Brandon Wong 030359639

William Trinh 030650397

Report 1

Demo Link

1. **Docker Containers & Cluster Masters** (Brandon + Will)

We created the following .yaml file for the docker compose tool to automate the setup of the 16 clusters where we designated the first container of each cluster to be the master by setting its role.

docker-compose.yaml

```
services:
 cluster_1_1:
   image: python:3.9-slim
   container_name: cluster_1_1
   hostname: Cluster A Master
   networks:
     network:
       ipv4_address: 192.18.0.2
   volumes:
     - ./master_a.py:/app/master_a.py
   command: python /app/master_a.py
   environment:
     - ROLE=master

 cluster_1_2:
   image: python:3.9-slim
   container_name: cluster_1_2
   hostname: Container 1
   networks:
     network:
       ipv4_address: 192.18.0.3
   volumes:
     - ./broadcast_listener.py:/app/broadcast_listener.py
   command: python /app/broadcast_listener.py
```

```
cluster_1_3:
  image: python:3.9-slim
  container_name: cluster_1_3
  hostname: Container 2
  networks:
    network:
      ipv4_address: 192.18.0.4
  volumes:
    - ./broadcast_listener.py:/app/broadcast_listener.py
  command: python /app/broadcast_listener.py

cluster_1_4:
  image: python:3.9-slim
  container_name: cluster_1_4
  hostname: Container 3
  networks:
    network:
      ipv4_address: 192.18.0.5
  volumes:
    - ./broadcast_listener.py:/app/broadcast_listener.py
  command: python /app/broadcast_listener.py

cluster_1_5:
  image: python:3.9-slim
  container_name: cluster_1_5
  hostname: Container 4
  networks:
    network:
      ipv4_address: 192.18.0.6
  volumes:
    - ./broadcast_listener.py:/app/broadcast_listener.py
  command: python /app/broadcast_listener.py

cluster_1_6:
  image: python:3.9-slim
  container_name: cluster_1_6
  hostname: Container 5
  networks:
    network:
      ipv4_address: 192.18.0.7
```

```yaml
  volumes:
    - ./broadcast_listener.py:/app/broadcast_listener.py
  command: python /app/broadcast_listener.py

cluster_1_7:
  image: python:3.9-slim
  container_name: cluster_1_7
  hostname: Container 6
  networks:
    network:
      ipv4_address: 192.18.0.8
  volumes:
    - ./broadcast_listener.py:/app/broadcast_listener.py
  command: python /app/broadcast_listener.py

cluster_1_8:
  image: python:3.9-slim
  container_name: cluster_1_8
  hostname: Container 7
  networks:
    network:
      ipv4_address: 192.18.0.9
  volumes:
    - ./broadcast_listener.py:/app/broadcast_listener.py
  command: python /app/broadcast_listener.py

cluster_2_1:
  image: python:3.9-slim
  container_name: cluster_2_1
  hostname: Cluster B Master
  networks:
    network:
      ipv4_address: 192.18.0.10
  environment:
    - ROLE=master
  volumes:
    - ./master_b.py:/app/master_b.py
  command: python /app/master_b.py

cluster_2_2:
  image: python:3.9-slim
  container_name: cluster_2_2
  hostname: Container 8
  networks:
    network:
      ipv4_address: 192.18.0.11
  volumes:
    - ./multicast_listener.py:/app/multicast_listener.py
  command: python /app/multicast_listener.py

cluster_2_3:
  image: python:3.9-slim
  container_name: cluster_2_3
  hostname: Container 9
  networks:
    network:
      ipv4_address: 192.18.0.12
  volumes:
    - ./multicast_listener.py:/app/multicast_listener.py
  command: python /app/multicast_listener.py

cluster_2_4:
  image: python:3.9-slim
  container_name: cluster_2_4
  hostname: Container 10
  networks:
    network:
      ipv4_address: 192.18.0.13
  environment:
    - MULTICAST_GROUP=224.1.1.1
  volumes:
    - ./multicast_listener.py:/app/multicast_listener.py
  command: python /app/multicast_listener.py

cluster_2_5:
  image: python:3.9-slim
  container_name: cluster_2_5
  hostname: Container 11
  networks:
    network:
      ipv4_address: 192.18.0.14
  environment:
    - MULTICAST_GROUP=224.1.1.1
  volumes:
    - ./multicast_listener.py:/app/multicast_listener.py
  command: python /app/multicast_listener.py

cluster_2_6:
  image: python:3.9-slim
  container_name: cluster_2_6
  hostname: Container 12
  networks:
    network:
      ipv4_address: 192.18.0.15
  environment:
    - MULTICAST_GROUP=224.1.1.1
  volumes:
    - ./multicast_listener.py:/app/multicast_listener.py
  command: python /app/multicast_listener.py

cluster_2_7:
  image: python:3.9-slim
  container_name: cluster_2_7
  hostname: Container 13
  networks:
    network:
      ipv4_address: 192.18.0.16
  volumes:
    - ./multicast_listener.py:/app/multicast_listener.py
  command: python /app/multicast_listener.py

cluster_2_8:
  image: python:3.9-slim
  container_name: cluster_2_8
  hostname: Container 14
  networks:
    network:
      ipv4_address: 192.18.0.17
  volumes:
    - ./multicast_listener.py:/app/multicast_listener.py
```

```
   command: python /app/multicast_listener.py

networks:
 network:
   driver: bridge
   ipam:
     config:
       - subnet: 192.18.0.0/24
         gateway: 192.18.0.1
```

