COMP 416 – Computer Networks

Project #2

Due: 06 December, 2020 @ 11:59pm (Late submissions will not be accepted).

Submission of the project deliverables is via Blackboard.

Note: This is an individual project.

Transport Layer and ARQ Protocols Analysis with Wireshark

This project is about the transport **layer** of the network protocol stack.

The focus is on the **SSL, TCP and UDP protocols**. For this purpose, you are asked to modify the provided SSL client/server codes as specified below, experiment with TCP and UDP features and implement a Stop-and-Wait ARQ protocol. You are asked to use the **WireShark network protocol analyzer** tool to answer transport layer related questions.

WireShark is the world's foremost network protocol analyzer, and is the **de facto standard** across many industries and educational institutions. It can be downloaded freely at **http://www.wireshark.org/download.html**. WireShark allows users to trace the network activity by capturing all the packets that hit your network interface. It tags the information of each layer by parsing the given byte stream according to the corresponding protocol.

You should read this project document carefully before starting your tasks.

Part 1 - SSL Implementation and Experiments:

Figure 1 illustrates an overview of SSL protocol. Recall that, you are provided an SSL client/server code that performs echoe on top of an SSL socket. The corresponding SSL practical content codes and slides are available through the course web site.

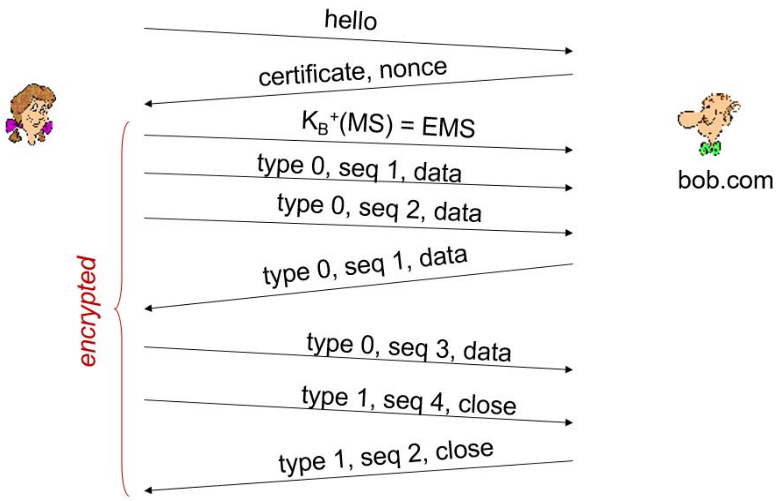


Figure 1. SSL Protocol Overview

As presented in the figure, the certificate is sent by the server to the user at the start of the session. The user adds this certificate to the local key store and uses it for authentication. The code is provided to add the certificate to the local key store, but the part where the server sends the certificate to the user is missing. In this part, you are asked to modify the provided code as follows:

- Set up a TCP connection on which the certificate will be transferred to the client. TCP connection can listen to any port. The server should ask for client verification before the certificate is transferred to the client. You can keep an already known users file on the server side and use them for login.
- Use the certificate to connect to the server through SSL. You should keep the certificate in the right directory.
- SSL connection at the server side should listen to the port with your KUSIS-ID + DD from your date of birth (DDMMYY) number (look at the first question). You may handle the case where the number becomes larger than the available ports by any mathematical manipulation and explain the answer in the report.
- After SSL connection is established, your client should receive your KUSIS-username (e.g. abcdef18) + KUSIS-ID character by character in separate messages in a non-persistent manner. Then, the KUSIS-username + KUSIS-ID should be printed at the client side.

Important Notes:

- Your modified SSL codes must be submitted along with your project report. In your project report, you should explain your answers and provide your WireShark outputs for each question, in order to get credit.
- On Windows, you might not be able to capture the Loopback interface, which is the traffic inside your operating system. When you run your server and client software in a single operating system, in order to capture incoming and outgoing packets, you need to capture the Loopback interface. To solve this problem, you can run one of the applications (client or server) in a separate machine.

After running your code, answer the following questions:

1. Locate the SSL Server IP address and port number, client IP address and port number through which these agents are communicating by using Wireshark.
2. Locate the data containing TCP segments. What is written in the data field? Compare it with the data you exchanged between the client and server. Why do you think is this the case?
3. How many TCP segments are transmitted in total while your KUSIS username + KUSIS ID is exchanged one by one with non-persistent connections?
4. What difference did you see in the payload of SSL and TCP? Can you locate the login name and password entered by the user? Can you locate the email information?

