



---

# CS2031 Telecommunication II

## Assignment #1: Command and Control

---

Yannick Gloster, Std# 18308167

November 3, 2019

### Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Theory</b>	<b>2</b>
2.1	Node . . . . .	2
2.1.1	CommandAndControl . . . . .	2
2.1.2	Broker . . . . .	2
2.1.3	Worker . . . . .	3
2.2	PacketContent . . . . .	3
2.2.1	AckPacketContent . . . . .	3
2.2.2	CommandPacket . . . . .	3
2.2.3	BrokerPacket . . . . .	3
2.2.4	WorkerPacket . . . . .	3
2.3	Communication . . . . .	3
<b>3</b>	<b>Implementation</b>	<b>3</b>
3.1	CommandAndControl . . . . .	4
3.2	Broker . . . . .	4
3.3	Worker . . . . .	5
3.4	Virtualization and Networking . . . . .	6
3.5	Packet Encoding . . . . .	7
<b>4</b>	<b>Summary</b>	<b>15</b>
<b>5</b>	<b>Reflection</b>	<b>15</b>

## 1 Introduction

Our problem description consisted of a Command and Control which sends messages to a broker which forwards and distributes those messages to a number of workers. Reference figure 1.

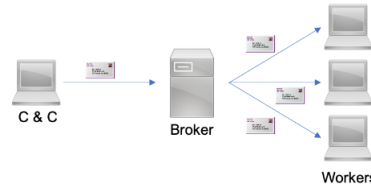


Figure 1: This diagram was provided to us as a means of description demonstrating how the different aspects of our project communicated.

We were provided with a basic sample program which implemented a client sending a packet including a string to a server to which the server sent an acknowledgment packet back to the client. I used this code to begin writing my implementation of our problem.

*Note:* To emulate multiple machines communicating with each other on a network, we used docker to set up multiple virtual instances communicating with each other in a network.

## 2 Theory

Before implementing my solution to the assignment, I conceptualized the communication between each instance.

I needed to build a network that each instance could connect to and communicate through. On my network, each instance would have a unique Internet Protocol Address (IP Address) that would allow each instance to communicate directly with one of the other instances by pointing to the other instance's unique address.

### 2.1 Node

In our example code, we were given *Node.java*. *Node.java* is an abstract class that creates a template node upon which our other node classes are built on. A node is described as “a communication endpoint that is connected to a network and is capable of creating, receiving, or transmitting information over a communications channel” ([1]).

At its fundamentals, the node given to us listens for incoming packets on a datagram socket and then notifies it's that it has recieved a packet so that it can then do something with the packet.

I then used node to create the following three classes:

#### 2.1.1 CommandAndControl

The Command And Control application is the start of our whole process. The Command and Control has a message that it sends to the broker. It will attempt to communicate with the Broker every 2000 milliseconds until it receives a response from the Broker acknowledging that it has recieved the task and is ready to complete it's task as layed out in **2.1.2 Broker**.

#### 2.1.2 Broker

The Broker continuously waits for two things; It waits for a task from the Command and Control and the number of repetitions of that task, then it also waits for a Worker to communicate it's availability with the Broker.

The Broker then sends out work to the workers as needed until the task given from the Command and Control is completed and once it is completed, it lets the Command And Control know that the task is complete.

### 2.1.3 Worker

The Worker communicates only with the Broker and lets the Broker know it's availability as set by the user. When available, it completes the singular task given from the Broker and then waits for another assessment.

## 2.2 PacketContent

As part of our example code, we were also given *PacketContent.java*. *PacketContent.java* is a abstract class that creates a template packet upon which our other packet classes are built on. A packet is described as “a group of bits that includes data plus control information. Generally refers to a network layer (OSI layer 3 [see **Table 1**]) protocol data unit.” ([3]).

Layer	Protocol data unit	Function ([2])
7 Application	Data	Application
6 Presentation	Data	Application
5 Session	Data	Application
4 Transport	Segment, Datagram 1	Insures that the data has no errors
<b>3 Network</b>	<b>Packet</b>	<b>Manages transmission including addressing</b>
2 Data Link	Frame	Reliable transmission of the frames between nodes
1 Physical	Symbol	Manages transmission over physical medium.

Table 1: OSI model

### 2.2.1 AckPacketContent

This packet class was given to us but all it does is pass a simple string that can be used as acknowledgment.

### 2.2.2 CommandPacket

The CommandPacket transports the data and the number of tasks from the command packet.

### 2.2.3 BrokerPacket

The BrokerPacket transports only the data from the Broker to the Worker.

### 2.2.4 WorkerPacket

The WorkerPacket transmits the availability of the Worker to the Broker.

