2 Graphs

Figure 2 illustrates a TCP trace graph. After a TCP experiment, this graph is shown by choosing *Statistics» TCP Stream Graph»tcptrace* on the menu bar. The figure shows the time in seconds on the x-axis, and the sequence number on the y-axis. The plots illustrate the segments, and the advertised window.

Unfortunately, `Wireshark` does not allow you to save the graphs to a file. However, by using the import utility, the graph can be dumped to a bitmap. In order to dump the graph to a bitmap do the following. Type

```
# import graph.jpg
```

in a terminal window, then click on the window containing the TCP graph. This saves the graph to a file with the corresponding image format.

`Wireshark` can also be used to create other useful graphs, including the throughput plotted as a function of the time.

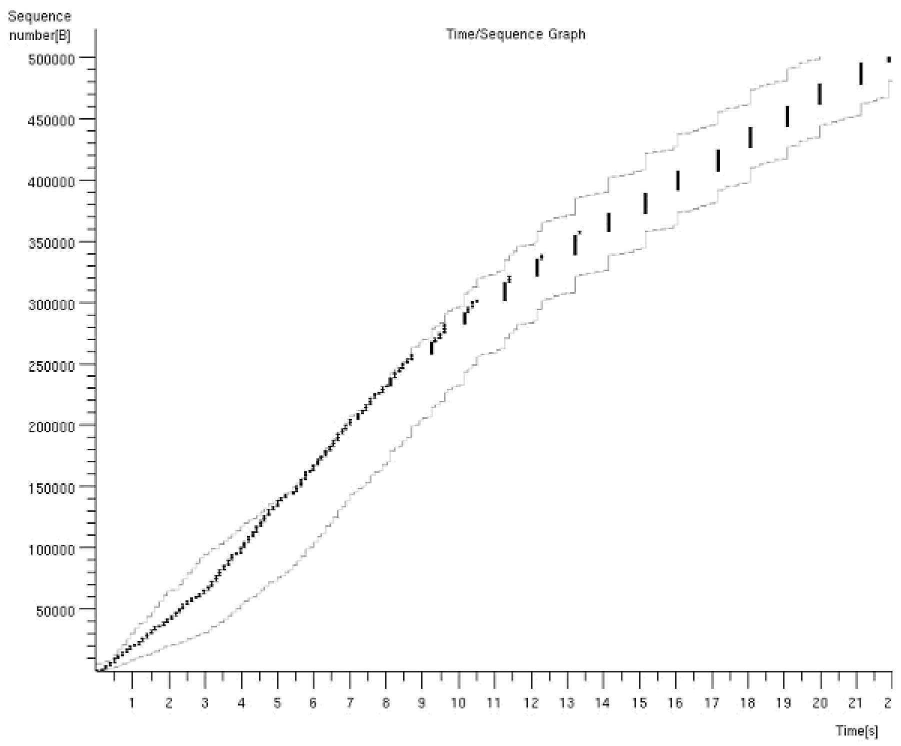


Figure 2: Example of a tcptrace graph.

## 3 Lab equipment

Lab groups consist of 2-4 people. Every group will be provided with the following equipment:

1. Two Microstar routers with four Ethernet interfaces and one serial interface each running Linux (RedHat 9).
2. One cross-wired serial cable (null-modem).
3. Four Ethernet cables.

Every group will also need two end-hosts equipped with one Ethernet interface each, running Linux and the software listed in Section 2.1. The choice of two end-hosts that you will use remains the same as in the first lab exercise.

## 4 Lab setup

### 4.1 Preparation

Consider the network topology in Figure 3, with the serial cable between 10.0.4.2 and 10.0.4.1 disconnected. Write down the forwarding table (i.e., the destination address, subnet mask, and the next hop or interface) of each device to make sure a datagram can be sent between any two interfaces.