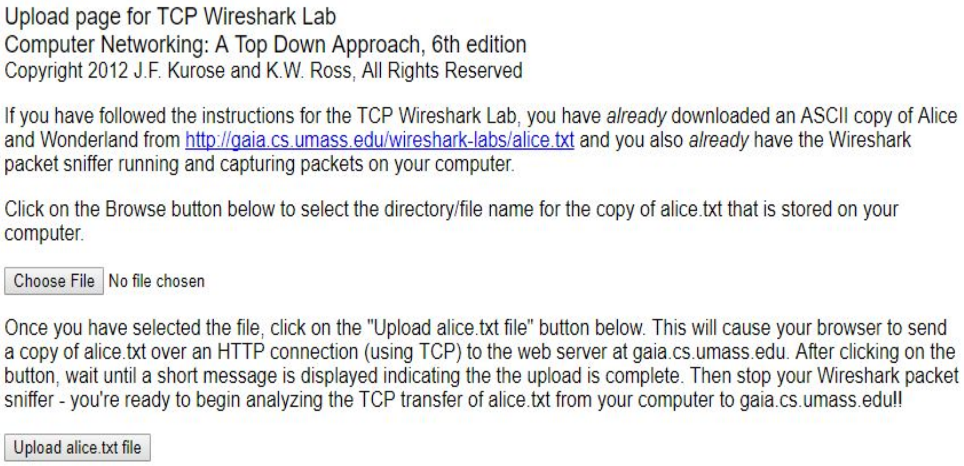
Part 2 - TCP Experiments:

Before beginning the exploration of TCP, you need to use Wireshark to obtain a packet trace of the TCP transfer of a file from your computer to a remote server.

You need to run Wireshark before starting this process to obtain the trace of the TCP segments sent and received from your computer.

You are asked to do so by accessing a Web page that will allow you to enter the name of a file stored on your computer (which contains the ASCII text of *Alice in Wonderland*), and then transfer the file to a Web server using the HTTP POST method. We are using the POST method rather than the GET method as we would like to transfer a large amount of data *from* your computer to another computer. Perform the following:

- Run Wireshark and start capturing the traffic.
- Start up your web browser. Go the http://gaia.cs.umass.edu/wireshark-labs/alice.txt and retrieve an ASCII copy of *Alice in Wonderland*. Store this file somewhere on your computer.
- Next go to http://gaia.cs.umass.edu/wireshark-labs/TCP-wireshark-file1.html
- You should see a screen that looks like:



Upload page for TCP Wireshark Lab
Computer Networking: A Top Down Approach, 6th edition
Copyright 2012 J.F. Kurose and K.W. Ross, All Rights Reserved

If you have followed the instructions for the TCP Wireshark Lab, you have *already* downloaded an ASCII copy of Alice and Wonderland from http://gaia.cs.umass.edu/wireshark-labs/alice.txt and you also *already* have the Wireshark packet sniffer running and capturing packets on your computer.

Click on the Browse button below to select the directory/file name for the copy of alice.txt that is stored on your computer.

Choose File No file chosen

Once you have selected the file, click on the "Upload alice.txt file" button below. This will cause your browser to send a copy of alice.txt over an HTTP connection (using TCP) to the web server at gaia.cs.umass.edu. After clicking on the button, wait until a short message is displayed indicating the the upload is complete. Then stop your Wireshark packet sniffer - you're ready to begin analyzing the TCP transfer of alice.txt from your computer to gaia.cs.umass.edu!!

Upload alice.txt file

- Use the *Browse button* in this form to enter the name of the file (full path name) on your computer containing *Alice in Wonderland* (or do so manually). Don't yet press the *"Upload alice.txt file"* button.
- Now start up Wireshark and begin packet capture *(Capture->Start)* and then press *OK* on the Wireshark Packet Capture Options screen (we will not need to select any options here).
- Returning to your browser, press the *"Upload alice.txt file"* button to upload the file to the gaia.cs.umass.edu.server. Once the file has been uploaded, a short congratulations message will be displayed in your browser window.
- Stop Wireshark packet capture.

