CS 315: Computer Networks Lab
Spring 2024-25, IIT Dharwad
Assignment-6
UDP, Socket Programming & SMTP
February 10, 2025

Lab Instructions

- Login to the Ubuntu OS on your machine. The login credentials are as follows:
  - Username: user
  - Password: 123456
- Mark your attendance in the attendance sheet before leaving the lab.
- Handle the lab resources with utmost care.
- Please go through the following exercises in today's lab.
- It is recommended that you complete all the following exercises during the lab slot itself.
- If you face any difficulties, please feel free to seek help online or from your peers or TAs.
- After finishing all exercises, please carry your solutions with you (via email/pen drive) for future reference, and delete the files from the desktop.

Introduction

In this lab, we'll take a quick look at the UDP transport protocol. UDP is a streamlined, no-frills protocol. Because UDP is simple and sweet, we'll be able to cover it pretty quickly in this lab. At this stage, you should be a Wireshark expert. Thus, we are not going to spell out the steps as explicitly as in earlier labs.

Part 0: Paste a screenshot of your system IP address, using `ipconfig` (on Windows) or `ifconfig` (on Mac and Linux), and fill out this Google form to submit the details of your system. The same system must be used to attempt all exercises of this lab.

Part-1: Wireshark UDP

Start capturing packets in Wireshark and then do something that will cause your host to send and receive several UDP packets. It's also likely that just by doing nothing (except capturing packets via Wireshark) some UDP packets sent by others will appear in your trace. In particular, the Domain Name System (DNS) protocol typically sends DNS query and response messages inside of UDP, so you'll likely find some DNS messages (and therefore UDP packets) in your trace.

Specifically, you can try out the `nslookup` command, which invokes the underlying DNS protocol, which in turn will send UDP segments from/to the host issuing the `nslookup`.

`nslookup` is available in most Microsoft, Apple IOS, and Linux operating systems. To run `nslookup` you just type the `nslookup` command on the command line in a DOS window, Mac IOS terminal window, or Linux shell.

Do the following:

1. Start Wireshark
2. Flush DNS cache in terminal
3. Use nslookup on `www.woodenstreet.com` domain
4. Stop the Wireshark.

Answer the following questions:

1. List the different protocols you observe in the trace that use the UDP in the transport layer.
   Hint: Select any packet you observe using UDP from the packet listing window then select Apply as Column

We don't need to go into any more details about `nslookup` or DNS, as we're just interested in getting a few UDP segments into Wireshark. Pick the **first UDP segment for the above hostname (by applying DNS in the filter) and expand the UDP fields in the details window.**

2. State the total number of fields you observe in the UDP and list them. State the size of each of those fields.
3. What does the Length field signify in the UDP?
4. What is the maximum number of bytes that can be included in a UDP payload?
5. What is the protocol number for UDP? Give your answer in decimal notation. To answer this question, you'll need to look into the Protocol field of the IP datagram containing this UDP segment.

Part-2: Socket Programming

Please refer to the following document for the basics of socket programming.
In this assignment, you will implement a client-server application using Python's socket programming library. The objective is to demonstrate your understanding of socket communication, file transfer, and data processing.

Client-Server Interaction Overview

1. The client will send a text file (alice.txt) to the server.
2. The server will:
   a. Receive the file.
   b. Extract the first 10 lines and the last 10 lines from the file.
   c. Return these lines to the client.

3. The client will display the received lines (first and last 10 lines) on the terminal.

Client and Server Requirements

Server (server.py)

1. Bind to a specific port and listen for incoming client connections.
2. Accept the connection from the client.
3. Receive the file from the client in chunks.
4. Read the received file, then extract:
    a. The first 10 lines.
    b. The last 10 lines.
5. Send the extracted lines (first and last 10 lines) back to the client.
6. Close the connection after processing the file.

Client (client.py)

1. Create a socket and connect to the server.
2. Read the file alice.txt (download it from here).
3. Send the file to the server in binary mode, in chunks.
4. Wait for the server's response containing the first 10 and last 10 lines of the file.
5. Display the received data (the first and last 10 lines) on the terminal.

Detailed Instructions

Server Implementation:

1. The server must bind to a specific port (e.g., 12345) and listen for incoming connections.
2. After accepting a connection, the server will receive the file sent by the client. The file will be received in chunks and stored as received.txt.
3. The server will then read the file, and extract the first 10 lines, and the last 10 lines.
4. The extracted lines will be sent back to the client as a response.
5. Ensure that the server gracefully closes the socket connection after processing.

Client Implementation:

1. The client must create a socket and connect to the server using the server's IP address and port number.
2. The client will read the file alice.txt in binary mode and send it to the server in chunks using the sendall() method.
3. After sending the entire file, the client will wait for the response from the server, which will contain the first and last 10 lines of the file.
4. The client will display the received lines (first 10 and last 10) on the terminal.

Part-3: Socket Programming: SMTP

In this assignment, you will acquire a better understanding of SMTP protocol. You will also gain experience in implementing a standard protocol using Python.

Your task is to develop a simple mail client that sends email to any recipient. Your client will need to connect to a mail server, dialogue with the mail server using the SMTP protocol, and send an email message to the mail server. Python provides a module, called smtplib, which has built in methods to send mail using SMTP protocol. However, we will not be using this module in this lab, because it hide the details of SMTP and socket programming.

In order to limit spam, some mail servers do not accept TCP connection from arbitrary sources. For the experiment described below, you may want to try connecting both to your university mail server and to a popular Webmail server, such as an AOL mail server. You may also try making your connection both from your home and from your university campus.