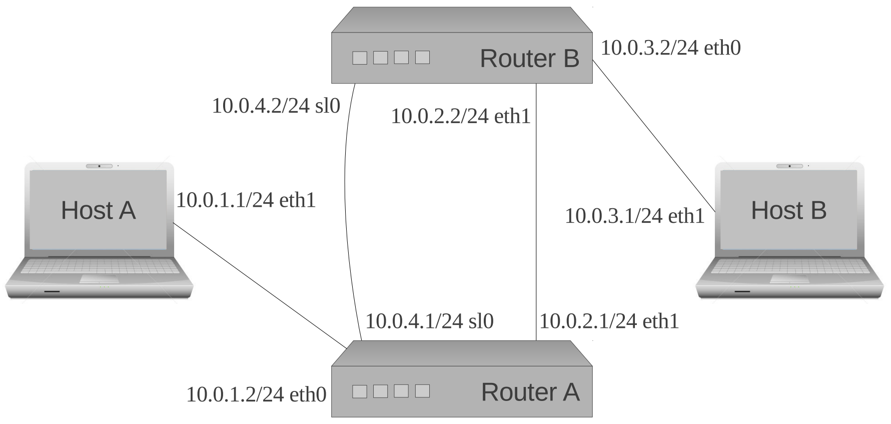


Figure 3: Two hosts, A and B, interconnected by routers A and B. Router A and Router B are connected by an Ethernet cable and a serial cable (cross-connect)

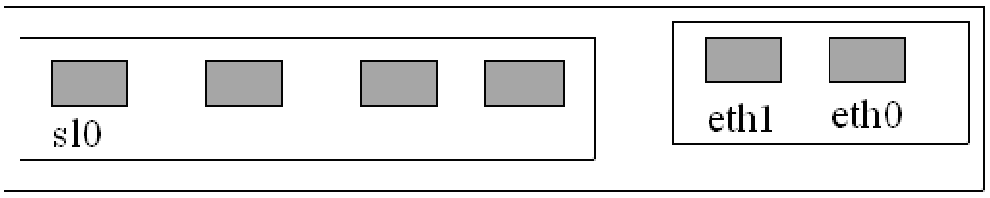


Figure 4: Illustration of the port numbering on a router. This is the rear view of the router.

## 4.2 Connecting the network

In the lab set up, hosts A and B are connected to routers A and B respectively, according to Figure 3. The two routers are connected with one Ethernet cable and with one serial cable. Use the interfaces as specified in Figure 3. The interface numbering of the routers is depicted in Figure 4.

## 4.3 Configuring the hosts

Turn on the hosts. The hosts will boot Linux and start a window system. Once the system boots you will be automatically logged in as:

```
Username:  user
password:  1234
```

As stated in the manual of Lab 1, in the Ubuntu distributions the user is strongly encouraged to login and work as a simple user and use `root` privileges only when necessary via the `sudo` command. However, in this lab since we are going to mainly use system commands, the constant use of `sudo` might become annoying. Therefore, during this lab session, starting a shell as `root` is recommended. You can do so by typing

Please ask one of the lab assistants to check your solution before you continue the lab.

```
sudo -s
```

- Open up a terminal window (right click on the desktop and choose "New Terminal"). To make it easy to remember which host you are working on, set the prompt to reflect the name of the host according to Figure 3.
  ```
  root@live:~# cat >> /root/.bashrc
  export PS1=’’Host A# ‘‘
  ctrl-D
  root@live:~# source /root/.bashrc
  Host A#
  ```

- The next step is to configure the Ethernet interfaces. The routers have the IP address 10.0.0.1/24 pre-configured on eth0. Set the IP address of the Ethernet interface of the host so that you can communicate with the router via `ssh`.

- The last step is to ensure that the host will not make DNS lookups. A straightforward way to do this is to remove the /etc/resolv.conf file.

- Repeat the host configuration procedure for Host B.

## 4.4 Configuring the routers

Upon booting the routers up, the routers will display an error message on their screens. In order to initiate each router properly, you should connect a keyboard to the USB port of the router, and then press `F1`. The error message should then disappear. The Ethernet interfaces for router A and router B are already configured. After connecting the cables you should be able to login to router A from host A.

- Start a new terminal window and use `ssh` to connect to the router. Remove the /root/.ssh/known_hosts file first if it exists.
  ```
  Host A# rm /root/.ssh/known_hosts
  Host A# ssh 10.0.0.1
  Router A# password for root:  qwerty
  ```

- Log in to router B from host B in the same way.

- After the login, you should change the IP address of the eth0 interface of the routers according to Figure 3. Note that after changing the IP addresses of the routers’ eth0 interfaces you must change the IP addresses of your hosts, so that they match the new networks.

