CECS 327

Project 2: A Bite of Distributed Communication

Ryan Tomas: 028210102

10/09/2025

Introduction

In this project, I worked alone, and the objective of this project was to explore the differences between TCP and UDP. The way I saw the differences was setting up one anycast for TCP and multicast for UDP. With the code, we can spot key differences between the two. This is possible with the use of Docker and monitoring the traffic between the server and clients.

Design

In this project I need to make two different folders because each protocol needs it's own dockerfile and ymal file. One folder is called TCP which is the anycast. This folder holds a dockerfile, ymal file, server.py and client.py. In server.py we inialize a server on host "0.0.0.0" and port 5000. Then I kept it alive with a while loop because I would have to run the client.py later to see if the handshakes happens. For client.py it picks a random server that was named server 1-5. After it picks a server it trys to connect with the the same host and port numbers. If done correctly it would receive the message form the server and print out the message. Now for the UDP multicast is the other folder and the files include dockerfile and ymal file with multicast_receiver.py and multicast_sender.py. For the multicast_sender.py I give the host number to "224.1.1.1" and port of 5007. It would send a normal text and json massage and binary data to the group of receiver that are on the same host and port. For muliticast_receiver.py it takes a argument that is just the duration that it's going to sleep for. Then it would listen for

any incoming message from the sender and prints them out. While making each protocol it showed me what differences each one does.

Differences

There are key differences between TCP and UDP, each with its own advantages and disadvantages. TCP is reliable because the server and client make a handshake that binds them. This action makes it reliable to send data. However, UDP isn't reliable because the sender does not know who is receiving the data, and if any packets are lost. The missing handshake is the difference between the reliable and unreliable, but this also had an influence on speed. While TCP is reliable but it's slow to complete the data transfer because it checks if any packets are lost. For UDP, it does not care if the packets are lost, which in turn makes it quicker.

Join/leave group

UDP is using multicast in this project, and how it's set up is with 2 senders and 2 receivers. Both receivers would join a group with specific ports that are being used by both senders. The sender does not control who gets packets. It only sends it to a specific group that any receiver can join. When joining or leaving a multicast group, the receiver has control. If a receiver does not join the group, it will not get the packets that are being sent from the sender. Only by joining the group would the receiver know and get the packets being sent. However, if the receiver decides to leave, it will not receive packets again. This all happens without the sender even knowing.

Single vs multi source

In single-source multicast, there's only one sender that transmits the message to a multicast group. Which in turn, the single source sends to receivers who joined, and there is no ambiguity about where the data is coming from. In multi-source multicast, there is more than one

sender that sends packets to a multicast group. Then the receivers get packets from any source sending to their group. With many senders, the receiver could get packets from different senders and at different speeds. This can lead to duplicate packets and arriving in different orders. For these reasons, the packet would need to go through a filtering protocol.

Challenges

While working on this project, it was difficult to run the TCP dump when doing the UDP multicast. There was a problem that it would not print anything in the terminal. I fixed it because I only did the command of the TCP dump in the Dockerfile and not the YAML file. In the multicast_Sender.py, I also had a problem where it would initialize first and end before the multicast_reciever would get the packets. The way around this was to sleep the multicast_sender.py for 5 seconds. After this, it would work as normal.

Conclusion

After finishing this project, I was able to observe the differences between TCP and UDP. Both had key differences for TCP anycast and UDP multicast. It became clear how reliability and speed differ between the two protocols. TCP shows its reliable data transfer through connection establishment and acknowledgments between the servers and clients. While UDP prioritizes speed and simplicity by sending packets without assurances of packet delivery. This project provided valuable insight into the trade-offs between the two protocols. As well as how multicast mechanisms manage group communication with each other.

Videos

https://drive.google.com/file/d/1TtifUEH5ZakMK-erbAy3iDwr1EQ63s1e/view?usp=sharing