With this yaml configuration file, we can run the command:

docker compose up

Which will start and run our containers and the intra a and inter communication protocols linked to them. Cluster 1 containers will receive a broadcasted message from Cluster 1's master. Cluster 2 containers will receive a multicast message from Cluster 2's master. After performing intra communication, Cluster 1's master will unicast a message to Cluster 2's master who will then broadcast it to its children.

2. **Intra + Inter-Cluster Communication** (Brandon)

master_a.py

```
import socket
from time import sleep

hostname = socket.gethostname()

print(f"[{hostname}] Starting Cluster {hostname.split()[-2]} with 8
containers...")

# INTRA

message = "Hello, Cluster A!"
print(f'[{hostname}] Sending intra-cluster broadcast message:
"{message}"')
sleep(3.5) # wait for other containers to await the message
```

```
# AF_INET = using IPv4, SOCK_DGRAM = socket will use UDP
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# sets socket options
sock.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)

# sends message through the socket, 255.255.255.255 is the destination address
# and that is the broadcast address for IPv4, allows sending messages to all devices
# on local network. 5000 is port, recieving must be listening on port 5000
sock.sendto(message.encode(), ("255.255.255.255", 5000))
sock.close()

# INTER

# wait to send message, wait for all intra stuff to finish for both clusters
sleep(2)
message = "Greetings from Cluster A. Hello Cluster B."

poop_sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
# send message to other cluster master who will then send it to little children
print(f'[{hostname}] Sending inter-cluster broadcast message: "{message}"')
poop_sock.sendto(message.encode(), ("192.18.0.10", 6000))
poop_sock.close()
```

This is executed by Cluster A Master which sends the message “Hello, Cluster A!” to all machines listening on port 5000. After that, it unicasts a message to Cluster B’s master on port 6000.

broadcast_listener.py

```
import socket

hostname = socket.gethostname()

# AF_INET = IPv4, SOCK_DRAM = UDP
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
# set socket options
sock.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)
# binds socket to port 5000 and listen on it
sock.bind(("", 5000))

# receive data from socket, 1024 is max bytes to accept
data, _ = sock.recvfrom(1024)
print(f'[{hostname}] Received broadcast message: "{data.decode()}"')

sock.close()
```

This is executed by all containers in Cluster A except the master. It listens on port 5000 for any traffic and once received, it will decode and print the message.

