CS212 - Computer Networks

Name - ANIKET CHAUDHRI
ADARSH ANAND
Date - Sep 15, 2022
Course instructor - Neha Karanjkar
TA - Tushar Lone

LAB 3: Socket Programming

Part 1:

Ans1.

```
aniket@aniket:~/Documents/lectures/cs212/lab3$ python3 UDP_Server.py
Server started...waiting for a connection from the client
Server received MESSAGE= I could tell you a UDP joke but I'm not sure you'll get
 it. from ADDR= ('127.0.0.1', 44332)
```

Fig - Code for UDP_Server.py that receives message from UDP_Client.py

```
aniket@aniket:~/Documents/lectures/cs212/lab3$ python3 UDP_Client.py
Client received MESSAGE= Ok  from ADDR= ('127.0.0.1', 43387)
aniket@aniket:~/Documents/lectures/cs212/lab3$
```

Fig - Code for UDP_Client.py that sends message to UDP_Server.py and receives confirmation

Ans3.

```
'''
Member 1: Aniket Akshay Chaudhri (2003104)
Member 2: Adarsh Anand (2003101)
'''
# socket program to send the current date and time to the client

import socket
import sys
import random
import time

# create a socket
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

host = 'localhost'

# bind the socket to a port
port = 43387

s.bind((host, port))

while True:
    # receive data from the client
    data, addr = s.recvfrom(1024)  # buffer size is 1024 bytes
    print("Server received MESSAGE=", data.decode(), " from ADDR=", addr)

    if data.decode() == "SEND_DATE":
        # send some bytes (encode the string into Bytes first)
        message = time.strftime("%d/%m/%Y")
        s.sendto(message.encode('utf-8'), (addr[0], addr[1]))
    elif data.decode() == "SEND_TIME":
        # send some bytes (encode the string into Bytes first)
        message = time.strftime("%H:%M:%S")
        s.sendto(message.encode('utf-8'), (addr[0], addr[1]))
    elif data.decode() == "quit":
        break

# close the socket
s.close()
```

Fig - Code for UDP_Server.py to answer queries of client

```python
'''
Member 1: Aniket Akshay Chaudhri (2003104)
Member 2: Adarsh Anand (2003101)
'''
# socket program to request the current date and time from the server repeatedly until the user enters "quit"

import socket
import sys
import random
import time

# create a socket
s=socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

host='localhost'
port=43387  # this is the server's port number, which the client needs to know

messages = ["SEND_DATE", "SEND_TIME"]

while True:
    # send some bytes (encode the string into Bytes first)
    time.sleep(random.uniform(1,2))
    message = random.choice(messages)
    s.sendto( message.encode('utf-8'), (host,port))

    # see if the other side responds
    data, addr = s.recvfrom(1024) # buffer size is 1024 bytes
    print("Client received MESSAGE=",data.decode()," from ADDR=",addr)

    if message == "quit":
        break

# close the connection
s.close()
```

Fig - Code for UDP_Client.py to send multiple random queries to UDP_Server.py

```
aniket@aniket:~/Documents/lectures/cs212/lab3$ python3 UDP_SendDateTime_Server.p
y
Server received MESSAGE= SEND_DATE  from ADDR= ('127.0.0.1', 48998)
Server received MESSAGE= SEND_TIME  from ADDR= ('127.0.0.1', 48998)
Server received MESSAGE= SEND_DATE  from ADDR= ('127.0.0.1', 48998)
Server received MESSAGE= SEND_DATE  from ADDR= ('127.0.0.1', 48998)
Server received MESSAGE= SEND_TIME  from ADDR= ('127.0.0.1', 48998)
Server received MESSAGE= SEND_TIME  from ADDR= ('127.0.0.1', 48998)
Server received MESSAGE= SEND_DATE  from ADDR= ('127.0.0.1', 48998)
Server received MESSAGE= SEND_TIME  from ADDR= ('127.0.0.1', 48998)
Server received MESSAGE= SEND_DATE  from ADDR= ('127.0.0.1', 48998)
Server received MESSAGE= SEND_TIME  from ADDR= ('127.0.0.1', 48998)
Server received MESSAGE= SEND_TIME  from ADDR= ('127.0.0.1', 48998)
Server received MESSAGE= SEND_DATE  from ADDR= ('127.0.0.1', 48998)
```

Fig - Output for UDP_Server.py

```
aniket@aniket:~/Documents/lectures/cs212/lab3$ python3 UDP_SendDateTime_Client.p
y
Client received MESSAGE= 15/09/2022  from ADDR= ('127.0.0.1', 43387)
Client received MESSAGE= 13:43:23  from ADDR= ('127.0.0.1', 43387)
Client received MESSAGE= 15/09/2022  from ADDR= ('127.0.0.1', 43387)
Client received MESSAGE= 15/09/2022  from ADDR= ('127.0.0.1', 43387)
Client received MESSAGE= 13:43:27  from ADDR= ('127.0.0.1', 43387)
Client received MESSAGE= 13:43:29  from ADDR= ('127.0.0.1', 43387)
Client received MESSAGE= 15/09/2022  from ADDR= ('127.0.0.1', 43387)
Client received MESSAGE= 13:43:33  from ADDR= ('127.0.0.1', 43387)
Client received MESSAGE= 15/09/2022  from ADDR= ('127.0.0.1', 43387)
Client received MESSAGE= 13:43:36  from ADDR= ('127.0.0.1', 43387)
Client received MESSAGE= 13:43:37  from ADDR= ('127.0.0.1', 43387)
Client received MESSAGE= 15/09/2022  from ADDR= ('127.0.0.1', 43387)
```

Fig - Output for UDP_Client.py

Part 2.