- Add the router’s new IP addresses as the default gateway on the hosts.

- Use `ssh` again to connect to the routers.
- In the routers, change directory to /home/ep2120_lab2. Create the directory if it does not exist.
- Add the routes to the 10.0.3.0/24 network for router A and the routes to the 10.0.1.0/24 network for router B.
- Make the router start forwarding traffic by writing the following line at the routers:

  ```
  Router A# echo "1" > /proc/sys/net/ipv4/ip_forward
  Router B# echo "1" > /proc/sys/net/ipv4/ip_forward
  ```
- Verify that the setup is correct by pinging Host B from Host A.
- After you verified the connectivity, attach the serial line and assign IP addresses to the serial interfaces. The name of the serial line interface is sl0.

  ```
  Router A# slattach -p slip -s 4800 /dev/ttyS0 &
  Router B# slattach -p slip -s 4800 /dev/ttyS0 &
  Router A# ifconfig sl0 10.0.4.1 pointopoint 10.0.4.2 up mtu 600
  Router B# ifconfig sl0 10.0.4.2 pointopoint 10.0.4.1 up mtu 600
  ```
  Verify that you can ping from 10.0.4.1 on router A to 10.0.4.2 on router B.
- At this point, the traffic between the two hosts goes over the fast Ethernet link. You can make the traffic go through the serial line by updating the routes on Router A and Router B respectively. Do not change the routes at this step.
- The `arp` tables at the hosts and the routers will timeout every minute, this may affect your measurements. So for all the routers and all the hosts, do the following:

  ```
  # echo 30000 > /proc/sys/net/ipv4/neigh/default/gc_interval
  ```

You have now successfully set up the measurement environment.

Please ask one of the lab assistants to check your progress before you continue the lab.

## 5 Measuring TCP with `ttcp` and `Wireshark`

This exercise is an introduction to measuring TCP traffic. You will use `ttcp` to send TCP traffic from host A to host B, and you will use `Wireshark` to capture the traffic on Host B. This exercise is very similar to the exercise that you performed during Lab 1 for UDP. Note however that `ttcp` does not send any extra packets when running TCP.

1. On Host B, start `Wireshark` to measure traffic on interface eth0, add a filter so it will only measure the traffic from and to host 10.0.1.1.

2. On Host B, start capturing packets with `Wireshark`. (Choose *Capture»Start* on the menu bar to start capturing the traffic (see Section 2.3).)

3. On Host B, start a `ttcp` receiver to receive TCP traffic at port number 1234.

4. On Host A, start a `ttcp` sender to send 10 packets with the length of 1000 bytes using TCP to Host B, at port number 1234.

5. When the transfer is completed, stop capturing. You should now have a packet trace in the main window. Make sure that you recognize the Ethernet header fields, the IP header fields, and the TCP header fields in the mid section. Furthermore, make sure that the number of total transmitted data bytes observed in `Wireshark` matches the expected value of 10,000.

6. Save the output to file output_5. (Section 2.3 shows in detail how to do it.)

7. **AFTER THE LAB:** Observe the traffic and answer the following questions:

   (a) How many packets are transmitted in total (count both directions)?

   (b) What is the range of the sequence numbers used by the sender (Host A)?

   (c) How many packets do not carry a data payload?

   (d) What is the total number of bytes transmitted in the recorded transfer? From those, calculate the amount of user data that was transmitted?

   (e) Compare the total amount of data transmitted in the TCP data transfer to that of a UDP data transfer. Which of the protocols is more efficient in terms of overhead? What is the efficiency in percentage for these two protocols? (Recall the UDP measurements from the previous lab. How many bytes were sent in total using UDP?)
   Hint: you can calculate the efficiency as $\frac{\text{User Data Bytes}}{\text{Total transmitted bytes}}$

**Lab report:** Use the captured data to answer the questions above. Support your answers with the saved `Wireshark` data. Upload the file output_5 together with the report.

## 6 TCP connection management

This exercise gives an insight in TCP connection establishment and termination. In this exercise we use the `telnet` application to establish and terminate TCP connections.