master_b.py

```
import socket
from time import sleep

hostname = socket.gethostname()

print(f"[{hostname}] Starting Cluster {hostname.split()[-2]} with 8 containers...")

# INTRA

# multicast address to send message to
multicast_group = "224.1.1.1"
```

```
message = "Hello, Group B!"
print(f'[{hostname}] Sending intra-cluster multicast message:
"{message}"')
sleep(3.5) # wait for other containers to await the message

# af_inet = ipv4, sock_dgram = UDP, ipproto_udp = furhter specify udp
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM,
socket.IPPROTO_UDP)

# socket options
sock.setsockopt(socket.IPPROTO_IP, socket.IP_MULTICAST_TTL, 2)
sock.settimeout(10)

# send message only to those with the multicast address at port 5007
sock.sendto(message.encode(), (multicast_group, 5007))
sock.close()

# INTER

# recieve from cluster A
poop_sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
poop_sock.bind(("", 6000))

data, _ = poop_sock.recvfrom(1024)
message = data.decode()
print(f"[{hostname}] Receieved communication from Cluster A. Sending
broacast message: {message}")
poop_sock.close()

# broadcast data from cluster A into cluster B children
nsock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
nsock.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)
nsock.sendto(message.encode(), ("255.255.255.255", 6001))
nsock.close()
```

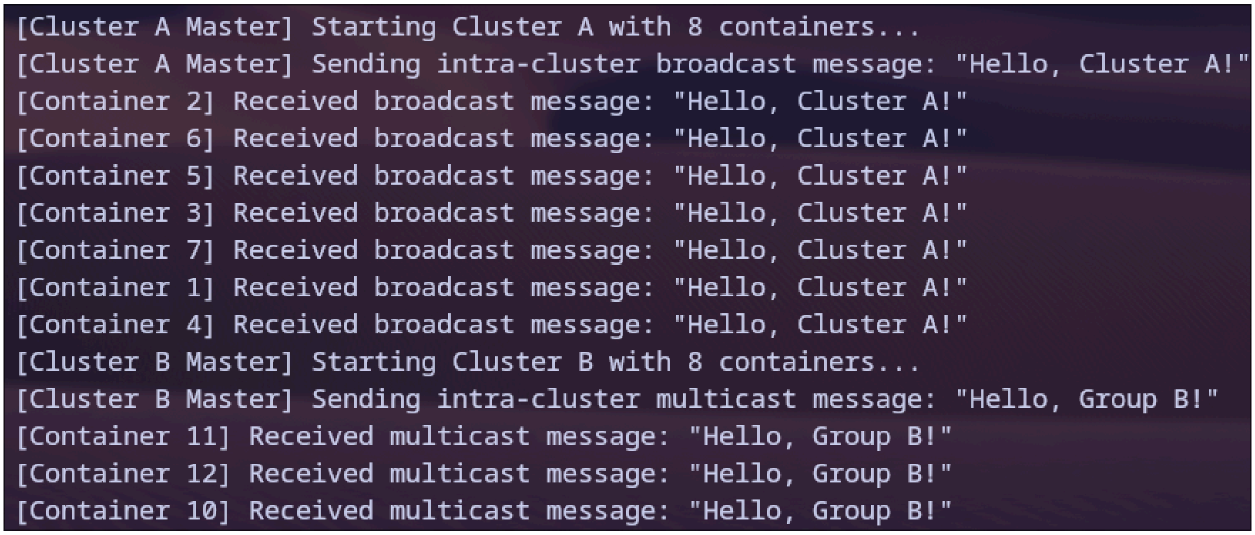
This is executed by Cluster B Master which sends the message “Hello, Group B!” to all machines listening on port 5007 that are part of the multicast group. After sending the message, it listens on port 6000 for the unicast message sent from Cluster A’s master and then broadcasts it to its own children.

