Computer networks. Experimental Study 3rd Laboratory Lab.

Transport layer on the Internet.

Departamento de
Tecnología Electrónica

- **DO NOT TURN ON THE PC UNTIL INSTRUCTED TO DO SO.**
- The experimental study of this lab consists of two parts.
- If you have any questions, consult the professor in charge of the practical session.

Previous steps

1. Make sure you are connected to the ETSII network.
2. Turn on your PC, restore Windows 7 (if instructed to do so by your professor), and boot Windows 7.
3. Disable the firewall.
4. Disconnect your PC from the ETSII network and connect it to LAB_DTE, specifically to the SWITCH_EUROPA (in G1.31) or SWITCH_SUDAMERICA (in G1.33). If there are no ports available in the appropriate SWITCH, tell the professor. Check that you have physical layer connectivity.
5. Check that your PC's IP address starts with 193, write it down, and verify that you have network layer connectivity to your border router and to the webserver.af.lab computer (with a ping).
6. Don't go ahead until you are sure you have connectivity to your border router.
7. Additionally, in this practice, "click" on the "Network connection" icon., then click "Abrir Centro de redes y recursos compartidos" > "Conexión de área local" and select "Propiedades" from the context menu. In the window that appears, deactivate " Cliente para redes Microsoft", deactivate " Compartir impresoras y archivos para redes Microsoft" and deactivate "Programador de paquetes QoS". Click "Accept" to close the window and then "Cerrar". Do not leave any windows open.

First Part: Design a filter to see only the T_PDUs generated when downloading a small web page.

The objective of this part of the lab is to be able to see the exchange of T_PDUs produced when loading in a browser the page http://webserver.af.lab/lab3/paginasimplelab3.html without referenced objects and with a very small size.

The main idea seems simple, because it would be enough to put Wireshark to capture traffic, start the browser, load the page, stop Wireshark and see what has been captured. However, it is not as simple as it seems because in the capture we can see much more traffic than we would expect at first, because on our computer there is a lot of software that generates network traffic. There is also other devices on the network that also generate traffic and finally also because the circumstances of saturation of the network can cause retransmissions to occur that we did not have at first. It is very normal or even "impossible" that we do not get a "clean" capture the first time, containing only the traffic we want to study.
There are several possible solutions:

A) Try to configure the software on our PC in such a way that it generates the least amount of "unwanted" traffic. That is what we have done, for example, in the section 7.
B) Apply display filters to the "imperfect" capture so that only the desired traffic is shown. These filters can be prepared before capturing, but it is also useful to prepare them after capturing, once we have obtained the appropriate values for certain filter parameters (for example port numbers, IP addresses, etc.).
C) Repeat again the generation of traffic and its capture until fortune makes us obtain a "clean" capture, with the desired traffic.

We are going to use all three techniques: First, configure the PC to generate little "extra" traffic. Second, use visualization filters that prevent us from seeing most of the traffic that we are not interested in and that we know that if we do not filter it, it will appear in most of the captures. Finally, if despite the filters we are shown some unwanted traffic but it is unlikely to appear, it is better to repeat the process of generating and capturing network traffic instead of designing a complex display filter that hides it from us.

8. In the previous practice, in sections 49 and 50 we were explained the need to configure the behavior of the Mozilla Firefox browser to adapt it to the laboratory's intranet, which has links between routers with a very low bandwidth (of about 56000bps), so that the browser does not open more TCP connections than necessary, even though the connection establishment time is long. Follow the same steps in section 50 of the previous practice again: Open Mozilla Firefox, enter the special URL about: config, accept the "aviso para manazas ", search for the phrase "retry-timeout", double-click the value of the "network.http.connection-retry-timeout" preference to set it to 12000 and close Mozilla Firefox .