## 2.3 Communication

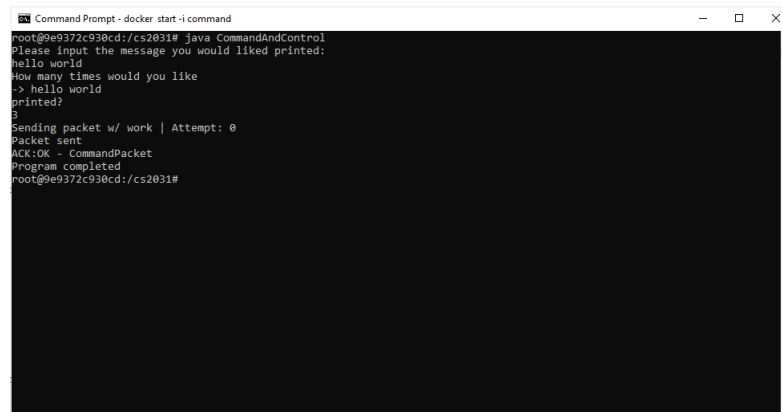
Each of my nodes implements selective repeat. Selective repeat is one way to make sure that there is no packet loss. In selective repeat, the program will continue to send packets without waiting for a receipt and once the packet that has been lost can be sent, it is automatically sent and the program continues. The program never hangs while waiting for a packet to be sent. By using selective repeat, we are only retransmitting the frames that are lost or damaged, This uses less bandwidth and is more efficient.

## 3 Implemenation

In this section, I will explain each component of my program and explain how I implemented the different aspects laid out in the theory section.

### 3.1 CommandAndControl

When the program is run, the application asks the user what string would the user like printed and then follows up by asking the number times the user would like that string printed. This information is then packaged into a `CommandPacket` and attempts to send it to the broker. It will continue to attempt to send the packet ever 2000 milliseconds until it receives an acknowledgment from the broker that it has received it. An example interaction can be seen in Figure 2 below.

A screenshot of a terminal window titled "Command Prompt - docker start -i command". The terminal shows the following interaction: the user runs "java CommandAndControl", the program prompts "Please input the message you would like printed:", the user enters "hello world", the program prompts "How many times would you like", the user enters "3", the program prompts "printed?", the user enters "3", the program outputs "Sending packet w/ work | Attempt: 0", "Packet sent", "Ack: OK - CommandPacket", "Program completed", and finally "root@9e9372c930cd:/cs2031#".

```
Command Prompt - docker start -i command
root@9e9372c930cd:/cs2031# java CommandAndControl
Please input the message you would like printed:
hello world
How many times would you like
-> hello world
printed?
3
Sending packet w/ work | Attempt: 0
Packet sent
Ack: OK - CommandPacket
Program completed
root@9e9372c930cd:/cs2031#
```

Figure 2: Here is an example of a user interaction of the Command and Control application.

The pseudo code for the `CommandAndControl` can be found in Listing 1 below:

```
println("Please input the message you would like printed:")
string inputString = input
println("How many times would you like " + inputString + " printed?")
int numTasks = input

CommandPacket = new CommandPacket(inputString, numTasks)
while(Broker Does Not Reply):
    send(commandPacket)
    wait(2000 ms)

wait(Broker Task Complete)

println("Task Complete")
```

Listing 1: `CommandAndControl` Pseudo Code

### 3.2 Broker

The Broker will wait until it receives data from the Command and Control or until a worker communicates with it. The Broker has a list of worker socket addresses, a count, and a `CommandPacket`.

A Worker sends `WorkerPacket` to the Broker which contains three boolean variables, volunteer for work, received work, and finished work. If the worker is ready for work, its socket address is added to the list and whenever the broker has work for the worker, it will communicate with it. If the worker communicates that it has finished a task that it has been assigned, the count is decremented.

If the Broker receives a `CommandPacket`, it saves the `CommandPacket` and copies the count to a local

variable counter. Then whenever there is an available worker in the list of socket addresses, it will send the work through a BrokerPacket to that worker in the list and then when it receives the work, it will decrements the counter. This will continue until count is 0 and the task is then complete.

Below in Figure 3 is an example of a terminal output of the Broker.

```

Command Prompt - docker start -i broker
root@1796aa3fce61:/cs2031# java Broker
Waiting for contact
Received WorkerPacket: Volunteered for Work: true - Received Work: false - Finished Work: false
Worker checked in for the first time.
Received CommandPacket: numTasks: 3 - Data: hello world
Sent AckPacket: java.net.DatagramPacket@c40caed
Received WorkerPacket: Volunteered for Work: false - Received Work: true - Finished Work: false
Received WorkerPacket: Volunteered for Work: true - Received Work: false - Finished Work: true
Received WorkerPacket: Volunteered for Work: false - Received Work: true - Finished Work: false
Received WorkerPacket: Volunteered for Work: true - Received Work: false - Finished Work: true
Received WorkerPacket: Volunteered for Work: false - Received Work: true - Finished Work: false
Received WorkerPacket: Volunteered for Work: true - Received Work: false - Finished Work: true
Task Complete

```

Figure 3: Here is an example of a terminal output of the Broker application.