Answer the following questions for the TCP segments:

5. Obtain the Flow Graph of the TCP communication. What is the significance of the various IP addresses shown in the Flow Graph? Using the flow graph, identify the three-way handshake and terminating handshake messages for the TCP connection. Provide screenshots for each explanation.
6. What are the sequence numbers (which appear in the Wireshark program) of the segments used for the 3-way handshake protocol that initiates the first TCP connection?

What are the sequence numbers of those segments and the port numbers used on client and server sides?

7. What is the sequence number of the SYNACK segment sent by gaia.cs.umass.edu to the client computer in reply to the SYN?
8. What is the value of the Acknowledgement field in the SYNACK segment? How did gaia.cs.umass.edu determine that value? What is it in the segment that identifies the segment as a SYNACK segment?
9. What is the sequence number of the TCP segment containing the HTTP POST command? Note that in order to find the POST command, you'll need to dig into the packet content field at the bottom of the Wireshark window, looking for a segment with a "POST" within its DATA field.
10. Consider the TCP segments containing the HTTP POST as the last segment in the TCP connection. What are the sequence numbers of the first six segments in the TCP connection? At what time was each segment sent? When was the ACK for each segment received? Given the difference between when each TCP segment was sent, and when its acknowledgement was received, what is the RTT value for each of the six segments? What is the **EstimatedRTT** value (see Section 3.5.3 in the textbook) after the receipt of each ACK? Assume that the value of the EstimatedRTT is equal to the measured RTT for the first segment, and then is computed using the EstimatedRTT equation (Section 3.5.3 in the textbook) for all subsequent segments.

    Note: Wireshark has a nice feature that allows you to plot the RTT for each of the TCP segments sent. Select a TCP segment in the "listing of captured packets" window that is being sent from the client to the gaia.cs.umass.edu server. Then select: *Statistics->TCP Stream Graph- >Round Trip Time Graph.*

Part 3 - UDP Experiments:

In this part, you are assigned a **unique** URL to work with. The list is provided in file Project2_URL_List.pdf, and you must use the URL assigned you. You should provide the appropriate screenshots and work on the correct domain, in order to get credit.

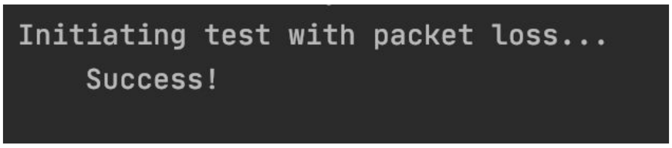
nslookup command works as an IP address resolver. When you provide a domain name as argument, it will return the IP address of that domain. Now, take the following steps provided and answer the questions accordingly.

- Start your Wireshark software and start capturing packets from the appropriate interface.
- Use nslookup command in order to resolve the IP address of the URL that is assigned to you.
- Stop packet capturing in the Wireshark.
- Apply an appropriate **display filter**.

11. What display filter did you apply in order to see appropriate packets?
12. Which application layer and transport layer protocol do nslookup work on? What is the reason that transport layer protocol is chosen?
13. Can you derive the local DNS server you connected work in iterative or recursive manner? If you can or cannot please provide a detailed explanation. Please also briefly explain the advantages and disadvantages of iterative and recursive approach over each other.
14. What are the header lengths of application layer protocol and transport layer protocol that nslookup works on?
15. How many checksums does an UDP segment have in the checksum field? Why?

Part 4.a - Stop-and-Wait ARQ Protocol

Recall the Stop-and-Wait ARQ protocols you have seen in the lectures. For this part, you are to implement a Stop-and-Wait ARQ protocol at the transport layer. We have already given you a base code in Java. **You must implement your methods over this code.** After your implementation is complete, you can run the main method of **Main.java** under the **main** package to see whether your code passes the tests. If you are having problems, you may set **Main.DEBUG** to true to see some more information on what is happening behind the scenes. You should see the following output when your implementation is completed successfully:



```
Initiating test with packet loss...
    Success!
```

Fig 1. The output you should be getting.

You only need to implement two methods residing in **Transport.java** under the **transport** package:

- **void sendWithARQ(Packet[] packets)**
  Used by the sender. Gets an ordered list of message packets and sends them one by one to the receiver, using Stop and Wait ARQ.
- **Packet[] receiveWithARQ()**
  Used by the receiver. Receives the message packets sent by the sender using Stop and Wait ARQ and returns them as an array.