9. Enter Wireshark and make sure the options **"Resolve network (IP) addresses"** and **"Resolve transport names"** are deactivated (something similar you did in point 8 of the previous laboratory session).
10. **Set Wireshark to capture traffic and do not stop until specifically instructed to do so.** You will be seeing that at least one new frame is received every two seconds. This indicates that it will be impossible to make a "clean" capture in the laboratory network, since without having generated the network traffic that interests us, we are already capturing frames. We will need to use display filters to see the only thing that interests us: the T_PDUs related to the download of a web page.
11. We are going to try a very simple first basic filter. Since we are only interested in the T_PDUs, enter and apply the display filter udp or tcp.port == 80 to see only the frames that make that logical expression true, that is, they contain T_PDUs of the UDP protocol (the UDP_PDUs) or they contain T_PDUs of the TCP protocol (the TCP_PDUs) that use port 80. Remember that TCP port 80 is the "well-known port" that HTTP uses, so that is why we have preferred to use the expression tcp.port == 80 in the filter. instead of simply tcp, to see only the TCP traffic that uses port 80 as a source or as a destination, avoiding seeing other TCP traffic that we are not interested in and that, depending on the software we are running, could be very abundant.
12. If all goes well, you will now be capturing frames, as before, since the display filter limits the frames displayed, not the captured ones. Thus, you can see that the "Packets" counter displayed at the bottom of the Wireshark window continues to grow, while the "Displayed" counter will be at a lower value than before. Wait for the value of those two counters to be greater than zero and record their current values here.
13. We're going to improve the display filter so that it lets through even less traffic. We will combine the filter udp or tcp.port == 80 with another filter that ensures that it is traffic that goes to or comes from our own IP address. The filter to use will be similar to (udp or tcp.port == 80) and ip.addr == 193.1.XY where obviously the values of X and Y will depend on what your IP address is. Write down the filter to use below, not forgetting the parentheses to make clear the precedence of the logical operators. Write and apply that filter in Wireshark.

14. **<u>Take note of</u>** the values of the "Packets" and "Displayed" counters.
15. Although better than before, the filter currently applied is insufficient. To demonstrate this we could wait for about fifteen or twenty minutes and we would see that the filter lets through a lot of traffic that is generated periodically and that we are not interested in visualizing. In order not to waste time waiting, we are going to do one thing that will force the generation of this traffic immediately without having to wait long. Disconnect the network patch cord from your PC from the wall socket for about three seconds and reconnect it. Wait a minute for the traffic to slow down, then write down the values for the "Packets" and "Displayed" counters.
16. We must not lose sight of the fact that what we want to see is the UDP and TCP traffic associated with loading a web page. This traffic is associated with DNS traffic (over UDP) and HTTP traffic (over TCP). Therefore, instead of using udp as a filter, it is better to use the dns filter. So write and apply the filter in Wireshark (dns or tcp.port == 80) and ip.addr == 193.1.XY and you will see how much UDP traffic disappears from the list of frames, leaving only the UDP traffic that contains A_PDUs of the protocol DNS. Take note of the values for the "Packets" and "Displayed" counters again.
17. Another possible improvement is to use the dns.qry.type == 1 filter instead of the dns filter. With it we will see that we are shown much less DNS traffic than we currently see. This is so because with this filter we are only viewing a certain type of DNS traffic, which is generated by asking for the version 4 IP address of a certain host. It will not allow us to see other types of "querys" (questions) in the plot list, for example those that appear with the AAAA mark, and which, as we already had the opportunity to comment in the previous practice, do not interest us. Modify the current filter in Wireshark to be (dns.qry.type == 1 or tcp.port == 80) and ip.addr == 193.1.XY and apply it. Take note of the values for the "Packets" and "Displayed" counters again.
18. As we continue to see a lot of DNS traffic, we are going to do something that we already saw in the previous lab, which was to use the filter dns contains word that allows us to show only the A_PDUS of the DNS protocol that contain a certain word. Modify the current Wireshark filter to read as ((dns.qry.type == 1 and dns contains lab) or tcp.port == 80) and ip.addr == 193.1.XY and be very careful not to forget the parentheses. Apply the filter, and you will see how the traffic shown will be only TCP traffic (probably none) and DNS traffic in which it asks for the IP version 4 address of those hosts that contain the word lab in their domain name (remember that computers of the lab are all in a domain name ending in .lab). Take note of the values for the "Packets" and "Displayed" counters again.

19. Look at the DNS traffic displayed and make a note of the host names that are being asked for and indicate if the questions are answered.