### 6.1 Connection establishment and termination

1. Start capturing packets on Host B with `Wireshark`.

2. Establish a `telnet` connection from Host A to Host B.

3. On Host A, terminate the connection. Type `ctrl-] (ctrl+AltGr+9)` at the `telnet` prompt and then type ``quit'', or instead type `ctrl-d`.

4. Stop the `Wireshark` capturing.

5. Save the `Wireshark` output to the file output_6_1.

6. **AFTER THE LAB:** Study the list of captured packets. Observe the TCP connection establishment and answer the following questions:

   (a) Which packets constitute the three-way handshake? Which flags are set in the headers of these packets?

   (b) What are the initial sequence numbers used by the client and the server, respectively?

   (c) Which packet contains the first application data?

   (d) What are the initial window sizes for the client and for the server?

   (e) How long does it roughly take to open the TCP connection?

7. **AFTER THE LAB:** Study the list of captured packets. Observe the TCP connection termination and answer the following questions.

   (a) Which packets are involved in closing the connection?

   (b) Which flags are set in these packets?

**Lab report:** Use the captured data to answer the questions above. Support your answers with the saved `Wireshark` data. Upload the file output_6_1 together with the report.

## 6.2 Connecting to a non-existing port

In the following exercise you will see what happens when you try to establish a TCP connection to a non-existing port. This could be the case when you try to load a home page from a host that does not have a web server.

1. Start capturing packets with `Wireshark` on Host B.

2. Try to make a `telnet` connection from Host A to Host B with a port number other than the default port for `telnet`, i.e., `23`.

3. Stop capturing packets.

4. Save the `Wireshark` output to file output_6_2.

5. **AFTER THE LAB:** Study the captured list of packets and observe the TCP segments that are transmitted. Answer the following questions.

   (a) How does the server host (Host B) close the connection?

   (b) How long does the process of ending the connection take?

**Lab report:** Use the captured data to answer the questions above. Support your answers with the saved `Wireshark` data. Upload the file output_6_2 together with the report.

### 6.3 Connecting to a non-existing host

In the following exercise you will see what happens when you try to establish a TCP connection to a non-existing host. This could be the case when you try to load a home page from a host given with its IP address that does not exist.

1. On Host A, start capturing packets with `Wireshark`.
2. Set a static `ARP` entry to a non-existent host. Without this fix, no TCP packets will be sent. Instead, Host A would start sending ARP requests and receive no answers.
   ```
   Host A# arp -s 10.0.1.42 1:2:3:4:5:6
   ```
3. Try to make a `telnet` connection to host with the above IP address from host A.
4. Wait for until it terminates and then stop capturing packets and save the `Wireshark` output to file output_6_3.
5. **AFTER THE LAB:** Study the captured packets and observe the TCP segments that are transmitted. Answer the following questions.
   (a) How often does the client try to open a connection? Note the time interval between attempts.
   (b) Does the client stop trying to connect at some point? If so, after how many attempts?

**Lab report:** Use the captured data to answer the questions above. Support your answers with the saved `Wireshark` data. Upload the file output_6_3 together with the report.

## 7 Fragmentation in TCP

In this exercise you will observe the effects of fragmentation in TCP. Note that it is common to make a mistake in this exercise. Please show the output you get to a lab assistant in order to ensure that you have the correct results.

1. Start capturing packets with `Wireshark` on Host B. For this exercise it is convenient to set the Display options "Update list of packets in real time" and "Automatic scrolling in live capture".
2. Start `Wireshark` on Host A as well
   ```
   Host A# Wireshark -ni eth0 -f "host 10.0.3.1 or icmp" &
   ```
3. Change the MTU of eth1 on Router A to 600.
4. Repeat the `ttcp` measurement from before.
   ```
   Host B# ttcp -rs -p1234
   Host A# ttcp -ts -l1000 -n10 -p 1234 10.0.3.1
   ```

5. When the transfer is completed stop capturing packets on Host A and Host B.

6. **AFTER THE LAB:** Observe the traffic and answer the following questions.

   (a) How many packets did Host A measure and how many packets did Host B measure? Why?
   (Notice that there are differences between the two `Wireshark` measurements!)

   (b) Is the DF flag set in the datagrams? Why?

   (c) Do you observe fragmentation? If so, where does it occur?