multicast_listener.py

```
import socket
import struct
import os

multicast_group = os.getenv("MULTICAST_GROUP", "224.1.1.2")

#INTRA

# use ipv4 and udp
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM,
socket.IPPROTO_UDP)
# these options let the port be used by other applications
sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
# listen on port 5007
sock.bind(("", 5007))

# lets socket join a multicast group
mreq = struct.pack("4sl", socket.inet_aton(multicast_group),
socket.INADDR_ANY)
sock.setsockopt(socket.IPPROTO_IP, socket.IP_ADD_MEMBERSHIP, mreq)
sock.settimeout(5)

# listen and wait for data on socket
try:
   data, _ = sock.recvfrom(1024)
   print(f'[{socket.gethostname()}] Received multicast message:
"{data.decode()}"')
except:
   pass
sock.close()

# INTER
# RECEIEVED MSG
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)
sock.bind(("", 6001))
data, _ = sock.recvfrom(1024)
print(f'[{socket.gethostname()}] Received inter-broadcast message:
"{data.decode()}"')
sock.close()
```

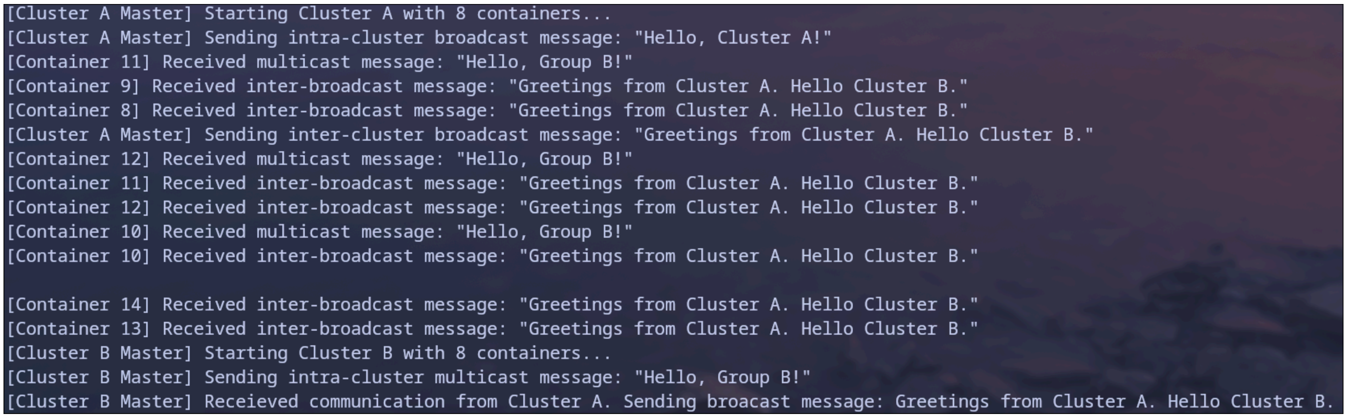
All containers will listen on port 5007, those within the multicast group will receive the message. The others will timeout within 5 seconds. The B Containers will then listen on port 6001 for the broadcast with the message from Cluster A.

Output for (intra only): *docker compose up*



```
[Cluster A Master] Starting Cluster A with 8 containers...
[Cluster A Master] Sending intra-cluster broadcast message: "Hello, Cluster A!"
[Container 2] Received broadcast message: "Hello, Cluster A!"
[Container 6] Received broadcast message: "Hello, Cluster A!"
[Container 5] Received broadcast message: "Hello, Cluster A!"
[Container 3] Received broadcast message: "Hello, Cluster A!"
[Container 7] Received broadcast message: "Hello, Cluster A!"
[Container 1] Received broadcast message: "Hello, Cluster A!"
[Container 4] Received broadcast message: "Hello, Cluster A!"
[Cluster B Master] Starting Cluster B with 8 containers...
[Cluster B Master] Sending intra-cluster multicast message: "Hello, Group B!"
[Container 11] Received multicast message: "Hello, Group B!"
[Container 12] Received multicast message: "Hello, Group B!"
[Container 10] Received multicast message: "Hello, Group B!"
```

