Project #1 Report – Part 1: iPerf Server

Running our code

Please open two terminal windows and run

```
python3 udp_servervictorialam_919626106_jinsiguo_[student_id2].py
```

on one window then

```
python3 udp_clientvictorialam_919626106_jinsiguo_[student_id2].py <size in bytes>
e.g.
python3 udp_clientvictorialam_919626106_jinsiguo_[student_id2].py 25
```

in the other window. The information that is required according to the document is all listed at the end of execution for readability.

Original Code

The code that we wrote without using ChatGPT (though we did use the Google AI assistant for smaller-scale implementation questions) already worked well and the client and server were able to communicate. However, something that we had trouble solving was how to include the entire data packet in the throughput calculation.

udp_clientvictorialam_919626106_jinsiguo_918709406.py:

```
import socket
import sys
import time

# Source: https://realpython.com/python-sockets/
SERVER_IP = "127.0.0.1"  # localhost
SERVER_PORT = 5555  # arbitrarily chosen socket

CLIENT_IP = "127.0.0.1"
CLIENT_PORT = 7777

# Validate input
if len(sys.argv) != 2:
    print("Usage: udp_xyzabc.py [size in MB to send]")
    exit(1)
if sys.argv[1].isdigit() == False:
    print("Not a number!")
    exit(1)
```

```
size_mb = int(sys.argv[1])
if (size_mb < 25 or size_mb > 200):
    print("size must be between 25 and 200")
    exit(1)


# Generate payload of size "size_mb"
start = time.time()
payload = bytearray(size_mb * 1000000)
for i in range(size_mb * 1000000): # For fun, fill this with lowercase alphabet
    payload[i] = 97 + (i % 26)

print(f"Completed string generation--Took {(time.time() - start):.2f} seconds.")

# Send this payload to the server using UDP
# Source: https://wiki.python.org/moin/UdpCommunication
sock_udp = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock_udp.bind((CLIENT_IP, CLIENT_PORT))

i = 0
segment_size=1500
print("Client is sending data now...")
print("CLIENT METADATA: IP Address: ", CLIENT_IP, "; Port: ", CLIENT_PORT)
# Split up the data into chunks
while i < len(payload): # Send in bunches of 1.5KB (maximum for my system)
    segment = payload[i:min(i + segment_size, len(payload))]
    if i == 0: print(f"Time of first packet sent: {time.time()}")
    sock_udp.sendto(segment, (SERVER_IP, SERVER_PORT))
    i += segment_size



# UDP is connectionless so we don't use socket.listen()
print("Server IP address: ", SERVER_IP)
print("Client is listening for a response...")
sock_udp.settimeout(1)
while True:
    try:
        # Keep sending the STOP message until the server returns something
        sock_udp.sendto("STOP".encode(), (SERVER_IP, SERVER_PORT))
        data, server_addr = sock_udp.recvfrom(4096) # Received message is not this
long
        print(data.decode())
        break
    except:
        continue

print("Client process completed iPerfectly!")
```

udp_servervictorialam_919626106_jinsiguo_918709406.py:

```
import socket
import time

SERVER_IP = "127.0.0.1"  # localhost
SERVER_PORT = 5555  # arbitrarily chosen socket

sock_udp = socket.socket(socket.AF_INET, socket.SOCK_DGRAM);
sock_udp.bind((SERVER_IP, SERVER_PORT))

times = [0, 0]
sizes = []
payload = []
src_addr = 0
first_time = 0
print("Server is receiving data now...")
print("SERVER METADTA: IP address: ", SERVER_IP, "; Port: ", SERVER_PORT)
while True:
    data, src_addr = sock_udp.recvfrom(4096) # BLOCKING function call
    if data == b"STOP":
        break
    now = time.time()
    if times[0] != 0: # If this not is the first record (this is scuffed, I know
LOL)
        sizes.append(len(data)) # Record number of bytes sent after first
"correct" time recorded
        times[1] = now
    else:
        times[0] = now
        first_time = time.time()

    payload.append(data.decode())

# Measure throughput and send back to client
# This measurement does not include the very first small packet sent
total_data = sum(sizes)
total_time = times[1] - times[0]
throughput = total_data / total_time
msg_str = f"Throughput: {(throughput / 1000):.3f} Kilobytes per second"

sock_udp.sendto(msg_str.encode(), src_addr)

print(f"Time that first packet was received: {first_time}")
print("Client IP address: ", src_addr[0])
print("Size of data received (bytes): ", total_data)
print("Time taken to receive (seconds): ", total_time)

print("Server process completed iPerfectly!")
```

Code generated by ChatGPT

- Link to ChatGPT session

Our problem that not all of the data was included in the throughput calculation was solved in ChatGPT's response. ChatGPT's response also used a different method of stopping the server from receiving data, which was to count the total number of bytes received and comparing it with the number sent in the first packet. However, this seems to be a risky way to send data since if a single packet is dropped, the server will block forever while waiting for more data (timeout is never set).