The pseudo code for the Broker can be found in Listing 2 below:

```

while( true)
    if(Receives CommandPacket)
        LocalCommandPacket = CommandPacket
        Count = CommandPacket.count
    if(Receives WorkerPacket)
        if(Available to Work)
            SocketList.add(worker.socket)
        else
            SocketList.remove(worker.socket)
        if(Finished Work)
            count--

    if(LocalCommandPacket != null)
        for(Worker : SocketList)
            sendPacketToWorker(LocalCommandPacket.Worker)

```

Listing 2: Broker Pseudo Code

### 3.3 Worker

A Worker has three boolean variables, volunteer for work, received work, and finished work. When a worker is started, it prompts the user for a name and then asks if the user wants the worker to volunteer for work. It sets the other two boolean variables to be false. It will attempt to communicate through a WorkerPacket with a Broker and if it doesn't receive a response, it will continue to reach out to the Broker to communicate with it. Once it receives a packet from the broker, it sets volunteer for work to false, received work to true, and finished work to false. That packet is then sent back to the Broker. Following this, the data from the BrokerPacket is printed. It then sends a packet back to the Broker saying that it finished the work and is ready for new work.

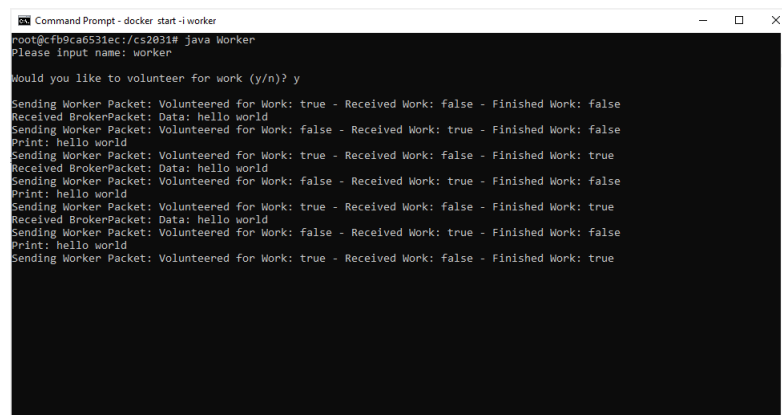


Figure 4: Here is an example of a user interaction of the Worker application.

The pseudo code for the Worker can be found in Listing 3 below:

```
println("Please input name:")
string name = input
println("Would you like to volunteer for work (y/n)?")
boolean volunteerForWork = input
boolean receivedWork = false
boolean finishedWork = false
send(WorkerPacket(volunteerForWork, receivedWork, finishedWork))

while(true)
    if(Receives BrokerPacket)
        receivedWork = true
        volunteerForWork = false
        finishedWork = false
        send(WorkerPacket(volunteerForWork, receivedWork, finishedWork))

        print(BrokerPacket.String)

        receivedWork = false
        volunteerForWork = true
        finishedWork = true
        send(WorkerPacket(volunteerForWork, receivedWork, finishedWork))
```

Listing 3: Worker Pseudo Code

### 3.4 Virtualization and Networking

I ran my different classes in different virtual linux instances using docker. I then connected them to a virtual network through docker. This allowed for each program to run independently of each other communicating as if they were multiple machines on one network when in reality they were multiple instances on my machine. I was also able to use wireshark on the broker to track the packets passing from one machine to the next.

Bellow in Listing 4, is the code I used to setup my network and clients through docker:

```
docker network create -d bridge --subnet 172.20.0.0/16 cs2031

docker create --name CommandAndControl --cap-add=ALL -ti c:\cs2031:/cs2031
    java /bin/bash

docker create --name Broker --cap-add=ALL -ti c:\cs2031:/cs2031
```

```
        java /bin/bash
docker create --name Worker --cap-add=ALL -ti c:\cs2031:/cs2031
        java /bin/bash

docker network connect cs2031 CommandAndControl
docker network connect cs2031 Broker
docker network connect cs2031 Worker
```

Listing 4: Docker Setup Code

### 3.5 Packet Encoding

As seen in **Figures 2-4**, I ran a sample program where I inputted string “hello world” and asked for it to be printed 3 times. Using wireshark, I was able to capture the packets being sent. Throughout the whole process, there were 13 packets sent in total.

