CAH11-01 Networks and Operating Systems

Transport Layer Part 2

Aims of the Seminar

Welcome to the Python Workshop on the TCP Transport Layer! This workshop focuses on the Transmission Control Protocol (TCP), a connection-oriented, reliable protocol that ensures data integrity and ordered delivery. By the end of this session, you will:

1. Understand TCP's role in the transport layer (connection management, flow control, error recovery).
2. Build TCP client-server applications in Python using the socket library.
3. Handle multiple clients concurrently using threading.
4. Implement practical use cases like file transfer and encrypted messaging.

Workshop Outline

1. TCP Basics: Setting up a TCP client-server connection.
2. Reliable Data Transmission: Sending and receiving streams of data.
3. Handling Multiple Clients: Using threading for concurrency.
4. File Transfer Application: Transferring files over TCP.
5. API Integration: Fetching and sending data from web APIs.
6. Security: Adding encryption and authentication.

Prerequisites

- Basic Python programming skills.
- Familiarity with IP addresses, ports, and networking fundamentals.

Feel free to discuss your work with peers, or with any member of the teaching staff.

Reminder

We encourage you to discuss the content of the workshop with the delivery team and any findings you gather from the session.

Workshops are not isolated, if you have questions from previous weeks, or lecture content, please come and talk to us.

Exercises herein represent an example of what to do; feel free to expand upon this.

Helpful Resources

- Python socket Library Documentation
- TCP Protocol RFC 793
- Real Python Socket Guide
- https://www.geeksforgeeks.org/multithreading-python-set-1/
- Cryptography Library Tutorial

Week Recap Quiz

To Access the week quiz via:
https://learn.gold.ac.uk/mod/quiz/view.php?id=1621521

Exercises

You may find it useful to keep track of your answers from workshops in a separate document, especially for any research tasks.
Where questions are asked of you, this is intended to make you think; it would be wise to write down your responses formally.

1. Building a Simple TCP Server

Objective: Create a TCP server to accept client connections and echo received messages.

Code Example:

```python
import socket

# Create a TCP socket
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind(('localhost', 65432))
server_socket.listen(1)  # Allow 1 pending connection

print("TCP Server is listening...")

while True:
    client_socket, client_address = server_socket.accept()
    print(f"Connected to {client_address}")

    data = client_socket.recv(1024)
    print(f"Received: {data.decode()}")

    # Echo back the data
    client_socket.sendall(b"ACK: " + data)
    client_socket.close()
```

- socket.SOCK_STREAM specifies TCP.
- listen(1) enables the server to accept connections (1 queued connection).
- accept() blocks until a client connects, returning a new socket for that client.
- sendall() ensures all data is sent (unlike send()).

2. Building a Simple TCP Client

Objective: Connect to the server and send a message.

➔ **You must use a new window in your vs code (not new tab)**

Code Example:

```
import socket

client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect(('localhost', 65432))

message = input("Enter message: ")
client_socket.sendall(message.encode())

response = client_socket.recv(1024)
print(f"Server response: {response.decode()}")

client_socket.close()
```

Exercise 1: **Use python datetime library to measure how long it took to send the data. To do so you need to import it**

```
import datetime
```

Then you can use:

```
datetime.datetime.now()
```

To get the current time

Exercise 2: (🤔) Change your script so that the data will be send using UDP and compare the time needed to send it using both protocols.

3. Logging data in TXT File Over TCP

Objective: Log the sent data from client to server.
Change the server script so that the received data will be saved in a txt file for logging.

➔ Make sure you restart the server kernel to be able to run the new version of the script

The following script show how you can save data in txt file. You need to incorporate it in the server script.

File write Code:

```
# ... (Server setup code)
with open('received_file.txt', 'wb') as f:
    while True:
        data = client_socket.recv(1024)
        if not data:
            break
        f.write(data)
print("File received!")
```

3. File Transfer Over TCP

Objective: Transfer a file from client to server.

➔ **Create a txt file with your name in the same folder as your notebook and name it** `‘file_to_send.txt’`

The following script show how you can read data from txt file. You need to incorporate it in the client script.

File reading Code:

```
# ... (Client setup code)
with open('file_to_send.txt', 'rb') as f:
    client_socket.sendfile(f)
```

Exercise 3: (🤔) Change your txt file to be a whole paragraph and compare the time needed to send it.

Exercise 4: (🤔) Resend the new txt file using UDP and compare the time needed.

5. (Challenge 🤔) **TCP-based chat system**

Work in pairs to design a TCP-based chat system where a central server relays messages between multiple clients in real time.

1. **Server Responsibilities:**
   - Accept and manage simultaneous client connections.
   - Forward messages from one client to all others (broadcast).
   - Track active users and handle disconnections gracefully.
2. **Client Responsibilities:**
   - Connect to the server and send messages.
   - Receive and display messages from other clients.

Hint: To enable the server to receive connections from multiple clients simultaneously, we need to use threading. You can refer to this tutorial to learn about threading in Python:

(https://www.geeksforgeeks.org/multithreading-python-set-1/)

Exercise 5: (Challenge 🤔) Implement a TCP server that can connect to multiple clients and receive data from them simultaneously.

Exercise 6: (Challenge 🤔) Implement chat application for 2 clients

Exercise 7: Add encryption for secure messaging (use the cryptography library).

Exercise 8: Fetch weather data from an API and send it over TCP.

```
api_url = "https://api.open-meteo.com/v1/forecast?latitude=51.47&longitude=-0.0363&current_weather=true"
```

Happy coding! 🚀