20. As the web page that we are interested in capturing is on the web server called webserver.af.lab, DNS traffic will be generated to ask for that name, so we are going to make a change to improve the filter. Type ((dns.qry.type == 1 and dns contains webserver) or tcp.port == 80) and ip.addr == 193.1.XY as the display filter in Wireshark, check well that it is correct and apply it. That filter is probably not showing us any frame now, but we will soon see that it can show us all the frames that interest us. Write down the values for the "Packets" and "Displayed" counters again and advise the professor if "Displayed" is not 0.

21. An even better filter, if we know the IP address of the webserver, it would be the following: ((dns.qry.type == 1 and dns contains webserver) or (tcp.port == 80 and ip.addr == ABCD)) and ip.addr == 193.1.XY where ABCD is the IP address of the web server.

Second Part: Analyze the exchange of T_PDUs generated when downloading a small web page

Finally we are going to generate the network traffic that we want to analyze. Follow the instructions below.

22. The Windows DNS cache prevents excessive use of DNS traffic. In our case, since we want to make sure that DNS traffic is generated, what we are going to do is delete it before starting to generate traffic. Clear the Windows DNS cache using the appropriate command in a Command Prompt window. Remember that you used this delete command in the third part of the previous laboratory session. Take note of the clear DNS cache command.
23. **Open Mozilla Firefox browser and clear your page cache.** As it was described to you in the fourth part of the experimental study from the previous lab. Take note of the procedure for clearing the browser's page cache.

24. **If Wireshark was capturing and NO frames are seen in the frame list**, then you don't need to do anything. If Wireshark was capturing and frames are seen in the list of frames, then we must press the "RESTART" icon to discard everything captured and start capturing from scratch. If Wireshark was not capturing, then you have to set it to capture.
25. Make sure that the Mozilla Firefox window is not too large, so that you can see behind the Wireshark window what is happening in the list of frames. Enter the URL in the browser **http: //webserver.af.lab/lab3/paginasimplelab3.html** to load the web page and generate traffic on the network.
26. **Wait about 30 seconds** to stop the traffic on the network and check the list of frames and the number shown in the "Displayed" counter. Two things can happen:
    A) Exactly 13 frames are being displayed (2 frames with DNS traffic and 11 frames with TCP traffic) showing one in the "Info" column the text "HTTP / 1.1 200 OK". This indicates that everything has gone well, so stop capturing Wireshark and go to the next section, the27.
    B) Exactly 15 frames to be seen (2 frames with DNS traffic and 13 frames with TCP traffic) showing one of them in the "Info" column the text "HTTP / 1.1 200 OK" and two of them being dark colored frames, which they are part of the same TCP connection with the server and they have appeared because of the TCP retransmissions. If this is the case, you can stop the capture, but before going to the next section, the 27, you must ensure that those two dark frames are not seen and only have 13 frames displayed as in the point. To do this, you can ignore these dark frames by right-clicking on them and selecting Ignore Packet (toggle) from the context menu, or you can add the condition and not tcp.analysis.flags to the current display filter, so that they are not show the frames related to TCP retransmissions.
    C) If nothing of this occur this is because something has gone wrong and you must close the Mozilla Firefox browser (VERY IMPORTANT) and repeat paying close attention all the steps from the section **22 to this**. Do not go to the section **27 without meeting the conditions of the point A).**
27. In Wireshark, press the "SAVE" icon (or enter "File> Save") to save all captured frames (even if they are not being displayed). Use "allframes" as the capture filename.
28. Wireshark also allows only part of the captured frames to be recorded in a file. In particular, it is very useful to record only the frames that are showed with the current display filter. We are going to save the 13 frames displayed by entering "File"> "Export Specified Packets ..." and marking the "All packets" and "Displayed" boxes in the section called "Packet Range" (if they were not already marked). Use the name "13frames" for the capture file.
29. Deactivate the display filter by clicking the "Clear" button and note the value of the "Displayed" counter when no filter is used. Write and apply (separately) the four simple filters **dns**, udp, tcp, and http and note the value of the "Displayed" counter for each of them. This allows you to get an idea of the power of the complex filter used and that it only showed us 13 frames