The following Figure 5, Figure 6, Figure 7, Figure 8, Figure 9, Figure 10, Figure 11, Figure 12, Figure 13, Figure 14, Figure 15, Figure 16, and Figure 17 show the each packet with a description of each packet:

*Note:* 172.20.0.3 is the Command and Control, 172.20.0.2 is the Broker, and 172.20.0.4 is the Worker

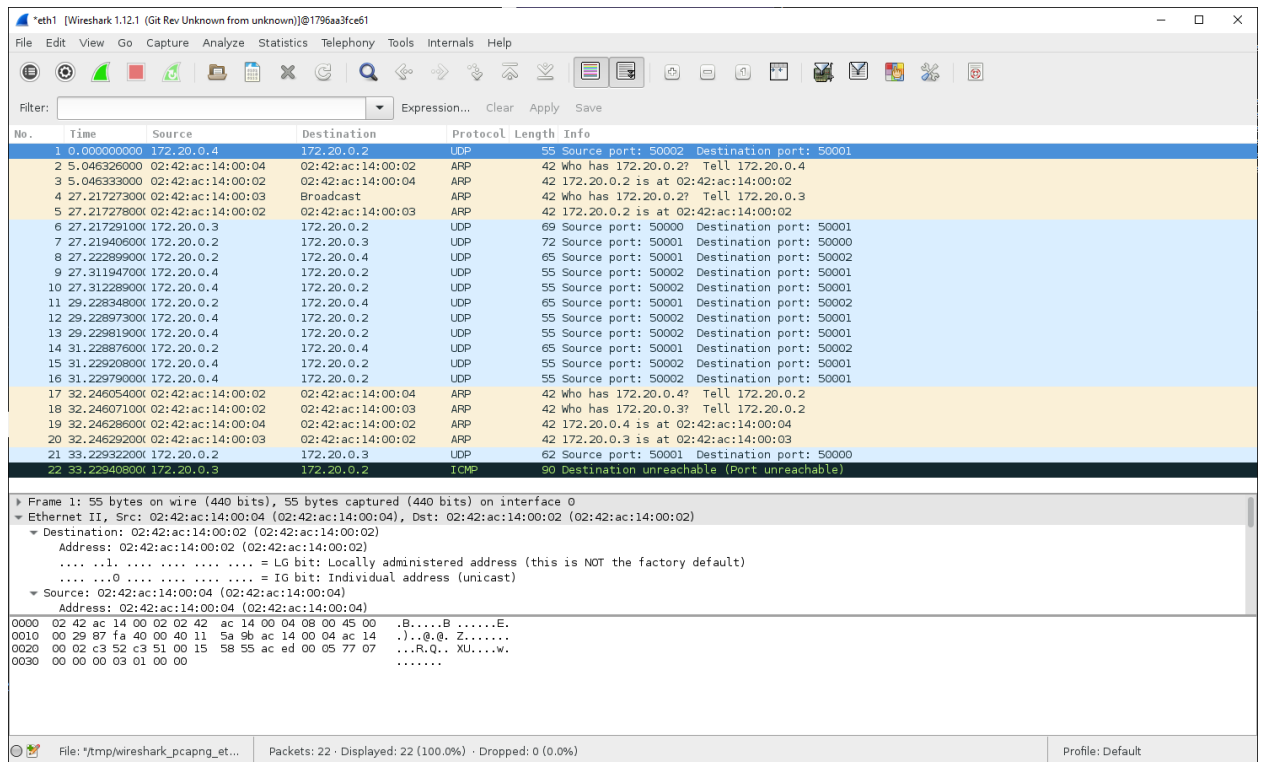


Figure 5: The Worker communicates its initial availability to the Broker. The file three final bytes are our three booleans. Ready to work is currently set as true, and received work and finished work are set to false.



Figure 6: The Command and Control sends the number of repetitions and then the string to be printed. In the bytecode, you can see 03 followed by the hex for hello world.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
2	5.046326000	02:42:ac:14:00:04	02:42:ac:14:00:02	ARP	42	Who has 172.20.0.2? Tell 172.20.0.4
3	5.046333000	02:42:ac:14:00:02	02:42:ac:14:00:04	ARP	42	172.20.0.2 is at 02:42:ac:14:00:02
4	27.217273000	02:42:ac:14:00:03	Broadcast	ARP	42	Who has 172.20.0.2? Tell 172.20.0.3
5	27.217278000	02:42:ac:14:00:02	02:42:ac:14:00:03	ARP	42	172.20.0.2 is at 02:42:ac:14:00:02
6	27.217291000	172.20.0.3	172.20.0.2	UDP	69	Source port: 50000 Destination port: 50001
7	27.219406000	172.20.0.2	172.20.0.3	UDP	72	Source port: 50001 Destination port: 50000
8	27.222899000	172.20.0.2	172.20.0.4	UDP	65	Source port: 50001 Destination port: 50002
9	27.311947000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
10	27.312289000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
11	29.228348000	172.20.0.2	172.20.0.4	UDP	65	Source port: 50001 Destination port: 50002
12	29.228973000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
13	29.229819000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
14	31.228876000	172.20.0.2	172.20.0.4	UDP	65	Source port: 50001 Destination port: 50002
15	31.229208000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
16	31.229790000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
17	32.246054000	02:42:ac:14:00:02	02:42:ac:14:00:04	ARP	42	Who has 172.20.0.4? Tell 172.20.0.2
18	32.246071000	02:42:ac:14:00:02	02:42:ac:14:00:03	ARP	42	Who has 172.20.0.3? Tell 172.20.0.2
19	32.246286000	02:42:ac:14:00:04	02:42:ac:14:00:02	ARP	42	172.20.0.4 is at 02:42:ac:14:00:04
20	32.246292000	02:42:ac:14:00:03	02:42:ac:14:00:02	ARP	42	172.20.0.3 is at 02:42:ac:14:00:03
21	33.229322000	172.20.0.2	172.20.0.3	UDP	62	Source port: 50001 Destination port: 50000
22	33.229408000	172.20.0.3	172.20.0.2	ICMP	90	Destination unreachable (Port unreachable)