   (d) Study the ICMP messages recorded at Host A.
   Which node is the source?
   What is the type and the code of the messages?

7. Reset the MTU of eth1 on router A to 1500.

8. You must also update Host A to use the larger MTU again
```
Host A# route del default
Host A# route add default gw 10.0.1.2
```

9. Save the `Wireshark` output to files output_7_A and output_7_B.

**Lab report:** Use the captured data to answer the questions above. Support your answers with the saved `Wireshark` data. Upload the files output_7_A and output_7_B together with the report.

## 8 TCP data transfer

In this exercise you will study TCP flow control and some properties of TCP data transfer. TCP consists of several heuristics to cope with different network and traffic conditions. In particular, TCP has different behaviour for interactive applications and for bulk transfer. TCP also has different behaviour on a fast link and on a slow link.

The interactive application in this exercise is `telnet`, the bulk data transfer is done with `ttcp`. The fast link is an Ethernet link, while the slow link is a SLIP serial link (see Table 8).

| | Interactive application | Bulk data transfer |
|---|---|---|
| Fast link | Telnet over Ethernet | Ttcp over Ethernet |
| Slow link | Telnet over Serial | Ttcp over Serial |

Table 1: TCP data transfer classification.

## 8.1 Interactive application - fast link

1. Establish a `telnet` connection from Host A to Host B on the fast link. Log in with as user *user* with the credentials mentioned above.

2. Start capturing packets with `Wireshark` on Host A.

3. Type a few characters slowly in the `telnet` application. The `telnet` client (Host A) sends each character in a separate TCP segment to the server (Host B) which in turn echoes the character back to the client. Including echoes, we would therefore expect to see four TCP segments for each typed character, instead you should observe three packets per typed character in `wireshark`.

   **AFTER THE LAB:**

   (a) Describe the payload of each packet.
   (b) Explain why you do not see four packets per typed character.
   (c) When the client receives the echo, it waits a certain time before sending the ACK. Why? How long is the delay?
   (d) In the segments that carry characters, what window size is advertised by the `telnet` client and by the server? Does the window size vary as the connection proceeds?

4. Type quickly a lot of characters in the `telnet` application, such as by pressing a key continuously.

   (a) **AFTER THE LAB:** Do you observe a difference in the transmission of segment payloads and ACKs?

5. Stop the capturing of packets.

6. Save the `Wireshark` output to file output_8_1.

**Lab report:** Use the captured data to answer the questions above. Support your answers with the saved `Wireshark` data. Upload the file output_8_1 together with the report.

## 8.2 Bulk transfer - fast link

In this exercise, the behavior of TCP is examined when large amounts of data are transmitted. TCP uses the acknowledgements to limit the sending rate; this together with the receiver window size is the basis of flow control.

1. Start capturing packets with `Wireshark` on Host A.

2. Start a `ttcp` receiver on B and a sender on A, send 1000 packets of length 1000 bytes each.

3. Stop capturing packets.

4. Draw a tcptrace graph. Study the graph carefully.

5. **AFTER THE LAB:** Observe the sliding window protocol from the output of `Wireshark`. The sender transmits data up to the window size of the receiver. Answer the following questions.

   (a) How often does the receiver send ACKs? Can you see a rule on how TCP sends ACKs?

   (b) How many bytes of data does a receiver acknowledge in a typical ACK?

   (c) How does the window size vary during the session?

   (d) Select any ACK packet in the `Wireshark` trace and note its acknowledgement number. Find the original segment in the `Wireshark` output. How long did it take from the transmission until it was ACKed?

   (e) Does the TCP sender generally transmit the maximum number of bytes as allowed by the receiver?

6. Save the tcptrace graph to file output_8_2_tcptrace.jpg and save the `Wireshark` packet trace to file output_8_2.

**Lab report:** Use the captured data to answer the questions above. Support your answers with the saved `Wireshark` data. Upload the files output_8_2_tcptrace.jpg and output_8_2 together with the report.

## 8.3 Interactive application - slow link

1. On router A and router B, change the routes so the traffic between the hosts will go through the slow link.

2. Establish a `telnet` connection from Host A to Host B on the slow link Log in with *qwerty* as the root password.

3. Start capturing packets with `Wireshark` on Host A.

4. Type a few characters in the `telnet` application. Vary the rate at which you type characters. Observe the output from `Wireshark`. Answer the following questions. **AFTER THE LAB:**

   (a) How many packets are transferred for each keystroke? Does the number change when you type faster?