Output with inter (Out of order):



```
[Cluster A Master] Starting Cluster A with 8 containers...
[Cluster A Master] Sending intra-cluster broadcast message: "Hello, Cluster A!"
[Container 11] Received multicast message: "Hello, Group B!"
[Container 9] Received inter-broadcast message: "Greetings from Cluster A. Hello Cluster B."
[Container 8] Received inter-broadcast message: "Greetings from Cluster A. Hello Cluster B."
[Cluster A Master] Sending inter-cluster broadcast message: "Greetings from Cluster A. Hello Cluster B."
[Container 12] Received multicast message: "Hello, Group B!"
[Container 11] Received inter-broadcast message: "Greetings from Cluster A. Hello Cluster B."
[Container 12] Received inter-broadcast message: "Greetings from Cluster A. Hello Cluster B."
[Container 10] Received multicast message: "Hello, Group B!"
[Container 10] Received inter-broadcast message: "Greetings from Cluster A. Hello Cluster B."

[Container 14] Received inter-broadcast message: "Greetings from Cluster A. Hello Cluster B."
[Container 13] Received inter-broadcast message: "Greetings from Cluster A. Hello Cluster B."
[Cluster B Master] Starting Cluster B with 8 containers...
[Cluster B Master] Sending intra-cluster multicast message: "Hello, Group B!"
[Cluster B Master] Receieved communication from Cluster A. Sending broacast message: Greetings from Cluster A. Hello Cluster B.
```

3. **Network Monitoring** (Will)

Type	**Time**	**Source Cluster**	**Destination Cluster**	**Source IP**	**Destination IP**	**Protocol**	**Length (bytes)**	**Flag (hex)**
Intra-Broadcast	0.102268	Cluster A	Cluster A	192.18.0.2	192.18.0.3	UDP	45	0xDF
Intra-Multicast	0.139374	Cluster B	Cluster B	192.18.0.10	192.18.0.11	UDP	102	0x00
Inter-Broadcast	1.622	Cluster A	Cluster B	192.18.0.2	192.18.0.10	UDP	42	0x00