Frame 6: 69 bytes on wire (552 bits), 69 bytes captured (552 bits) on interface 0  
 Ethernet II, Src: 02:42:ac:14:00:03 (02:42:ac:14:00:03), Dst: 02:42:ac:14:00:02 (02:42:ac:14:00:02)  
 Destination: 02:42:ac:14:00:02 (02:42:ac:14:00:02)  
 Address: 02:42:ac:14:00:02 (02:42:ac:14:00:02)  
 ....1. .... = LG bit: Locally administered address (this is NOT the factory default)  
 ....0. .... = IG bit: Individual address (unicast)  
 Source: 02:42:ac:14:00:03 (02:42:ac:14:00:03)  
 Address: 02:42:ac:14:00:03 (02:42:ac:14:00:03)  
 0000 02 42 ac 14 00 02 02 42 ac 14 00 03 08 00 45 00 .B....B .....E.  
 0010 00 3f 9f b3 40 00 40 11 42 45 ac 14 00 03 ac 14 .7..@.B .....  
 0020 00 02 c3 51 00 23 58 62 ac ed 00 05 77 15 ...P.Q.# Xb...w.  
 0030 00 00 00 04 00 00 00 03 00 0b 68 65 6c 6c 6f 20 .....Xb...hello  
 0040 7f 6f 72 6c 64 world

Figure 6: The Command and Control sends the number of repetitions and then the string to be printed. In the bytecode, you can see 03 followed by the hex for hello world.

Figure 7: The Broker then sends an acknowledgment to the Command and Control saying that it received the string.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
2	5.046326000	02:42:ac:14:00:04	02:42:ac:14:00:02	ARP	42	Who has 172.20.0.2? Tell 172.20.0.4
3	5.046333000	02:42:ac:14:00:02	02:42:ac:14:00:04	ARP	42	172.20.0.2 is at 02:42:ac:14:00:02
4	27.217273000	02:42:ac:14:00:03	Broadcast	ARP	42	Who has 172.20.0.2? Tell 172.20.0.3
5	27.217278000	02:42:ac:14:00:02	02:42:ac:14:00:03	ARP	42	172.20.0.2 is at 02:42:ac:14:00:02
6	27.217291000	172.20.0.3	172.20.0.2	UDP	69	Source port: 50000 Destination port: 50001
7	27.219406000	172.20.0.2	172.20.0.3	UDP	72	Source port: 50001 Destination port: 50000
8	27.222899000	172.20.0.2	172.20.0.4	UDP	65	Source port: 50001 Destination port: 50002
9	27.311947000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
10	27.312289000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
11	29.228348000	172.20.0.2	172.20.0.4	UDP	65	Source port: 50001 Destination port: 50002
12	29.228973000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
13	29.229819000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
14	31.228876000	172.20.0.2	172.20.0.4	UDP	65	Source port: 50001 Destination port: 50002
15	31.229208000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
16	31.229790000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
17	32.246054000	02:42:ac:14:00:02	02:42:ac:14:00:04	ARP	42	Who has 172.20.0.4? Tell 172.20.0.2
18	32.246071000	02:42:ac:14:00:02	02:42:ac:14:00:03	ARP	42	Who has 172.20.0.3? Tell 172.20.0.2
19	32.246286000	02:42:ac:14:00:04	02:42:ac:14:00:02	ARP	42	172.20.0.4 is at 02:42:ac:14:00:04
20	32.246292000	02:42:ac:14:00:03	02:42:ac:14:00:02	ARP	42	172.20.0.3 is at 02:42:ac:14:00:03
21	33.229322000	172.20.0.2	172.20.0.3	UDP	62	Source port: 50001 Destination port: 50000
22	33.229408000	172.20.0.3	172.20.0.2	ICMP	90	Destination unreachable (Port unreachable)

Frame 7: 72 bytes on wire (576 bits), 72 bytes captured (576 bits) on interface 0  
 Ethernet II, Src: 02:42:ac:14:00:02 (02:42:ac:14:00:02), Dst: 02:42:ac:14:00:03 (02:42:ac:14:00:03)  
 Destination: 02:42:ac:14:00:03 (02:42:ac:14:00:03)  
 Address: 02:42:ac:14:00:03 (02:42:ac:14:00:03)  
 ....1. .... = LG bit: Locally administered address (this is NOT the factory default)  
 ....0. .... = IG bit: Individual address (unicast)  
 Source: 02:42:ac:14:00:02 (02:42:ac:14:00:02)  
 Address: 02:42:ac:14:00:02 (02:42:ac:14:00:02)  
 0000 02 42 ac 14 00 03 02 42 ac 14 00 02 08 00 45 00 .B....B .....E.  
 0010 00 3a 12 43 40 00 40 11 d0 42 ac 14 00 02 ac 14 ..C@.B .....  
 0020 00 03 c3 51 c3 50 00 26 58 65 ac ed 00 05 77 18 ...Q.P.& Xe...w.  
 0030 00 00 01 00 12 4f 4b 20 2d 20 43 6f 6d 6d 61 .....OK - Comma  
 0040 6e 64 50 61 63 66 65 74 ndPacket