   (b) Do you observe delayed acknowledgements?

   (c) Do you observe the effect of Nagle's algorithm? How many characters can you see in a segment?

5. Stop capturing packets.

6. Save the `Wireshark` output to file output_8_3.

**Lab report:** Use the captured data to answer the questions above. Support your answers with the saved `Wireshark` data. Upload the file output_8_3 together with the report.

## 8.4 Bulk transfer - slow link

1. Start capturing packets with `Wireshark` on Host A.
2. Start a `ttcp` receiver on host B and a sender on host A. Send 50 packets of the length 1000 bytes each.
3. Stop capturing packets.
4. Draw a tcptrace graph. Study the graph carefully.
5. **AFTER THE LAB:** Observe the differences compared to the bulk data transfer on the fast link. Answer the following questions.
   (a) Look at the pattern of segments and ACKs. Did the frequency of ACKs change compared to the bulk transfer on the fast link? How?
   (b) Are the window sizes advertised by the receiver different from those of the previous exercise?
   (c) Does the TCP sender generally transmit the maximum number of bytes as allowed by the receiver?
6. Save the tcptrace graph to file output_8_4_tcptrace.jpg and save the `Wireshark` packet trace to file output_8_4.

**Lab report:** Use the captured data to answer the questions above, with special emphasis on the difference with the bulk transfer on the fast link. Support your answers with the saved `Wireshark` data. Upload the files output_8_4_tcptrace.jpg and output_8_4 together with the report.

# 9 TCP retransmissions

In this exercise you will study TCP retransmissions. TCP uses ACKs and timers to trigger retransmissions of lost segments. In order to cause retransmissions, you will have to artificially introduce errors on some of the links. To do so, you are going to use `tc`, a kernel module for manipulating traffic control settings. In this exercise you are going to introduce losses on the interface *eth1* of host *B*. The command to introduce losses is:

```
tc qdisc add dev eth1 root netem loss 1%
```

You can verify that the delay is applied by typing

```
tc qdisc show dev eth1
```

The above command tells the kernel to introduce a 1% loss at the interface *eth1*, namely to drop on average 1 every 100 packets going out from that interface. To restore zero losses you can simply type

```
tc qdisc change dev eth1 root netem loss 0%
```

or you can cancel outright the queueing discipline between the network layer and the network adapter if you are not going to use it later on. Cancelling the queueing discipline is done by invoking the command

```
tc qdisc del dev eth1 root
```

1. Start a second terminal window (Terminal 2) as `root` with `sudo -s`. You are going to use this terminal to switch on and then off losses at step 4
2. Start capturing packets with `Wireshark` on Host B.
3. Start a `ttcp` receiver on host B and a sender on host A, send 50 packets of length 1000 bytes each.
4. When at least 40 packets have been captured by `Wireshark`, in Terminal 2 introduce 100% losses at the interface *eth1* of Host *B*. Wait for one minute (watch `Wireshark` and wait for a few packets to be transmitted) then reinstate zero losses at *eth1*.
5. When the transfer is completed, stop capturing packets.
6. Study the list of captured packets. Draw a tcptrace graph. Observe when retransmissions take place (during the ”disconnection” of the Ethernet cable) and answer the following questions.
   **AFTER THE LAB:**
   (a) How many packets are transmitted at retransmission timeout?
   (b) Do the retransmissions end at some point?
7. Save the `Wireshark` output to file output_9_1.
8. Save the tcptrace graph to file output_9_1_tcptrace.jpg.

**Lab report:** Use the captured data to answer the questions above. Support your answers with the saved `Wireshark` data. Upload the files output_9_1 and output_9_1_tcptrace.jpg together with the report.

## 10 TCP congestion control

In this exercise, the behaviour of TCP congestion control is examined. TCP congestion control operates in two phases, called slow start and congestion avoidance. Congestion is simulated by introducing losses on the link between Host *A* and Router *A*.

1. On the terminal of both hosts run the following command to check and make sure the congestion control algorithm used is Reno.

