IK1203 VT20 - Networks and Communication

Lab 1 - HTTP, SMTP, POP3 and Wireshark

Student Name: _____

Date: _____

Lab Instructor: _____

Network Systems Lab (NSLab)
Communication Systems Division (COS)
EECS School
KTH - Royal Institute of Technology
Isafjordsgatan 39
164 40 Kista

Task 1

Internet Protocols

1.1 Purpose

This lab will give you hands-on experience with some of the most widely used application protocols in the Internet: HTTP, SMTP, and POP3.

1.2 Preparations

Before the lab, read the following sections in the textbook:
- Section 2.2, The Web and HTTP
- Section 2.4, Electronic Mail and the Internet
- Follow the instructions on the course web that describe how to install a VPN client and to connect to the laboratory network.

1.3 Introduction to Telnet

Telnet is available in most operating systems, including Windows, Linux, and Mac OS X. It provides you with a TCP connection to a server, which you can use to send text to the server and see any messages returned. By default, the connection is line-buffered, which means that no data is sent until you press the <Enter> key.

In case of Windows, the telnet tool may not be installed by default (it depends on the Windows version and installation procedure). You can check it by issuing the command telnet in the command line. If there is a problem in running telnet, then you can use PuTTY, a free telnet and SSH client. PuTTY can be downloaded from the website: http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html

Instructions for PuTTY users.
- Chose telnet as connection type.
- Make sure that the host name and connection port are clearly defined (the default port for telnet is 23 and in this lab you will use other ports too).
- In the telnet configuration (Connection/Telnet), change the Telnet negotiation mode to Passive.
- Select the option 'Never' in "Close window on exit:" in the Session menu.

An alternative to PuTTY is Daves Telnet, a replacement for Windows standard Telnet program. You can download this program from the website: http://sourceforge.net/projects/dtelnet/files/4.

If you are using Mac or Linux OS, the terminal should be more than adequate. For starting telnet on Mac, you might need to install it first. You can do this using "brew install telnet" command. Open the terminal app located in the **Applications/Utilities** folder. From the **Shell** menu → Select **New Remote Connection...** → Select **Remote Login (telnet)** under service. In the field at the bottom of the dialog, you can directly enter the commands as shown in the examples.

NOTE: Before starting PuTTY/Daves Telnet/terminal, make sure that you are connected to the VPN and able to ping 192.168.3.12. You can do so by following the VPN setup instructions on the course web. Moreover, you should be able to resolve the DNS name mail.private.lab to 192.168.3.16. If not, you may need to add 192.168.3.12 to your

list of DNS server. (You can do this in Linux by adding a line "nameserver 192.168.3.12" to /etc/resolve.conf file).

Telnet accepts two arguments. The first is the address of the host that you want to connect to. The second argument is the port number of the service or server. If you don't provide the second argument, port 23 will be used. The port number determines which service you access. The Internet Assigned Numbers Authority (IANA) maintains a list of well-known port numbers for services. See http://www.iana.org/assignments/port-numbers for the complete list of well-known port numbers. Below, follows a transcript from a short Telnet session against the SMTP server on host mail.private.lab. As shown here, the service name, instead of the port number, can be used as the second argument to the telnet command.

```
student@laptop:/etc$ telnet mail.private.lab smtp
Trying 192.168.3.16...
Connected to mail.private.lab.
Escape character is '^]'.
220 mail.private.lab ESMTP Postfix (Ubuntu)
quit
221 2.0.0 Bye
Connection closed by foreign host.
```

Figure 1.1: An example SMTP session.

Figure 1.1 shows a simple SMTP session initiated via telnet. The first line shows the command prompt followed by the telnet command. The next three lines are informational messages generated by telnet, followed by a line that comes from the server process ("220 mail.private.lab […]"). We have connected to port 25 (SMTP), which is the mail server, and a brief welcome message is sent by the server. We enter the command quit and press the <Enter> key. The server responds with the 221 2.0.0 Bye message and immediately closes the connection, causing telnet to generate the message: Connection closed by foreign host.

NOTE: The first four lines in the example do not show up if you are using PuTTY.

1.4 The HyperText Transfer Protocol (HTTP)

1.4.1 Introduction

The HyperText Transfer Protocol (HTTP) is the primary transfer protocol that a Web browser uses to interact with a Web server. HTTP specifies what messages Web browsers may send to servers, and what responses they get back in return. The most common version of HTTP, i.e., HTTP 1.1, is defined in the IETF standard RFC 2616. HTTP interaction typically begins with a browser requesting a page from a server. The browser sends a HTTP GET request over the connection and the server responds by sending a header, a blank line, and the requested document. A transcript from an HTTP session is provided in Figure 1.2.