Full dump + Analysis:

```
listening on br-55671e8c68df, link-type EN10MB (Ethernet), snapshot length
262144 bytes
18:43:24.009961 ARP, Request who-has 192.18.0.2 tell 192.18.0.2, length 28
18:43:24.011185 IP6 arch-envy > ff02::16: HBH ICMP6, multicast listener
report v2, 2 group record(s), length 48
18:43:24.018607 ARP, Request who-has 192.18.0.9 tell 192.18.0.9, length 28
18:43:24.023365 ARP, Request who-has 192.18.0.14 tell 192.18.0.14, length
28
18:43:24.102129 ARP, Request who-has 192.18.0.12 tell 192.18.0.12, length
28
```

```
18:43:24.131696 ARP, Request who-has 192.18.0.11 tell 192.18.0.11, length
28
18:43:24.161023 ARP, Request who-has 192.18.0.4 tell 192.18.0.4, length 28
18:43:24.164771 ARP, Request who-has 192.18.0.13 tell 192.18.0.13, length
28
18:43:24.165217 IP 192.18.0.14 > igmp.mcast.net: igmp v3 report, 1 group
record(s)
18:43:24.174191 IP6 arch-envy > ff02::16: HBH ICMP6, multicast listener
report v2, 2 group record(s), length 48
18:43:24.188622 ARP, Request who-has 192.18.0.5 tell 192.18.0.5, length 28
18:43:24.202267 IP 192.18.0.12 > igmp.mcast.net: igmp v3 report, 1 group
record(s)
18:43:24.219470 ARP, Request who-has 192.18.0.16 tell 192.18.0.16, length
28
18:43:24.224623 ARP, Request who-has 192.18.0.15 tell 192.18.0.15, length
28
18:43:24.225229 IP 192.18.0.11 > igmp.mcast.net: igmp v3 report, 1 group
record(s)
18:43:24.248890 ARP, Request who-has 192.18.0.8 tell 192.18.0.8, length 28
18:43:24.250246 IP 192.18.0.13 > igmp.mcast.net: igmp v3 report, 1 group
record(s)
18:43:24.266544 ARP, Request who-has 192.18.0.17 tell 192.18.0.17, length
28
18:43:24.290146 ARP, Request who-has 192.18.0.10 tell 192.18.0.10, length
28
18:43:24.300239 IP 192.18.0.16 > igmp.mcast.net: igmp v3 report, 1 group
record(s)
18:43:24.311240 IP 192.18.0.15 > igmp.mcast.net: igmp v3 report, 1 group
record(s)
18:43:24.317002 ARP, Request who-has 192.18.0.6 tell 192.18.0.6, length 28
18:43:24.330886 ARP, Request who-has 192.18.0.7 tell 192.18.0.7, length 28
18:43:24.339469 ARP, Request who-has 192.18.0.3 tell 192.18.0.3, length 28
18:43:24.341199 IP 192.18.0.17 > igmp.mcast.net: igmp v3 report, 1 group
record(s)
18:43:24.461253 IP 192.18.0.14 > igmp.mcast.net: igmp v3 report, 1 group
record(s)
18:43:24.470173 IP 192.18.0.11 > igmp.mcast.net: igmp v3 report, 1 group
record(s)
18:43:24.565268 IP 192.18.0.17 > igmp.mcast.net: igmp v3 report, 1 group
record(s)
```

```
18:43:24.629253 IP 192.18.0.12 > igmp.mcast.net: igmp v3 report, 1 group
record(s)
18:43:24.821252 IP 192.18.0.15 > igmp.mcast.net: igmp v3 report, 1 group
record(s)
18:43:24.973275 IP 192.18.0.13 > igmp.mcast.net: igmp v3 report, 1 group
record(s)
18:43:25.011210 ARP, Request who-has 192.18.0.2 tell 192.18.0.2, length 28
18:43:25.019419 ARP, Request who-has 192.18.0.9 tell 192.18.0.9, length 28
18:43:25.024567 ARP, Request who-has 192.18.0.14 tell 192.18.0.14, length
28
18:43:25.103120 ARP, Request who-has 192.18.0.12 tell 192.18.0.12, length
28
18:43:25.133575 ARP, Request who-has 192.18.0.11 tell 192.18.0.11, length
28
18:43:25.161990 ARP, Request who-has 192.18.0.4 tell 192.18.0.4, length 28
18:43:25.165070 ARP, Request who-has 192.18.0.13 tell 192.18.0.13, length
28
18:43:25.189301 ARP, Request who-has 192.18.0.5 tell 192.18.0.5, length 28
18:43:25.220518 ARP, Request who-has 192.18.0.16 tell 192.18.0.16, length
28
18:43:25.225688 ARP, Request who-has 192.18.0.15 tell 192.18.0.15, length
28
18:43:25.250060 ARP, Request who-has 192.18.0.8 tell 192.18.0.8, length 28
18:43:25.267278 ARP, Request who-has 192.18.0.17 tell 192.18.0.17, length
28
18:43:25.290727 ARP, Request who-has 192.18.0.10 tell 192.18.0.10, length
28