Edited code using LLM

Inspired by the way ChatGPT used a message to start communication before sending the actual payload, we added a message to start the server's size summation and time records.

udp_clientvictorialam_919626106_jinsiguo_918709406.py:

```
import socket
import sys
import time

# Source: https://realpython.com/python-sockets/
SERVER_IP = "127.0.0.1"  # localhost
SERVER_PORT = 5555  # arbitrarily chosen socket

CLIENT_IP = "127.0.0.1"
CLIENT_PORT = 7777

# Validate input
if len(sys.argv) != 2:
    print("Usage: udp_xyzabc.py [size in MB to send]")
    exit(1)
if sys.argv[1].isdigit() == False:
    print("Not a number!")
    exit(1)

size_mb = int(sys.argv[1])
if (size_mb < 25 or size_mb > 200):
    print("size must be between 25 and 200")
    exit(1)


# Generate payload of size "size_mb"
start = time.time()
payload = bytearray(size_mb * 1000000)
for i in range(size_mb * 1000000): # For fun, fill this with lowercase alphabet
    payload[i] = 97 + (i % 26)

print(f"Completed string generation--Took {(time.time() - start):.2f} seconds.")

# Send this payload to the server using UDP
# Source: https://wiki.python.org/moin/UdpCommunication
# ---- socket.AF_INET is a macro for internet in general
# ---- socket.SOCK_DGRAM is a macro representing UDP
sock_udp = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock_udp.bind((CLIENT_IP, CLIENT_PORT))
```

```
i = 0
segment_size=1500
print("Client is sending data now...")
print("CLIENT METADATA: IP Address: ", CLIENT_IP, "; Port: ", CLIENT_PORT)

sock_udp.sendto("START".encode(), (SERVER_IP, SERVER_PORT)) # Send start message
print(f"Time of first packet sent: {time.time()}")

while i < len(payload): # Send in bunches of 1.5KB (maximum for my system)
    segment = payload[i:min(i + segment_size, len(payload))]
    sock_udp.sendto(segment, (SERVER_IP, SERVER_PORT))
    i += segment_size

print("Amount of data sent (bytes): ", len(payload))
# UDP is connectionless so we don't use socket.listen()

print("Client is listening for a response...")
sock_udp.settimeout(1)
while True:
    try:
        sock_udp.sendto("STOP".encode(), (SERVER_IP, SERVER_PORT))
        data, server_addr = sock_udp.recvfrom(4096) # buffer size is 4096 bytes
(blocking)
        print(data.decode())
        break
    except:
        continue

print("Client process completed iPerfectly!")
```

udp_servervictorialam_919626106_jinsiguo_918709406.py:

```
import socket
import time

SERVER_IP = "127.0.0.1"  # localhost
SERVER_PORT = 5555  # arbitrarily chosen socket

sock_udp = socket.socket(socket.AF_INET, socket.SOCK_DGRAM);
sock_udp.bind((SERVER_IP, SERVER_PORT))

start = 0; end = 0
sizes = []
payload = []
src_addr = 0
first_time = 0

print("Server is receiving data now...")
print("SERVER METADTA: IP address: ", SERVER_IP, "; Port: ", SERVER_PORT)
# Receive the START message
data, src_addr = sock_udp.recvfrom(4096)
```

```
assert data == b"START"
start = time.time()

# Receive the payload
while True:
    data, src_addr = sock_udp.recvfrom(4096) # BLOCKING function call

    if data == b"STOP":
        break

    sizes.append(len(data)) # Record number of bytes sent after first "correct"
time recorded
    payload.append(data.decode())
    end = time.time()

# Measure throughput and send back to client
total_data = sum(sizes)
total_time = end - start
throughput = total_data / total_time
msg_str = f"Throughput: {(throughput / 1000):.3f} Kilobytes per second"

sock_udp.sendto(msg_str.encode(), src_addr)

print(f"Time that first packet was received: {start}")
print("Client IP address: ", src_addr[0])
print("Size of data received (bytes): ", total_data)
print("Time taken to receive (seconds): ", total_time)

print("Server process completed iPerfectly!")
```

Actual output from this code (Sending 25MB):

Server:

```
Server is receiving data now...
SERVER METADTA: IP address:  127.0.0.1 ; Port:  5555
Time that first packet was received: 1740105380.246206
Client IP address:  127.0.0.1
Size of data received (bytes):  24500500
Time taken to receive (seconds):  0.09058976173400879
Server process completed iPerfectly!
```

Client:

```
Completed string generation--Took 4.43 seconds.
Client is sending data now...
CLIENT METADATA: IP Address:  127.0.0.1 ; Port:  7777
Time of first packet sent: 1740105380.2459898
Amount of data sent (bytes):  25000000
```

```
Client is listening for a response...
Throughput: 270455.508 Kilobytes per second
Client process completed iPerfectly!
```

As shown in these outputs, the amount of data received was less than the amount of data sent. In the shown outputs, it looks like 499500 bytes were lost, which was 333 1.5KB-sized packets. The metric that can describe this is packet loss, and in this case, the packet loss is 333. This happens because UDP is a best-effort service with no reliability guarantees. Packets may or may not make it over the link without errors or without getting lost, and UDP does not attempt to recover lost packets.