Figure 7: The Broker then sends an acknowledgment to the Command and Control saying that it received the string.

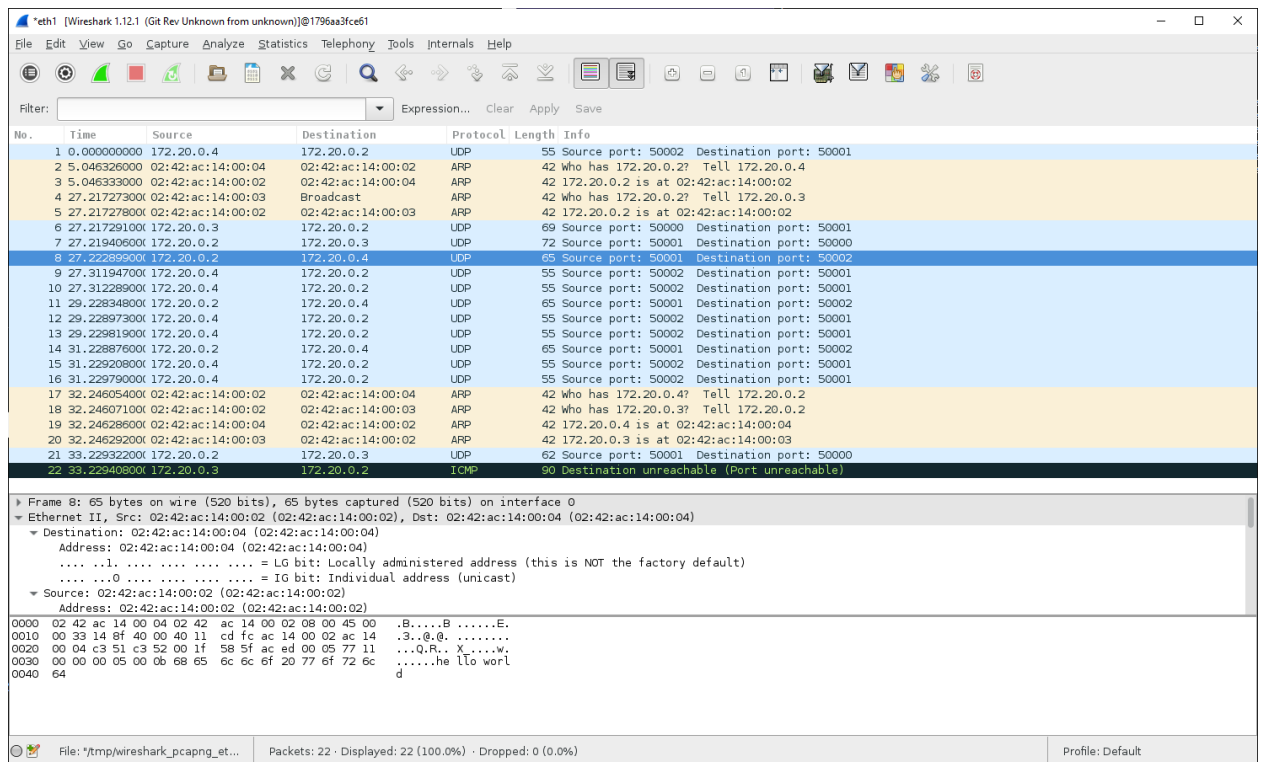


Figure 8: The Broker sends the string to the Worker to print.