Output

UDP Multicast:

```
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
6 packets captured
6 packets received by filter
0 packets dropped by kernel
03:33:55.898146 IP 192.168.65.3.43306 > 224.1.1.1.5007: UDP, length 48
03:33:55.898166 IP 192.168.65.3.53839 > 224.1.1.1.5007: UDP, length 48
03:33:55.898167 IP 192.168.65.3.53839 > 224.1.1.1.5007: UDP, length 34
03:33:55.898167 IP 192.168.65.3.53839 > 224.1.1.1.5007: UDP, length 16
03:33:55.898182 IP 192.168.65.3.43306 > 224.1.1.1.5007: UDP, length 38
03:33:55.898186 IP 192.168.65.3.43306 > 224.1.1.1.5007: UDP, length 16
[receiver] Joined multicast group
[receiver] Listening on multicast group 224.1.1.1:5007
[receiver] Received message from ('192.168.65.3', 43306): Message: Hello, this is a multicast message from sender2.
[receiver] Received message from ('192.168.65.3', 53839): Message: Hello, this is a multicast message from sender1.
[receiver] Received {'sensor': 'temp', 'value': 22.29}
[receiver] Received binary from ('192.168.65.3', 53839):
b'1\xae\xdf)\xc1\x9a\xa7\x11_\xb4\x06\x9f\x88\xd2k]'
[receiver] Received {'sensor': 'humidity', 'value': 22.35}
[receiver] Received binary from ('192.168.65.3', 43306):
b'\xd3D\xe0\x8f\xb0\r\xb9\xef\xdf\xee\xe9\xed9x~\x97'
[receiver] Leaving multicast group and closing socket.
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
03:33:55.898147 IP 192.168.65.3.43306 > 224.1.1.1.5007: UDP, length 48
03:33:55.898166 IP 192.168.65.3.53839 > 224.1.1.1.5007: UDP, length 48
03:33:55.898167 IP 192.168.65.3.53839 > 224.1.1.1.5007: UDP, length 34
03:33:55.898167 IP 192.168.65.3.53839 > 224.1.1.1.5007: UDP, length 16
03:33:55.898182 IP 192.168.65.3.43306 > 224.1.1.1.5007: UDP, length 38
03:33:55.898186 IP 192.168.65.3.43306 > 224.1.1.1.5007: UDP, length 16
6 packets captured
```

```
6 packets received by filter
0 packets dropped by kernel
[receiver] Joined multicast group
[receiver] Listening on multicast group 224.1.1.1:5007
[receiver] Received message from ('192.168.65.3', 53839): Message: Hello, this is a multicast message from sender1.
[receiver] Received message from ('192.168.65.3', 43306): Message: Hello, this is a multicast message from sender2.
[receiver] Received {'sensor': 'temp', 'value': 22.29}
[receiver] Received binary from ('192.168.65.3', 53839):
b'1\xae\xdf)\xc1\x9a\xa7\x11_\xb4\x06\x9f\x88\xd2k]'
[receiver] Received {'sensor': 'humidity', 'value': 22.35}
[receiver] Received binary from ('192.168.65.3', 43306):
b'\xd3D\xe0\x8f\xb0\r\xb9\xef\xdf\xee\xe9\xed9x~\x97'
[receiver] Leaving multicast group and closing socket.
```

TCP anycast:

tcpdump: verbose output suppressed, use -v[v]... for full protocol decode

listening on eth0, link-type EN10MB (Ethernet), snapshot length 262144 bytes

03:05:05.115157 IP 172.20.0.7.35768 > 172.20.0.6.5000: Flags [S], seq 2707721084, win 64240, options [mss 1460,sackOK,TS val 2086597136 ecr 0,nop,wscale 7], length 0

03:05:05.115171 IP 172.20.0.6.5000 > 172.20.0.7.35768: Flags [S.], seq 567000468, ack 2707721085, win 65160, options [mss 1460,sackOK,TS val 3641983113 ecr 2086597136,nop,wscale 7], length 0

03:05:05.115186 IP 172.20.0.7.35768 > 172.20.0.6.5000: Flags [.], ack 1, win 502, options [nop,nop,TS val 2086597136 ecr 3641983113], length 0

03:05:05.115329 IP 172.20.0.6.5000 > 172.20.0.7.35768: Flags [P.], seq 1:20, ack 1, win 510, options [nop,nop,TS val 3641983113 ecr 2086597136], length 19

03:05:05.115381 IP 172.20.0.7.35768 > 172.20.0.6.5000: Flags [.], ack 20, win 502, options [nop,nop,TS val 2086597136 ecr 3641983113], length 0

03:05:05.115402 IP 172.20.0.6.5000 > 172.20.0.7.35768: Flags [F.], seq 20, ack 1, win 510, options [nop,nop,TS val 3641983113 ecr 2086597136], length 0

03:05:05.115412 IP 172.20.0.7.35768 > 172.20.0.6.5000: Flags [F.], seq 1, ack 20, win 502, options [nop,nop,TS val 2086597136 ecr 3641983113], length 0