   ```
   sudo sysctl net.ipv4.tcp_congestion_control
   ```

   If the congestion control algorithm is not Reno, change the algorithm by using the following command:

```
echo reno > /proc/sys/net/ipv4/tcp_congestion_control
```

2. Configure the routes so the traffic from Host A to Host B goes through the fast link and the traffic from Host B to Host A goes through the slow link.
3. Start a new terminal on Host A and introduce a 1% loss rate for the outgoing traffic at interface *eth1* using the `tc` command as in the previous exercise.
4. Start capturing packets with `Wireshark` on Host A.
5. Start a `ttcp` receiver on Host B and a sender on Host A, send 2000 packets with a length of 1000 bytes.
6. After the transmission is completed go carefully through the list of captured packets by Wireshark. What do you observe?
7. Draw a tcptrace graph in `Wireshark`.
   **AFTER THE LAB:**
   (a) Try to observe periods when TCP sender is in slow start phase and when the sender switches to congestion avoidance. Verify if the congestion window follows the rule of the slow-start phase.
   (b) Can you find occurrences of fast recovery?
8. Save the `Wireshark` output to file output_10_1.
9. Save the tcptrace graph to file output_10_1_tcptrace.jpg.
10. Configure the routes so the traffic from Host B to Host A goes through the fast link again. Go through steps 4 - 7 again. Include `Wireshark` data and the tcptrace graph to support your answer. Annotate the events in the tcptrace graph, and explain the events that you observe. Save the wireshark output to be output_10_2 and the tcptrace graph to file output_10_2_tcptrace.jpg.

Please ask one of the lab assistants to check your progress before you continue the lab.

## 11 OPTIONAL: Link Sharing

The `qdisc` scheduling we used previously does not distinguish between different classes of traffic. In this exercise we will use priority queuing to share the bandwidth of a link among competing flows and we will see how TCP reacts. Such a solution can be used, for example, when you have one physical link and you want to divide it into several slower links for different purposes (Figure 5).

Recall that priority queuing is used in the Differentiated Services (DiffServ) architecture, and the priority of a datagram is determined by its ToS field. Unlike in DiffServ, in

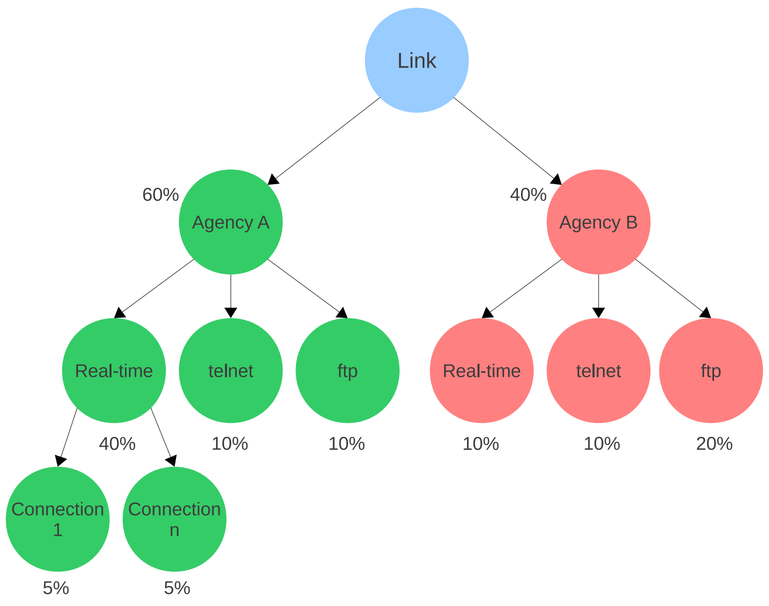


Figure 5: A hierarchical link-sharing structure

this exercise we do not use the ToS field in the IP header to assign priority to datagrams but we create classes of flows based on the TCP port that they use.

To divide the link capacity, we use a packet scheduler called *Hierarchical Token Bucket* (HTB). HTB assigns capacity to classes of flows, and ensures that the amount of service provided to each class is at least the minimum of the amount it requests and the amount reserved for it.