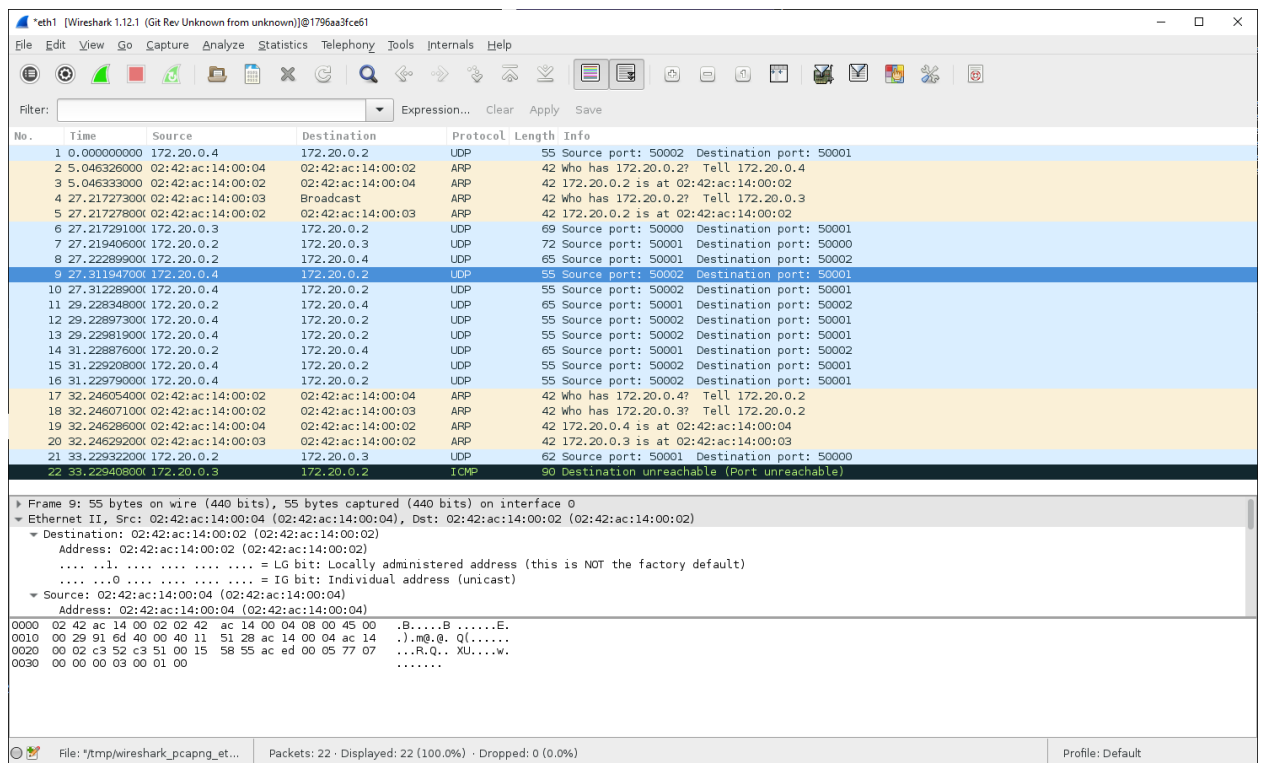


Figure 9: The worker then sends a response to the Broker saying that it is no longer available to work, it has received the string to print, and that it hasn't completed the task.

Figure 10 shows a Wireshark capture of network traffic. The packet list shows 22 packets. The selected packet (No. 22) is an ICMP Destination unreachable (Port unreachable) message from 172.20.0.3 to 172.20.0.2. The packet details pane shows the Ethernet II header, Internet Protocol Version 4 header, and ICMP header. The packet bytes pane shows the raw data in hexadecimal and ASCII.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
2	5.046326000	02:42:ac:14:00:04	02:42:ac:14:00:02	ARP	42	Who has 172.20.0.2? Tell 172.20.0.4
3	5.046333000	02:42:ac:14:00:02	02:42:ac:14:00:04	ARP	42	172.20.0.2 is at 02:42:ac:14:00:02
4	27.217273000	02:42:ac:14:00:03	Broadcast	ARP	42	Who has 172.20.0.2? Tell 172.20.0.3
5	27.217278000	02:42:ac:14:00:02	02:42:ac:14:00:03	ARP	42	172.20.0.2 is at 02:42:ac:14:00:02
6	27.217291000	172.20.0.3	172.20.0.2	UDP	69	Source port: 50000 Destination port: 50001
7	27.219406000	172.20.0.2	172.20.0.3	UDP	72	Source port: 50001 Destination port: 50000
8	27.222899000	172.20.0.2	172.20.0.4	UDP	65	Source port: 50001 Destination port: 50002
9	27.311947000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
10	27.312289000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
11	29.228348000	172.20.0.2	172.20.0.4	UDP	65	Source port: 50001 Destination port: 50002
12	29.228973000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
13	29.229819000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
14	31.228876000	172.20.0.2	172.20.0.4	UDP	65	Source port: 50001 Destination port: 50002
15	31.229208000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
16	31.229790000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
17	32.246054000	02:42:ac:14:00:02	02:42:ac:14:00:04	ARP	42	Who has 172.20.0.4? Tell 172.20.0.2
18	32.246071000	02:42:ac:14:00:02	02:42:ac:14:00:03	ARP	42	Who has 172.20.0.3? Tell 172.20.0.2
19	32.246286000	02:42:ac:14:00:04	02:42:ac:14:00:02	ARP	42	172.20.0.4 is at 02:42:ac:14:00:04
20	32.246292000	02:42:ac:14:00:03	02:42:ac:14:00:02	ARP	42	172.20.0.3 is at 02:42:ac:14:00:03
21	33.229322000	172.20.0.2	172.20.0.3	UDP	62	Source port: 50001 Destination port: 50000
22	33.229408000	172.20.0.3	172.20.0.2	ICMP	90	Destination unreachable (Port unreachable)

Figure 10: The worker sends another response to the Broker saying that is now available to work, it has not received the string to print, and that it hasn't completed the task.