*It should be noted that no cursor will be displayed. It is also necessary to hit the <Enter> key twice after the **Host:** line.

1.4.2 Questions

Q 1. Choose a Web server and try to access the main page using Telnet to port 80 and HTTP/1.1. What was the HTTP request you used to download the page successfully?

Q 2. Connect to **www.stockholm.se, www.time.com, www.un.org**, and **www.google.se**. For each Web server, try to figure out what Web server software it is using (for example, Apache or Microsoft Internet Information Services), and on what operating system it is running.

Q 3. You used the GET method to get the entire HTTP response, including the header and the body. Try to retrieve only the header of the webpage index.html on webserver ik1203.private.lab (IP: 192.168.3.16) using HTTP/1.1. What command did you use for this and what response did you get?

```
student@laptop:~$ telnet ik1203.private.lab 80
Trying 192.168.3.16...
Connected to mail.private.lab.
Escape character is '^]'.
GET /index.html HTTP/1.1
Host: ik1203.private.lab

HTTP/1.1 200 OK
Date: Sun, 20 Jan 2019 21:24:54 GMT
Server: Apache/2.4.7 (Ubuntu)
Last-Modified: Mon, 26 Jan 2015 13:44:06 GMT
ETag: "79-50d8e5500f280"
Accept-Ranges: bytes
Content-Length: 121
Vary: Accept-Encoding
Content-Type: text/html

<HTML>
<HEAD>
        <TITLE>IK1203 web server</TITLE>
</HEAD>

<BODY>
        <H1>Welcome to IK1203 web server!</H1>
</BODY>
</HTML>

Connection closed by foreign host.
```

Figure 1.2: An example HTTP session.

1.5 The Simple Mail Transfer Protocol (SMTP)

E-mail is one of the most popular services on the Internet, and the Simple Mail Transfer Protocol (SMTP) is the standard protocol that is used to transfer e-mail messages from the sender to the recipient's mailbox. The SMTP protocol is defined in RFC 2821. SMTP is a text-based protocol: it uses textual control messages and only transfers text messages.

```
student@laptop:~$ telnet 192.168.3.16 smtp
Trying 192.168.3.16
Connected to 192.168.3.16
Escape character is '^]'
220 mail.private.lab ESMTP Postfix (Ubuntu)
```

```
Client: HELO mail.private.lab
Server: 250 mail.private .lab
Client: MAIL FROM: <user1@private.lab>
Server: 250 2.1.0 Ok
Client: RCPT TO: <user1@private.lab>
Server: 250 2.1.5 Ok
Client: DATA
Server: 354 End data with <CR><LF>.<CR><LF>
Client: A test message from student at time 09:10
Client:.
Server: 250 2.0.0 Ok: queued as 64D127EAEF
Client: quit
Server: 221 2.0.0 Bye
```

Figure 1.3: An example SMTP session.

Figure 1.3 shows an example SMTP session. In the figure, lines are labeled **Client:** or **Server:** to indicate whether the client or the server sends the line. Please note that the protocol **does not include** these labels. SMTP is a protocol where the client and server take turns in speaking. When the server is ready to service the client, the server starts by sending a 220 response code.

The mail.private.lab machine hosts the SMTP server for the private.lab domain (that is, the NSLab lab network). There are 50 user accounts setup on the SMTP-server host (username: user1, user2, …, user50, and passwords are the same as the usernames). The SMTP server only permits these users to send e-mail messages to each other, meaning that e-mail messages to other domains are not permitted.

The example SMTP session in Figure 1.3 is actually a real session against the SMTP server of the private.lab domain. Note that the sender and receiver e-mail addresses have to use fully qualified domain names. The accounts login/password, follow user1/user1, userX/userX … rule.

1.5.1 Questions

Q 4. Send a one-line email message to one of the user accounts on the SMTP server. Give a transcript of your interaction with the server.

Q 5. Try out the example in Figure 1.3 and check whether it is possible to provide a non-existent e-mail address in the MAIL FROM: command?

Q 6. Is it possible to provide a non-existent e-mail address in the RCPT TO: command?