First, we configure the HTB queuing discipline at the interface *eth1* and give it the "handle" 1. We then limit the rate of the outgoing link to $12kbps$.

```
tc qdisc add dev eth1 root handle 1:0 htb default 3
tc class add dev eth1 parent 1:  classid 1:1 htb rate 12kbps ceil 12kbps
```

Next, we divide traffic into two classes of flows, both have an average (assured) rate of $5kbps$ and a maximum (ceil) rate of $12kbps$.

```
tc class add dev eth1 parent 1:1 classid 1:2 htb rate 5kpbs ceil 12kbps
tc class add dev eth1 parent 1:1 classid 1:3 htb rate 5kpbs ceil 12kbps
```

We create filters for the two classes of flows and we assign priority 0 to the high-priority traffic and priority 1 to the low-priority traffic.

```
tc filter add dev eth1 protocol ip parent 1:0 prio 0 u32 match ip
dport <dest_port_1> 0xffff flowid 1:2
tc filter add dev eth1 protocol ip parent 1:0 prio 1 u32 match ip
dport <dest_port_2> 0xffff flowid 1:3
```

Traffic that doesn't match any of the filters will be given an assured rate of 2kbs (i.e., the remaining bandwidth).

```
tc class add dev eth1 parent 1:1 classid 1:4 htb rate 2kpbs ceil 12kbps
```

1. On Host B, start capturing packets with `Wireshark` and set `ttcp` listeners on two ports.

2. On Host A, start sending 200 low-priority packets with a length of 1000 bytes. The destination port should be *dest_port_2* to match the configured filter.
3. When around 50 packets are sent, establish the high-priority flow by sending 100 packets with a length of 1000 bytes to destination port *dest_port_1*.
4. After the transmission is finished, observe the list of the captured packets.
5. What happened when you introduced the high-priority flow?
6. Calculate the rates at which the packets were sent.

## 12 Completing the lab

In order to complete the lab, you will have to put together all the output of Wireshark (packet data and graphs).

- In case you used a raspberry Pi, you will find instructions on the Desktop of the raspberry Pi on how to move files from the RPi to your computer. After the files are saved to your computer, you should delete all Wireshark files from the raspberry Pi, and show one of the lab assistants that you have removed the files.
- In case you used one of the Lab computers, you can follow the following instructions: If you have a USB stick, you can use it to send the files. Please be careful while inserting the USB stick in order to not to affect the USB stick including the OS. Otherwise if you do not have a USB stick, then you will have to first transfer the files to one place, as follows.

  1. Collect the output into one common archive. One way to do this is to copy all files over to Host A, and then build an archive.
     ```
     Host A# scp IP_of_Host_A:/home/user/* .
     Host A# tar czf lab2.tgz *
     ```
  2. If your output is not directly saved on a USB stick you can transfer the archive to your personal mailbox or computer by connecting to the Internet. The lab assistants will give you instructions on how to connect to the Internet and upload your archive.
  3. Transfer the archive to a remote Internet site by using ftp, scp, mozilla, or some other tool.
  4. Erase the files from the routers and power them down.
     ```
     Router A# rm -rf /home/ep2120_lab2
     Router B# rm -rf /home/ep2120_lab2
     Router A# halt -p
     Router B# halt -p
     ```
  5. Power down the hosts.
     ```
     Host # halt -p
     ```

At the end, disconnect the cables and ensure that the equipment is in the same state (or better) as when you started.

## Appendix: Example lab report outline

Measurements

- Measuring TCP with `ttcp` and `Wireshark`
- TCP connection management
  - Connection establishment and termination
  - Connecting to a non-existent port
  - Connecting to a non-existing host
- Fragmentation in TCP
- TCP data transfer
  - Interactive application - fast link
  - Bulk transfer - fast link
  - Interactive application - slow link
  - Bulk transfer - slow link
- TCP retransmission
- TCP congestion control
- Attachment: `Wireshark` output
  - output_5
  - output_6_1
  - output_6_2
  - output_6_3
  - output_7_A
  - output_7_B
  - output_8_1
  - output_8_2
  - output_8_2_tcptrace.jpg
  - output_8_3
  - output_8_4
  - output_8_4_tcptrace.jpg
  - output_9_1
  - output_9_1_tcptrace.jpg
  - output_10_1
  - output_10_1_tcptrace.jpg
  - output_10_2
  - output_10_2_tcptrace.jpg