Ans 4. TCP Client and Server processes

```
aniket@aniket:~/Documents/lectures/cs212/lab3$ python3 TCP_Server.py
Server started...waiting for a connection from the client
Connection initiated from  ('127.0.0.1', 49084)
SERVER RECEIVED:  Knock knock..
```

Fig - Output for TCP_Server.py

```
aniket@aniket:~/Documents/lectures/cs212/lab3$ python3 TCP_Client.py
CLIENT RECEIVED:  Whose there?
```

Fig - Output for TCP_Client.py

Ans 5. Output of wireshark

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	TCP	74	49086 → 43389 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM…
2	0.000010677	127.0.0.1	127.0.0.1	TCP	74	43389 → 49086 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=6549…
3	0.000017984	127.0.0.1	127.0.0.1	TCP	66	49086 → 43389 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=2772574…
4	0.000036113	127.0.0.1	127.0.0.1	TCP	79	49086 → 43389 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=13 TSval=2…
5	0.000038336	127.0.0.1	127.0.0.1	TCP	66	43389 → 49086 [ACK] Seq=1 Ack=14 Win=65536 Len=0 TSval=277257…
6	0.000133585	127.0.0.1	127.0.0.1	TCP	78	43389 → 49086 [PSH, ACK] Seq=1 Ack=14 Win=65536 Len=12 TSval=…
7	0.000136489	127.0.0.1	127.0.0.1	TCP	66	49086 → 43389 [ACK] Seq=14 Ack=13 Win=65536 Len=0 TSval=27725…
8	0.000147736	127.0.0.1	127.0.0.1	TCP	66	43389 → 49086 [FIN, ACK] Seq=13 Ack=14 Win=65536 Len=0 TSval=…
9	0.000207754	127.0.0.1	127.0.0.1	TCP	66	49086 → 43389 [FIN, ACK] Seq=14 Ack=14 Win=65536 Len=0 TSval=…
10	0.000212115	127.0.0.1	127.0.0.1	TCP	66	43389 → 49086 [ACK] Seq=14 Ack=15 Win=65536 Len=0 TSval=27725…

The first 3 are the 3-way TCP handshake.
Here Port of Server = 43389 and Client =49086

Steps involved in entire process

1. Client sends Server a [SYN] request to initiate connection
2. Server responds back with [SYN-ACK] to acknowledge the request
3. Client sends [ACK] to acknowledge the response
4. Client sends [PSH, ACK] to Server to send some data
5. Server sends [ACK] to acknowledge the data

6. Server sends [PSH, ACK] to Client to send data
7. Client sends [ACK] to acknowledge the data
8. Server sends [FIN, ACK] to Client to close the connection
9. Client sends [FIN, ACK] to Server to close the connection
10. Server sends [ACK] to acknowledge the request

- Yes the contents of the message are visible as the data is unencrypted.
- If we inspect the TCP packet using Wireshark, we can see the data in the packet in the data section.

Ans 6.

```
aniket@aniket:~/Documents/lectures/cs212/lab3$ python3 TCP_Arithmetic_Server.py
Server started...waiting for a connection from the client
Connection initiated from  ('127.0.0.1', 43058)
SERVER RECEIVED:  5 + 2
SERVER RECEIVED:  34 * 76
SERVER RECEIVED:  2 - 4
SERVER RECEIVED:  2 * 1.4
SERVER RECEIVED:  q
Session ended
```

```
aniket@aniket:~/Documents/lectures/cs212/lab3$ python3 TCP_Arithmetic_Client.py
Enter your name: Aniket
CLIENT RECEIVED:  Hello Aniket
Enter an expression: 5 + 2
CLIENT RECEIVED:  7
Enter an expression: 5 / 0
Invalid expression
Enter an expression: 34 * 76
CLIENT RECEIVED:  2584
Enter an expression: 3/3
Invalid expression
Enter an expression: 2 - 4
CLIENT RECEIVED:  -2
Enter an expression: 2 * 1.4
CLIENT RECEIVED:  2.8
Enter an expression: q
CLIENT RECEIVED:  Bye
```

Ans 7.

- **Is the application-layer protocol designed by you for question 3 Stateless?** We used UDP sockets (Transport Layer) in question 3. It is a stateless protocol since it does not maintains a state of previous requests
- **Is the application-layer protocol designed by you for question 6 Stateless?** It is a stateful protocol since it does maintains a state of previous requests
- **Is TCP a Stateless protocol?**

TCP is Stateful protocol because it expects something like message acknowledgement, tracks conversation so that it can retransmit some data etc.

- **Is UDP a Stateless protocol?**
  UDP is a Stateless protocol because it does not expect anything, it does not care if the data sent was received as whole, does not wait for acknowledgement etc.

Ans 8.
Done

References

- Stateless protocol. (2022, February 6). In *Wikipedia*. https://en.wikipedia.org/wiki/Stateless_protocol
-