1.6 The Post Office Protocol (POP3)

The most popular e-mail access protocol is known as version 3 of the Post Office Protocol (POP3), and is defined in RFC 1939. The user invokes a user agent (POP3 client), which creates a TCP connection to a POP3 server on the mailbox host (typically, the SMTP-server host). The user first sends a login and a password to authenticate the session. Once authentication has been accepted, the user agent sends commands to do such things as list all messages in the mailbox, retrieve a copy of one or more messages, or delete a message from the mailbox. The messages are stored and transferred as text files.

```
student@laptop:~$ telnet 192.168.3.16 pop3
Trying 192.168.3.16
Connected to 192.168.3.16
Escape character is '^]'
+OK Hello there.
Client: user user1
Server: +OK Password required.
Client: pass user1
Server: +OK logged in.
Client: list
Server: +OK POP3 clients that break here, they violate STD53.
1 338
.
Client: quit
Server: +OK Bye-bye.
Connection closed by foreign host.
```

Figure 1.4: An example POP3 session

In the private.lab domain, the host, mail.private.lab, hosts both the SMTP and POP3 servers. Figure 1.4 shows a transcript of a simple POP3 session against the POP3 server of the private.lab domain. As before, lines are labeled **Client:** or **Server:** to indicate whether it is the client or the server that sends the line. They are not part of the SMTP commands or responses.

Look up RFC 1939, and answer the questions below. The RFC 1939 can, for example, be found at the RFC Editors search page, http://rfc-editor.org/rfcsearch.html.

1.6.1 Questions

Q 7. How do you retrieve the e-mail message you sent in Section 1.5 assuming that you are now the recipient of that email. Answer with a transcript of the POP3 session.

Q 8. Try to retrieve only the header lines of an e-mail message without retrieving the complete message. What POP3 command is used for doing this?

Q 9. How do you delete the email message you sent to yourself in Section 1.5? Answer with a transcript of the POP3 session.

NOTE: You can now stop PuTTY/Daves Telnet/terminal as well as the VPN client. If you manually added the DNS server (192.168.3.12) to your computer, you will need to remove it as well.

Task 2

Wireshark

2.1 Purpose

This lab is essentially a compilation of several Wireshark labs that accompany the course textbook. This lab will provide you with the basics on capturing and analyzing Internet traffic using a so-called packet sniffer. Further, since you will observe Internet protocols such as HTTP, TCP, and IP in action, the lab will also reinforce your knowledge about these protocols.

2.2 Preparations

Before the lab, the following sections in the textbook have to be read:

- Section 2.2, The Web and HTTP
- Section 3.5, Connection-Oriented Transport: TCP
- Section 3.7, TCP Congestion Control
- Section 4.4, The Internet Protocol (IP): Forwarding and Addressing in the Internet
- Section 5.4, Link-Layer Addressing

2.3 Introduction to Protocol Analysis

The basic tool for observing the messages exchanged between executing protocol entities is called a packet sniffer. As the name suggests, a packet sniffer captures ("sniffs") messages being sent/received from/ by your computer; it will also typically store and/or display the contents of the various protocol fields in these captured messages. A packet sniffer itself is passive. It observes packets sent and received by applications and protocols on your computer, but never sends packets itself. Similarly, received packets are never explicitly addressed to the packet sniffer. Instead, a packet sniffer receives a copy of packets that are sent and received from/by applications and protocols executing on your machine.

In this lab, we will be using the Wireshark packet sniffer. Wireshark is a free network protocol analyzer that runs on Windows, Linux/Unix, and Mac computers. Wireshark has a rich functionality that includes the capability to analyze hundreds of protocols. This means that Wireshark understands the structure of all of the messages exchanged by the protocols in question. It also has a well-designed, relatively easy-to-use user interface.

2.4 Running Wireshark

To be able to capture packets from the network interfaces of your computer, you need to run Wireshark as a user with administrator privileges. If you run Windows, and are using your personal computer, you could probably run from your own account, since you are probably your own administrator. In Linux, you could run Wireshark from your personal account (which probably does not have root privileges) and use the "sudo "command to run Wireshark with root privileges. Some basic information about different components in the Wireshark window is provided in Appendix.

The easiest way to start a packet sniffing session is to click on the appropriate network interface in the Capture part

of the main window. However, you can also start a sniffing session by selecting an interface from the Interface List, or via the Capture Options dialog window.

2.5 Sniffing HTTP Traffic

In this section, we will explore several aspects of the HTTP protocol: the basic GET/response interaction, caching and conditional GET, retrieval of long HTML files, and finally retrieval of HTML files with embedded objects.

2.5.1 The Basic HTTP GET/Response Interaction

Let's begin our exploration of HTTP by downloading a very simple HTML file, one that is very short, and contains no embedded objects. Do the following:

1. Start your Web browser. (Make sure that there is no other Web browser session running!)
2. Start the Wireshark packet sniffer and enter http in the packet-display filter field.
3. Wait a few seconds and then begin Wireshark packet sniffing.
4. Enter the following URL in your Web browser: http://www-net.cs.umass.edu/wireshark-labs/HTTP-wireshark-file1.html. Your browser should display a very simple HTML file.
5. Stop the packet sniffing.