30. Deactivate the display filter again by clicking the "Clear" button. In order to work without using a filter, we are going to load the file that only contains 13 frames. To do this, press the folder icon and load the file you recorded earlier with name "13frames".
31. **Notice the 13 frames** that Wireshark is showing you in the frame listing. Those are the frames generated as a result of loading the simple web page in the browser. The first two frames are related to the DNS traffic that has occurred with the DNS server in order to resolve the name of the web server webserver.af.lab. The remaining 11 frames are related to the HTTP traffic needed to ask the web server for the page.
32. Write and apply the dns filter to see only the frames that encapsulate DNS_PDUs.
    How many DNS requests are there relative to the load of the web page of the point 25?
    What IP does the DNS server have?
    What IP does the computer named webserver.af.lab have?
    What RTT is observed for the response to the request?
    Note: if to see the RTT you use the technique of marking a frame as a time reference "Set Time Reference", do not forget to make the * REF * mark disappear before moving on to the next section.
33. Write and apply the udp filter to see only the frames that encapsulate UDP_PDUs.
    Why does the dns filter and the udp filter show the exact same frames?
    In what cases could the dns and udp filters show different things?

    How many T_PDUs of the UDP protocol have been exchanged relative to the simple page load?
    How many bytes does the UDP_UD of each of these UDP_PDUs occupy?
    Note: Don't make the mistake of using the "Length" column to answer this last question. Use the frame details panel and remember what is the structure, meaning and size of the fields of the UDP_PCI present in the UDP_PDU.
34. Write and apply the http filter to see only the frames that encapsulate HTTP_PDUs. That is, to see exclusively the exchange of A_PDUs between the two application entities (the browser and the web server). Look at the information in the frame listing and respond.
    How many A_PDUs are there of the HTTP protocol relative to the load of the web page of the point 25?
    How many HTTP requests are there?
    What IP does the web server named webserver.af.lab have?
    What IP does our PC have?
    What RTT is observed between an HTTP request and its corresponding response?
    Note: if to see the RTT you use the technique of marking a frame as a time reference "Set Time Reference", do not forget to make the * REF * mark disappear before moving on to the next section.
35. Write and apply the tcp filter to see only the frames that encapsulate TCP_PDUs. This will allow us to see the exchange of TCP_PDUs between the TCP entity of our PC and the TCP entity of the computer where the web server is.
    How many T_PDUs of the TCP protocol have been exchanged relative to the page load?
    Why do two frames in the listing show HTTP in the "Protocol" column instead of TCP?
    Is there also a TCP_PDU within those two frames?
    The exchange of TCP_PDUs is much more numerous than the exchange of HTTP_PDUs, as can be seen in the list of frames.
    How many TCP_PDUs are there that do not encapsulate application layer data?
    What use are TCP_PDUs that have a TCP_UD of size zero?

    Let the professor know if you want to review the answers to the four previous sections.

36. Notice that the first two TCP_PDUs in the frame listing are marked in the "Info" column with the labels [SYN] and [SYN, ACK] respectively.
    What is Wireshark based on to label those two segments with the word SYN?
    Is it because they are the first two segments of the capture file?
    Is it because there is some special data inside those two segments?
    If so, locate that data in the center panel (the frame details panel).

37. The same is true for segments labeled with [FIN]. Locate in the central panel that special field that makes the segments to show the word FIN in the "Info" field.
38. **Draw, by hand, a temporal diagram showing the 11 TCP segments** represented as arrows that go from left to right or from right to left with a slight inclination. Y-axis represents time. Label each segment with the Sequence No. (Seq =), Acknowledge No. (Ack =), and size of the application-layer data carried within the segment (Len =). If the segment has special bits set, such as the SYN or FIN bit, indicate this as well. You may find all data you need in the "Info" column for each TCP segment. However, you may find some difficulty with TCP segments that contain data from the HTTP protocol, because in this case the "Info" column shows the HTTP details and not TCP details. To solve this problem, you can find the

Seq, Ack and Len (and other TCP details) of these TCP segments by selecting the desired frame and looking at the line "Transmission Control Protocol" in the central panel (details of frame), which contains very useful information.

