CAH11-01 Networks and Operating Systems
Transport Layer in Networks PT-1

Aims of the Seminar

Welcome to the Python Workshop on the Transport Layer in Networks! This workshop is designed for you to learn about the transport layer of the TCP/IP model and how to implement basic network applications using Python. By the end of this workshop, you will have a solid understanding of the transport layer, and you will have built a few simple applications.

The Transport Layer plays a crucial role in computer networks by managing end-to-end communication between devices. It provides essential services such as segmentation, error control, flow control, and connection management. Unlike the more reliable TCP, the User Datagram Protocol (UDP) is a lightweight, connectionless protocol that prioritises speed over reliability, making it suitable for time-sensitive applications like live streaming and gaming.

In this lab, you will explore the fundamentals of UDP communication using Python. You will create simple client-server applications, work with API data, and implement basic error handling techniques. Through hands-on exercises, you will gain practical insights into how UDP operates and its trade-offs compared to TCP. By the end of the lab, you will be better equipped to understand the importance of transport protocols in real-world networking scenarios.

Workshop Outline:
- Setting up UDP client-server communication using the `socket` library.
- Transmitting and receiving data over UDP.
- Fetching data from APIs using the `requests` library and transmitting it via UDP.
- Handling communication errors and implementing simple retransmission logic.

Prerequisites

- Basic understanding of Python programming.
- Familiarity with basic networking concepts (e.g., IP addresses, ports).

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

The official Python documentation provides detailed information about the socket library: https://docs.python.org/3/library/socket.html

Socket Programming in Python (Guide) https://realpython.com/python-sockets/

A Complete Guide to Socket Programming in Python https://www.datacamp.com/tutorial/a-complete-guide-to-socket-programming-in-python

Python Requests: https://www.w3schools.com/python/module_requests.asp

Open-Meteo https://open-meteo.com/

1. Building a Simple UDP Server

Objective:
Create a simple Python script that acts as an UDP server to receive client messages.

Code Example:

```python
import socket

server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
server_socket.bind(('localhost', 65433))

print("UDP Server is ready to receive API data...")

while True:
    data, client_address = server_socket.recvfrom(2048)
    print(f"Received data from {client_address}: {data.decode()}")
```

Explanation:

```python
import socket
```

- This imports Python's built-in socket module
- Provides low-level networking interface
- Allows creating network connections and working with network protocols

```python
server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
```

Creates a new socket object

```python
socket.AF_INET
```

Specifies IPv4 addressing (Internet Protocol version 4)

```python
socket.SOCK_DGRAM
```

Indicates UDP (User Datagram Protocol) socket

- Connectionless protocol
- Sends packets without establishing a continuous connection
- Faster but less reliable compared to TCP

```python
server_socket.bind(('localhost', 65433))
```

- Binds the socket to a specific network interface and port
- 'localhost' means it will listen on the local machine (127.0.0.1)
- 65433 is the port number
- Any client trying to send UDP packets to this IP and port will reach this server

```
while True:
```

Creates an infinite loop to continuously listen for incoming UDP packets

```
    data, client_address = server_socket.recvfrom(2048)
```

recvfrom() is a UDP-specific method that:

- Waits for an incoming packet
- Returns two things:
    1. The received data (up to 2048 bytes)
    2. The address of the sender (IP and port)

```
    print(f"Received data from {client_address}: {data.decode()}")
```

- Prints the sender's address
- data.decode() converts received bytes to a readable string

Note: Do not close the server window keep it open to receive data from the client.

2. Building a Simple UDP Client

Objective:

Create a simple Python script that acts as an UDP client sending messages.

Code Example:

Start a new Vs code window and type the following script:

```
import socket

client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

server_address = ('localhost', 65433)
message = b"Hello, UDP Server!"

# Send message to the server
client_socket.sendto(message, server_address)

client_socket.close()
```

Once you run the script open the server window and check if it was received.

Exercise 1: (🤔) Using the previous two scripts make a chat application in which clients can chat using a server.

Exercise 2: Let the use make a dictionary with the users IP address

Exercise 3: Add authentication step to the application in which the sever would ask the client for username and password to initiate the communication.

Exercise 4: (🤔) Write functions that would encrypt the messages in the client and decrypt them in the server. Hint: you can refer to this tutorial https://www.tutorialspoint.com/cryptography_with_python/cryptography_with_python_quick_guide.htm

3. Building a Simple API data collection Application

Objective:
Create an application that would collect weather information from API and send it to the server.

Code Example:

```
import socket
import requests

# Fetch weather data
api_url = "https://api.open-meteo.com/v1/forecast?latitude=51.47&longitude=-
0.0363&current_weather=true"
response = requests.get(api_url)

if response.status_code == 200:
    weather_data = response.json()
    temperature = weather_data["current_weather"]["temperature"]
    message = f"Current temperature: {temperature}°C"
else:
    message = "Failed to fetch weather data"

# Send the weather data using UDP
client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
server_address = ('localhost', 65433)

client_socket.sendto(message.encode(), server_address)
print("Weather data sent!")

client_socket.close()
```

Exercise 5: Check the API website and update the script to compare the temperate between the university and the British Library.

Exercise 6: Experiment with other applications of the API.

Conclusion

Congratulations! You've completed the Python Workshop on the Trasport Layer Part1. You've learned how to build client, server network and applications using Python, and you've tackled some challenging exercises to further your understanding. Keep experimenting and building more complex applications to deepen your knowledge of networking and Python.

Happy coding! 🚀