While implementing your methods, you will need to use the methods and classes that are already provided to you. First, take a look at **Packet.java** under the **network** package. A packet can either be a message packet or an acknowledgement packet. Here are the important fields that you should be familiar with in your implementation: **ack, lastPacket, sequenceNumber, characters, timedOut**.

The methods that you should be using are in the **Transport.java** under the **transport** package. Here are the explanations:

- **Packet receivePacket(int timeout)**
  Receives a packet from the other process. If the timeout parameter is given as > 0, the process waits **timeout** milliseconds for a packet before failing. Upon timeout failure, this method will return an empty packet with its **timedOut** flag set to true. If the timeout parameter is given as ≤ 0, this method blocks infinitely.
- **void sendMsgPacket(int sequenceNumber, boolean lastPacket, Packet packet)**
  Sends a single message packet to the other process. The **sequenceNumber** parameter is either 0 or 1, and it denotes the sequence number of the message packet that you are sending. The **lastPacket** parameter should be set to true only when you are sending the last message packet to the receiver. The **packet** parameter is the actual message packet you are trying to send.
- **void sendAckPacket(int sequenceNumber, boolean lastAck)**
  Sends an acknowledgement packet to the other process. The **sequenceNumber** parameter is either 0 or 1, denoting the sequence number of the acknowledgement packet. The **lastAck** parameter should be set to true only when this acknowledgement message is being sent in response to the last message packet received from the sender.

In conclusion, your implementation should do the following:

- At the sender (i.e., **sendWithARQ** method):
    1. Sender must send each message packet using **sendMsgPacket**, setting the arguments correctly.
    2. After a packet is sent, the sender must wait for an acknowledgement from the receiver using **receivePacket** with a reasonable timeout of your choice.
    3. If the acknowledgement is not received within this duration, the sender must retransmit the packet.
    4. If the acknowledgement was received within this duration and the sequence number is as expected, the sender must proceed to send the next packet.
- At the receiver **receiveWithARQ**:
    1. Receiver must wait for packets using **receivePacket**.
    2. Once a packet is received, the receiver should check the sequence number (using **Packet.sequenceNumber**) and discard the packet or save it depending on its sequence number.
    3. Receiver should also check **Packet.lastPacket** field of the received message packet.
        - **If the received packet is not the last message packet**: The receiver must send a single acknowledgement packet with the correct sequence number using **sendAck**, and then wait for the next message packet.
        - **If the received packet is the last message packet**: If this is the last packet, we will send our *last acknowledgement* using **sendAck** and break out of our loop.

Part 4.b - Analyzing Your Implementation

After your implementation is complete, please answer the following questions in your report.

16. Start listening on your Loopback interface with Wireshark and run your code. How many retransmissions have occurred from the sender to the receiver? Explain.
17. Attach a screenshot of the first packet sent by the sender and the last packet received by the receiver. How long did it take to transfer all the packets? Consider the timeout period that you have chosen and the amount of retransmissions. Does this result make sense? Why or why not?

Note: You will not see the “lost packets” on Wireshark.

Project Deliverables:

Important Note: You are expected to submit a project report, in PDF format, that documents and explains all the steps you have performed in order to achieve the assigned tasks of the project. A full grade report is one that clearly details and illustrates the execution of the project. Anyone who follows your report should be able to reproduce your performed tasks without effort. Use screenshots to illustrate the steps and provide clear and precise textual descriptions as well. All reports would be analyzed for plagiarism. Please be aware of the KU Statement on Academic Honesty.

The name of your project .zip file must be <surname>-<KUSIS-id>.zip
You should turn in a single .zip file including:

- Source codes: Containing the source codes of client and server, and also your completed version of part 4 code.

- Project.pdf file (Your report, should include answers and the corresponding Wireshark screenshots)

- Saved capture files from the Wireshark.

Figures in your report should be scaled to be visible and clear enough. All figures should have captions, should be numbered according to their order of appearance in the report, and should be referenced and described clearly in your text. All pages should be numbered, and have headers the same as your file naming criteria.
If you employ any (online) resources in this project, you must reference them in your report.
There is no page limit for your report, and no specific requirements on the design.

Good Luck!