03:05:05.115418 IP 172.20.0.6.5000 > 172.20.0.7.35768: Flags [.], ack 2, win 510, options [nop,nop,TS val 3641983113 ecr 2086597136], length 0

03:05:05.115428 IP 172.20.0.7.35768 > 172.20.0.6.5000: Flags [.], ack 21, win 502, options [nop,nop,TS val 2086597136 ecr 3641983113], length 0

03:05:19.686641 IP 172.20.0.7.53920 > 172.20.0.6.5000: Flags [S], seq 3528163136, win 64240, options [mss 1460,sackOK,TS val 2086611708 ecr 0,nop,wscale 7], length 0

03:05:19.686659 IP 172.20.0.6.5000 > 172.20.0.7.53920: Flags [S.], seq 4013093071, ack 3528163137, win 65160, options [mss 1460,sackOK,TS val 3641997685 ecr 2086611708,nop,wscale 7], length 0

03:05:19.686677 IP 172.20.0.7.53920 > 172.20.0.6.5000: Flags [.], ack 1, win 502, options [nop,nop,TS val 2086611708 ecr 3641997685], length 0

03:05:19.686805 IP 172.20.0.6.5000 > 172.20.0.7.53920: Flags [P.], seq 1:20, ack 1, win 510, options [nop,nop,TS val 3641997685 ecr 2086611708], length 19

03:05:19.686861 IP 172.20.0.7.53920 > 172.20.0.6.5000: Flags [.], ack 20, win 502, options [nop,nop,TS val 2086611708 ecr 3641997685], length 0

03:05:19.686889 IP 172.20.0.6.5000 > 172.20.0.7.53920: Flags [F.], seq 20, ack 1, win 510, options [nop,nop,TS val 3641997685 ecr 2086611708], length 0

03:05:19.686891 IP 172.20.0.7.53920 > 172.20.0.6.5000: Flags [F.], seq 1, ack 20, win 502, options [nop,nop,TS val 2086611708 ecr 3641997685], length 0

03:05:19.686904 IP 172.20.0.7.53920 > 172.20.0.6.5000: Flags [.], ack 21, win 502, options [nop,nop,TS val 2086611708 ecr 3641997685], length 0

03:05:19.686909 IP 172.20.0.6.5000 > 172.20.0.7.53920: Flags [.], ack 2, win 510, options [nop,nop,TS val 3641997685 ecr 2086611708], length 0

03:05:26.488352 IP 172.20.0.7.46060 > 172.20.0.6.5000: Flags [S], seq 3818385567, win 64240, options [mss 1460,sackOK,TS val 2086618509 ecr 0,nop,wscale 7], length 0

03:05:26.488368 IP 172.20.0.6.5000 > 172.20.0.7.46060: Flags [S.], seq 3136718206, ack 3818385568, win 65160, options [mss 1460,sackOK,TS val 3642004486 ecr 2086618509,nop,wscale 7], length 0

03:05:26.488383 IP 172.20.0.7.46060 > 172.20.0.6.5000: Flags [.], ack 1, win 502, options [nop,nop,TS val 2086618509 ecr 3642004486], length 0

03:05:26.488556 IP 172.20.0.6.5000 > 172.20.0.7.46060: Flags [P.], seq 1:20, ack 1, win 510, options [nop,nop,TS val 3642004486 ecr 2086618509], length 19

03:05:26.488608 IP 172.20.0.7.46060 > 172.20.0.6.5000: Flags [.], ack 20, win 502, options [nop,nop,TS val 2086618509 ecr 3642004486], length 0

03:05:26.488629 IP 172.20.0.6.5000 > 172.20.0.7.46060: Flags [F.], seq 20, ack 1, win 510, options [nop,nop,TS val 3642004487 ecr 2086618509], length 0

03:05:26.488644 IP 172.20.0.7.46060 > 172.20.0.6.5000: Flags [F.], seq 1, ack 20, win 502, options [nop,nop,TS val 2086618510 ecr 3642004486], length 0

03:05:26.488651 IP 172.20.0.6.5000 > 172.20.0.7.46060: Flags [.], ack 2, win 510, options [nop,nop,TS val 3642004487 ecr 2086618510], length 0

03:05:26.488662 IP 172.20.0.7.46060 > 172.20.0.6.5000: Flags [.], ack 21, win 502, options [nop,nop,TS val 2086618510 ecr 3642004487], length 0

27 packets captured

27 packets received by filter

0 packets dropped by kernel