39. Look at the temporal diagram and answer the following question: Have any of the TCP entities (client or server) used piggybacking?
40. **Ask the teacher to check your answers.**
41. If we are only interested in seeing in the list of frames all the details of the TCP protocol and we are not interested in seeing the HTTP protocol, we can tell Wireshark to hide the HTTP details. To do so: Go to "Analyze> Enabled Protocols", click on the list of protocols and type HTTP to quickly locate that protocol in the list. Uncheck its box in the "Status" column and press "OK". You will now see that the 11 frames that contain TCP segments only show information about TCP.
42. What is the size of the A_PDU sent by the client?
43. What is the size of the A_PDU sent by the server?
44. In the diagram you made in step 38, both the server and the client appear to be using zero as the initial sequence number. That is not true. In the default configuration, Wireshark shows us **relative sequence numbers**. This means that all initial sequence numbers (both the client and the server), seem to begin with zero when, in fact, they are not.
45. Go to "Edit"> "Preferences" and in the branch "Protocols" look for TCP (type TCP) and **uncheck** the box called "**Relative sequence numbers**". Press "OK" to exit. Notice how the "Info" column of the frame listing now shows the **real sequence numbers** (they are 32-bit positive integers).
46. Look at the first two TCP segments. Are the initial sequence numbers chosen by the client and the server the same?
47. In which segment of the list does the server send its first ACK?
48. What is the difference between the client's initial sequence number and the first acknowledgment number (ACK) sent by the server?
49. Notice that the client does not send an ACK on the first TCP segment. The client does not yet know the server's initial sequence number, so this is the only time that a TCP segment does not have the ACK bit set (set to 1). Check this by writing and applying the filter **tcp.flags.ack == 0** that will show us only one TCP segment, the one with the ACK bit set to 0.
50. The word “**flags**” is what Wireshark uses to refer to the **bits** that appear in the TCP_PCI that we call the ACK bit, SYN bit, FIN bit and RST bit of TCP_PCI. Filters such as **tcp.flags.syn == 1** or **tcp.flags.fin == 1** are frequently used to analyze how TCP establishes and closes a connection.
51. Apply again the tcp filter in Wireshark. What is the difference between the web server's initial sequence number and the first ACK number sent by the client?
52. Undo the changes you have made in the Wireshark configuration: That is, **enable relative sequence numbers** in TCP and **enable HTTP** again. This is important because in the following steps we will always use relative sequence numbers.
53. In step 42 you wrote down the size of the A_PDU sent by the client. Each byte of that A_PDU is assigned a sequence number within the data flow that goes from the client to the server through the TCP connection. The client has used sequence number 0 to assign it to the SYN bit. From there it has started assigning sequence numbers to every byte of the A_PDU and finally, it has assigned a sequence number to the FIN bit.
54. What is the sequence number assigned by the client to its FIN bit?
55. What is the sequence number assigned by the server to its FIN bit?
56. Is it possible to know which of the two applications (the client or the server) has sent more data through the TCP connection simply by looking at the (relative) sequence number assigned by the client and server TCP entities to their own FIN bit?
    If so, what is the relationship between the sequence number (relative) of the FIN bit and the amount of application-level data sent over a TCP connection?
57. When TCP sends a segment with no data, it is forced to put a sequence number on the segment. That sequence number must be the sequence number that would be used if a segment were to be sent containing data (TCP_UD). You can check this in any of the segments with Len = 0. For example, frame 7 contains a segment with Len=0 sent by the server. Notice that the next segment sent by the server (frame 8) has the same sequence number.
58. Be careful because there is an exception to this “rule” about segments with Len=0. Segments with SYN or FIN bit set, count as if they were carrying one byte of data when, in fact, they are not. You can check this at the beginning or at the end of the TCP connection.
59. What does the TCP entity do when the application entity it serves gives it the order to close the TCP connection?
60. Which application entity (server or client) is the first to decide to close the connection?
61. Notice in the list of frames that when one of the TCP entities has sent a FIN segment, the other TCP entity also sends a FIN. Does it do it almost immediately or does it take a few seconds?