2.5.2 Questions

By looking at the information in the HTTP GET and response messages, answer the following questions:

Q 10. Which version of HTTP is your Web browser running? What version is the server running?

Q 11. What is the IP address of your computer and of the web server?

Q 12. What size in bytes has the "HTTP-wireshark-file1.html" file?

2.5.3 The HTTP Conditional GET/Response Interaction

Recall from Section 2.2.6 of the course textbook, that most Web browsers perform object caching, and thus perform a conditional GET when retrieving an HTTP object. That is, when the browser sends a request, it includes a header, **If-Modified-Since**, that qualifies conditions under which the request should be honored. Now, do the following:

1. Start your Web browser, and make sure your browser's cache is cleared.
2. Start the Wireshark packet sniffer and begin packet sniffing.
3. Enter the following URL in your Web browser: http://www-net.cs.umass.edu/wireshark-labs/HTTP-wireshark-file2.html. Your browser should display a very simple five-line HTML file.
4. Quickly enter the same URL into your browser again (or simply select the refresh button on your browser).
5. Stop the Wireshark packet sniffing, and enter http in the packet-display filter field so that only captured HTTP messages will be displayed in the packet-listing window.

2.5.4 Questions

Study the captured packets and answer the following questions:

Q 13. Inspect the contents of the first HTTP GET request from your browser to the server. Do you see an If-Modified-Since line in the HTTP GET?

Q 14. Inspect the contents of the server response. Did the server explicitly return the contents of the HTML file?

Q 15. Now inspect the contents of the second HTTP GET request from your browser to the server. Do you see an If-Modified-Since line in the HTTP GET?

Q 16. What is the HTTP status code and phrase returned from the server in response to the second HTTP GET? Did the server return the content of the file?

2.5.5 Retrieving Long HTML Files

So far, the retrieved HTML files have been simple and short. Let's see what happens when we download a large HTML file. Do the following:

1. Start your Web browser, and make sure the browser's cache is cleared.
2. Start the Wireshark packet sniffer and begin packet sniffing.
3. Enter the following URL in your Web browser: http://www-net.cs.umass.edu/wireshark-labs/HTTP-wireshark-file3.html. Your browser should display the rather lengthy US Bill of Rights.
4. Stop the Wireshark packet sniffing, and enter http in the packet-display filter field so that only captured HTTP messages are displayed in the packet-listing window.

2.5.6 Questions

Study the captured packets and answer the following questions:

Q 17. How many HTTP GET request messages were sent by your Web browser?

Q 18. How many TCP packets were included in the response from the Web server to retrieve HTML file?

2.5.7 Retrieving HTML files with Embedded Objects

To conclude this section about sniffing HTTP traffic, we will look at what happens when your Web browser downloads a file with embedded objects such as image files. Do the following:

1. Start your Web browser, and make sure your browsers cache is cleared.
2. Start the Wireshark packet sniffer and begin packet sniffing.
3. Enter the following URL in your Web browser: http://www-net.cs.umass.edu/wireshark-labs/HTTP-wireshark-file4.html. Your browser should display a short HTML file with two images.
4. Stop the Wireshark packet sniffing, and enter http in the packet-display filter field, so that only captured HTTP messages are displayed in the packet-listing window.

2.5.8 Questions

Study the captured packets and answer the following questions:

Q 19. How many HTTP GET request messages were sent by your Web browser? To which IP addresses were these GET requests sent?

Q 20. Can you tell whether your browser downloaded the images in HTTP-wireshark-file4.html serially or in parallel? Explain.

Congratulations!
This concludes the lab

Appendix

The Wireshark main window in Figure 2.1 has six major components:

Menu Bar: The Menu Bar contains standard pull-down menus located at the top of the window. Of particular interest to us is the File and Capture menus. The File menu allows us to save captured packet data, or open a file containing previously captured packet data, and exit the Wireshark application. The Capture menu allows us to start, stop, or restart a packet capture.

Toolbar: The Toolbar provides us with the most common tasks including selecting network interfaces, and starting and stopping a packet sniffing session. Move the mouse pointer over the toolbar to find out which icon provides what task.

Capture Filter Field: The Capture Filter Field enables us to filter what is displayed in the packet-listing window (and hence what is displayed in the packet-header details window and the packet-contents window). For example, by typing in http in lower-case letters, only HTTP traffic is displayed.