18:43:25.293242 IP 192.18.0.16 > igmp.mcast.net: igmp v3 report, 1 group
record(s)
18:43:25.318216 ARP, Request who-has 192.18.0.6 tell 192.18.0.6, length 28
18:43:25.331412 ARP, Request who-has 192.18.0.7 tell 192.18.0.7, length 28
18:43:25.340572 ARP, Request who-has 192.18.0.3 tell 192.18.0.3, length 28
18:43:26.011700 ARP, Request who-has 192.18.0.2 tell 192.18.0.2, length 28
18:43:26.019981 ARP, Request who-has 192.18.0.9 tell 192.18.0.9, length 28
18:43:26.024147 ARP, Request who-has 192.18.0.14 tell 192.18.0.14, length
28
18:43:26.102615 ARP, Request who-has 192.18.0.12 tell 192.18.0.12, length
28
18:43:26.133235 ARP, Request who-has 192.18.0.11 tell 192.18.0.11, length
28
```

```
18:43:26.161638 ARP, Request who-has 192.18.0.4 tell 192.18.0.4, length 28
18:43:26.165850 ARP, Request who-has 192.18.0.13 tell 192.18.0.13, length
28
18:43:26.189902 ARP, Request who-has 192.18.0.5 tell 192.18.0.5, length 28
18:43:26.220849 ARP, Request who-has 192.18.0.16 tell 192.18.0.16, length
28
18:43:26.225073 ARP, Request who-has 192.18.0.15 tell 192.18.0.15, length
28
18:43:26.249685 ARP, Request who-has 192.18.0.8 tell 192.18.0.8, length 28
18:43:26.267412 ARP, Request who-has 192.18.0.17 tell 192.18.0.17, length
28
18:43:26.291135 ARP, Request who-has 192.18.0.10 tell 192.18.0.10, length
28
18:43:26.317820 ARP, Request who-has 192.18.0.6 tell 192.18.0.6, length 28
18:43:26.332479 ARP, Request who-has 192.18.0.7 tell 192.18.0.7, length 28
18:43:26.339821 ARP, Request who-has 192.18.0.3 tell 192.18.0.3, length 28
18:43:27.622160 IP 192.18.0.2.33136 > 255.255.255.255.commplex-main: UDP,
length 17
18:43:27.863303 IP 192.18.0.10.48268 > 224.1.1.1.wsm-server-ssl: UDP,
length 15
18:43:27.867229 IP 192.18.0.13 > igmp.mcast.net: igmp v3 report, 1 group
record(s)
18:43:27.867229 IP 192.18.0.14 > igmp.mcast.net: igmp v3 report, 1 group
record(s)
18:43:27.867229 IP 192.18.0.15 > igmp.mcast.net: igmp v3 report, 1 group
record(s)
18:43:28.013413 IP 192.18.0.14 > igmp.mcast.net: igmp v3 report, 1 group
record(s)
18:43:28.029193 IP 192.18.0.15 > igmp.mcast.net: igmp v3 report, 1 group
record(s)
18:43:28.181228 IP 192.18.0.13 > igmp.mcast.net: igmp v3 report, 1 group
record(s)
18:43:29.207223 IP 192.18.0.12 > igmp.mcast.net: igmp v3 report, 1 group
record(s)
18:43:29.230237 IP 192.18.0.11 > igmp.mcast.net: igmp v3 report, 1 group
record(s)
18:43:29.304188 IP 192.18.0.16 > igmp.mcast.net: igmp v3 report, 1 group
record(s)
18:43:29.346246 IP 192.18.0.17 > igmp.mcast.net: igmp v3 report, 1 group
record(s)
```

```
18:43:29.627295 ARP, Request who-has 192.18.0.10 tell 192.18.0.2, length 28
18:43:29.627321 ARP, Reply 192.18.0.10 is-at 36:3f:9a:18:05:93 (oui Unknown), length 28
18:43:29.627323 IP 192.18.0.2.38899 > 192.18.0.10.x11: UDP, length 42
18:43:29.627454 IP 192.18.0.10.56613 > 255.255.255.255.6001: UDP, length 42
```

Analysis: The provided tcpdump shows regular ARP requests for IP-to-MAC resolution within a local network, alongside IGMP v3 reports indicating multicast group membership. There are UDP packets exchanged both within the same cluster (intra-cluster) and between different clusters (inter-cluster), primarily for network discovery or communication. The traffic reflects standard network operations involving ARP, multicast communication, and inter-cluster UDP messaging.

Broadcast and multicast protocols were chosen due to their level of difficulty for implementing them. Anycast would require further programming to find the route of the closest container. We believed it was best to start with easier implementations and work our way up to harder ones to best gain a foundational understanding of implementing the protocols.