Code

Below you will find the skeleton code for the client. You are to complete the skeleton code. The places where you need to fill in code are marked with **#Fill in start** and **#Fill in end**. Each place may require one or more lines of code.

Additional Notes

In some cases, the receiving mail server might classify your e-mail as junk. Make sure you check the junk/spam folder when you look for the e-mail sent from your client.

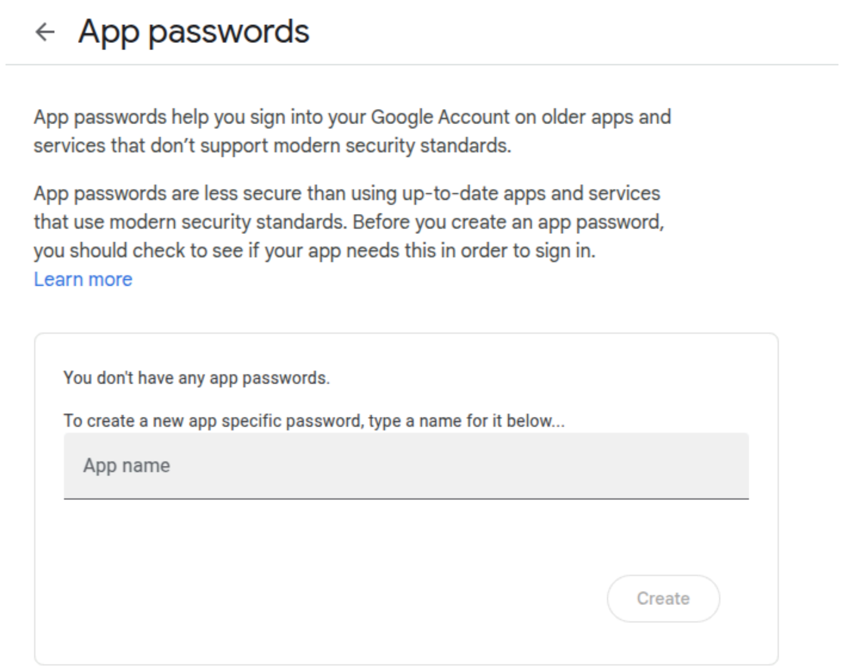
The **userEmail** & **userPassword**: Credentials for authentication. Ideally, **userPassword** should be an App Password (not a regular Gmail password) as mentioned in the skeleton below.

Steps to generate the App Password:

1. Open your `Google Account Security` settings:

   `https://myaccount.google.com/security`
2. Scroll down to "`How you sign in to Google`".
3. Click on "`2-Step Verification`" and follow the setup process if it is not already enabled.
4. In your `Google Account Security`, in the search bar find `App passwords` you will see a window as shown below. Now, create a new app specific password, `App Name: Name`, then click on the `create` button
5. The Google will generate a 16-character password (e.g., abcd efgh ijkl mnop).

   copy it and use it in your skeleton below instead of your regular Gmail password.



App passwords

App passwords help you sign into your Google Account on older apps and services that don't support modern security standards.

App passwords are less secure than using up-to-date apps and services that use modern security standards. Before you create an app password, you should check to see if your app needs this in order to sign in.
Learn more

You don't have any app passwords.

To create a new app specific password, type a name for it below...

Skeleton Python Code for the Mail Client

```
#add in prompt
userEmail = "your-email@iitdh.ac.in"
userPassword = "abcd efgh ijkl mnop"   # Use the generated app password
userDestinationEmail = input("Enter Email Destination: ")
userSubject = input("Enter Subject: ")
userBody = input("Enter Message: ")
msg = '{}.\r\n I love computer networks!'.format(userBody)

# Choose a mail server (e.g. Google mail server) and call it mailserver
#Fill in start

#Fill in end

# Create socket called clientSocket and establish a TCP connection with mailserver
#Fill in start

#Fill in end

recv = clientSocket.recv(1024).decode()
```

```
print(recv)
if recv[:3] != '220':
	print('220 reply not received from server.')

# Send HELO command and print server response.
heloCommand = 'HELO Alice\r\n'
clientSocket.send(heloCommand.encode())
recv1 = clientSocket.recv(1024).decode()
print(recv1)
if recv1[:3] != '250':
	print('250 reply not received from server.')

#account authentication
clientSocket.send("STARTTLS\r\n".encode())
clientSocket.recv(1024)
sslClientSocket = ssl.wrap_socket(clientSocket)
sslClientSocket.send("AUTH LOGIN\r\n".encode())
print(sslClientSocket.recv(1024))
sslClientSocket.send(b64encode(userEmail.encode()) + "\r\n".encode())
print(sslClientSocket.recv(1024))
sslClientSocket.send(b64encode(userPassword.encode()) + "\r\n".encode())
print(sslClientSocket.recv(1024))

# Send MAIL FROM command and print server response.
#Fill in start

#Fill in end

# Send RCPT TO command and print server response.
#Fill in start

#Fill in end

# Send DATA command and print server response.
#Fill in start

#Fill in end

# Send message data.
#Fill in start
```

```
#Fill in end

# Message ends with a single period.
#Fill in start

#Fill in end

# Send QUIT command and get server response.
#Fill in start

#Fill in end
```

Submission Details

- **Part-1:** Write your answers wireshark UDP in a single doc/tex file, and submit its PDF named after your IIT Dharwad roll number, which contains all answers (with screenshots).
- **Part-2:** Submit the both server and client files with your roll number prefixed as **`<Roll_number>_server.py`** and **`<Roll_number>_client.py`**
- **Part-3:** Submit the client code with your roll number prefixed as **`<Roll_number>_client.py`** and attach the screenshots at the recipient inbox proving the successful mail delivery.