Packet List Panel: The packet List Panel displays a one-line summary for each packet captured, including the packet number (assigned by Wireshark; this is not a packet number contained in any protocols header), the time at which the packet was captured, the packets source and destination addresses, the protocol type, and protocol-specific information contained in the packet. The packet listing can be sorted according to any of these categories by clicking on a column name. The protocol type field lists the highest level protocol that sent or received this packet, i.e., the protocol that is the ultimate source or sink for this packet.

Packet Details Panel: The packet Details Panel provides details about the packet selected (highlighted) in the packet-listing window. To select a packet in the packet listing window, place the cursor over the packets one-line summary in the packet listing window and click with the left mouse button. These details include information about the Ethernet frame (assuming the packet was sent/received over an Ethernet interface) and IP datagram that contains this packet. The amount of Ethernet and IP-layer detail displayed can be expanded or minimized by clicking on the plus/minus boxes to the left of the Ethernet frame or IP datagram line in the packet details window. If the packet has been carried over TCP or UDP, TCP or UDP details will also be displayed, which can similarly be expanded or minimized. Finally, details about the highest level protocol that sent or received this packet are also provided.

Packet Dissector Panel: Finally, the Packet Dissector Panel displays the entire contents of the captured frame in both ASCII and hexadecimal format.

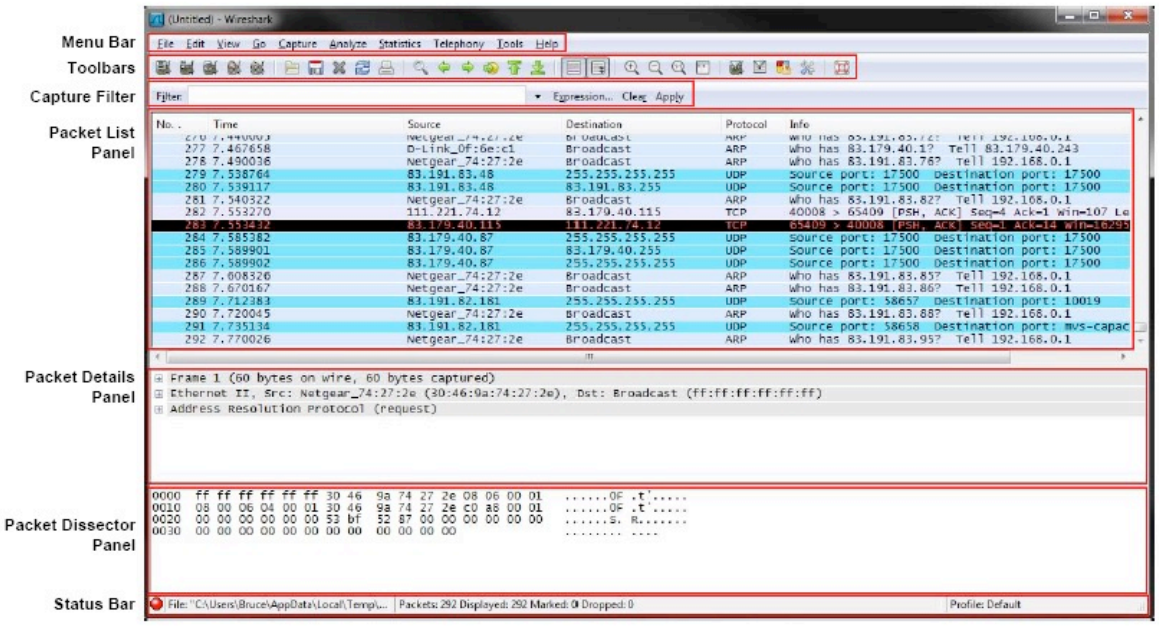

Figure 2.1: The Wireshark main window

Appendix

This lab instruction has been through many iterations of updates by the following people:
Johan Bilen (bilen@kth.se), Erik Eliasson (ekiasson@kth.se), Fredrik Lundevall (flu@kth.se), Karl-Johan Grinnemo (grinnemo@kth.se), Georgios Cheimonidis (gche@kth.se), Andreas Tsopelas (tsopelas@kth.se), Bruce Zamaere (bsiza@kth.se), Pawel Wiatr (wiatr@kth.se), Mozhgan Mahloo (mahloo@kth.se), Muhammad Rehan Raza (mrraza@kth.se), Yuxin Cheng (yuxinc@kth.se), Voravit Tanyingyong (voravit@kth.se), Cihan Eryonucu (eryonucu@kth.se) and Marco Spanghero (marcosp@kth.se)