62. Look at frame 8. Notice the HTTP_PCI of the HTTP_PDU sent by the server application. What is the relationship between the precise moment in time the server closes the connection and that HTTP_PCI?
63. **Write down** the elapsed time between the client sending the SYN and receiving the SYN-ACK. That can give you an idea of the **RTT between client and server (round trip time)** when sending an empty T_PDU and returning an empty T_PDU.
64. **Write down** the RTT observed between the client sending the FIN and receiving the ACK of that FIN. Is it similar to the one in the previous section?
65. **Write down** the RTT between the GET sent by the client and the ACK received.
    Is it greater or less than the two RTTs you have previously calculated?
    Why do you think that happens?
66. The third TCP segment shown in the listing is an ACK segment. This ACK segment is used to acknowledge the arrival of the SYN-ACK segment to the server. Think what would have happened if this ACK got lost and did not reach the server. Note that after that segment, the TCP client sends a segment containing application data.
    Would this segment serve to acknowledge the arrival of the SYN segment to the server?
    If so ... Do you think that ACK would arrive on time or too late?
    Note: Compare the time when the third segment with the time when the fourth segment is sent.
67. Suppose we want to see, in an easy way, the length ("**length**") of the **application data** contained in the TCP segments. To do this, write the filter **tcp.len> 0** and apply it to show only the frames that contain TCP segments that contains application data. That is fine, but it is not exactly what we want because we want the list of frames to show the value of the **tcp.len**.
68. As we have seen in the previous lab session, Wireshark allows us to **customize the columns** of the frame list. **It allows us to show not only any header (PCI) field but also calculated fields** (i.e. fields that are not part of any PCI but calculated by Wireshark itself). This is the case of the tcp.len field because there is no such field within TCP_PCI called length. Tcp.len is calculated by Wireshark using other fields in any PCI.
69. We are going to add the tcp.len field as a custom column. Go to "Edit"> "Preferences", and within the "User Interface" branch we select "Columns" and then click in "Add" to add a new column. We change the "Title" and choose a meaningful name, for example tcp.len and press ENTER. Inside "Properties" we edit the "Field Type" so that it is "Custom" and now in "Field name" we put the name of the field that interests us (in this case tcp.len), and then close the window with "OK". Please notice that in the field name you may enter any name used in display filters.
70. The new column tcp.len has already been created, but it is located on the rightmost part of the central panel, just after the "Info" column. Try to drag it to the left, to an area that suits us better.
71. Write and apply the tcp filter again to see the 11 TCP segments and thus see the effect of the new column. It shows us the tcp.len field and allows us to quickly locate the segments with data and to know how much data they carry. It is important to highlight the fact that the **tcp.len field is NOT a field in the TCP header**. It is a value that Wireshark calculates from other existing fields in the TCP header and in the IP header.
72. Let's now see the quick way to **create a column that shows the value of a field**. Click on any frame that contains a TCP_PDU. Go into the details of that frame in the central panel and look in the TCP protocol information for the section "Flags" and then look for the flag named "Syn". You can add this field as a column quickly. Just, select the "Syn" flag and then, using the right button select the option "**Apply as Column**". When we do this, a new column appears in the list, named "Syn", which shows the status of that bit. We can easily see that only the first two frames in the list have the SYN bit set, as we already knew.
73. **Do the same with the TCP Acknowledgment flag** (not to be confused with the ACK number). Which segment, within the list of TCP_PDUs, is the only one that does not have the Acknowledgment flag set?
74. If you wanted to know which segments have a **TCP header of 20-bytes** and differentiate them from those that are longer, we could inspect, segment by segment through the list of frames, looking at the value from the "Header length" field in the TCP segment header. But the best thing is to take advantage of what we have learned and **create a column** that shows us the value of the TCP "**Header Length**" field. Do that and create a column with the "Apply as Column" technique and answer: Which segments have a header larger than normal, knowing that 20 bytes is "normal" (TCP header without options)?
75. Segments with a TCP header greater than 20 bytes are those that have options in the header. These options serve several things. An important option of the TCP header is the one that is used to inform the other entity of the value of the MSS ("Maximum Segment Size"). If this option is going to be used, it can only be set in the first segment that we send over the connection. Go into the details of the frames with TCP segments with options and answer:
    Has the client used the MSS option?
    And the server?

If so, what is its value in each case?
What other options (other than NOP) have the client and server used?

77. Before leaving, you must leave your PC turned off and connected to the ETSII network (as it was at the beginning) and you must also return the patch cord that you used to connect to the LAB_DTE network.