Figure 11 shows a Wireshark capture of network traffic. The packet list shows 22 packets. The selected packet (No. 22) is an ICMP Destination unreachable (Port unreachable) message from 172.20.0.3 to 172.20.0.2. The packet details pane shows the Ethernet II header, Internet Protocol Version 4 header, and ICMP header. The packet bytes pane shows the raw data in hexadecimal and ASCII.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
2	5.046326000	02:42:ac:14:00:04	02:42:ac:14:00:02	ARP	42	Who has 172.20.0.2? Tell 172.20.0.4
3	5.046333000	02:42:ac:14:00:02	02:42:ac:14:00:04	ARP	42	172.20.0.2 is at 02:42:ac:14:00:02
4	27.217273000	02:42:ac:14:00:03	Broadcast	ARP	42	Who has 172.20.0.2? Tell 172.20.0.3
5	27.217278000	02:42:ac:14:00:02	02:42:ac:14:00:03	ARP	42	172.20.0.2 is at 02:42:ac:14:00:02
6	27.217291000	172.20.0.3	172.20.0.2	UDP	69	Source port: 50000 Destination port: 50001
7	27.219406000	172.20.0.2	172.20.0.3	UDP	72	Source port: 50001 Destination port: 50000
8	27.222899000	172.20.0.2	172.20.0.4	UDP	65	Source port: 50001 Destination port: 50002
9	27.311947000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
10	27.312289000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
11	29.228348000	172.20.0.2	172.20.0.4	UDP	65	Source port: 50001 Destination port: 50002
12	29.228973000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
13	29.229819000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
14	31.228876000	172.20.0.2	172.20.0.4	UDP	65	Source port: 50001 Destination port: 50002
15	31.229208000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
16	31.229790000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
17	32.246054000	02:42:ac:14:00:02	02:42:ac:14:00:04	ARP	42	Who has 172.20.0.4? Tell 172.20.0.2
18	32.246071000	02:42:ac:14:00:02	02:42:ac:14:00:03	ARP	42	Who has 172.20.0.3? Tell 172.20.0.2
19	32.246286000	02:42:ac:14:00:04	02:42:ac:14:00:02	ARP	42	172.20.0.4 is at 02:42:ac:14:00:04
20	32.246292000	02:42:ac:14:00:03	02:42:ac:14:00:02	ARP	42	172.20.0.3 is at 02:42:ac:14:00:03
21	33.229322000	172.20.0.2	172.20.0.3	UDP	62	Source port: 50001 Destination port: 50000
22	33.229408000	172.20.0.3	172.20.0.2	ICMP	90	Destination unreachable (Port unreachable)

Figure 11: Same as Figure 8

Wireshark 1.12.1 (Git Rev Unknown from unknown) @1796a3fce61

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help

Filter: Expression... Clear Apply Save

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
2	5.046326000	02:42:ac:14:00:04	02:42:ac:14:00:02	ARP	42	Who has 172.20.0.2? Tell 172.20.0.4
3	5.046333000	02:42:ac:14:00:02	02:42:ac:14:00:04	ARP	42	172.20.0.2 is at 02:42:ac:14:00:02
4	27.217273000	02:42:ac:14:00:03	Broadcast	ARP	42	Who has 172.20.0.2? Tell 172.20.0.3
5	27.217278000	02:42:ac:14:00:02	02:42:ac:14:00:03	ARP	42	172.20.0.2 is at 02:42:ac:14:00:02
6	27.217291000	172.20.0.3	172.20.0.2	UDP	69	Source port: 50000 Destination port: 50001
7	27.219406000	172.20.0.2	172.20.0.3	UDP	72	Source port: 50001 Destination port: 50000
8	27.222899000	172.20.0.2	172.20.0.4	UDP	65	Source port: 50001 Destination port: 50002
9	27.311947000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
10	27.312289000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
11	29.228348000	172.20.0.2	172.20.0.4	UDP	65	Source port: 50001 Destination port: 50002
12	29.228973000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
13	29.229819000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
14	31.228876000	172.20.0.2	172.20.0.4	UDP	65	Source port: 50001 Destination port: 50002
15	31.229208000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
16	31.229790000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
17	32.246054000	02:42:ac:14:00:02	02:42:ac:14:00:04	ARP	42	Who has 172.20.0.4? Tell 172.20.0.2
18	32.246071000	02:42:ac:14:00:02	02:42:ac:14:00:03	ARP	42	Who has 172.20.0.3? Tell 172.20.0.2
19	32.246286000	02:42:ac:14:00:04	02:42:ac:14:00:02	ARP	42	172.20.0.4 is at 02:42:ac:14:00:04
20	32.246292000	02:42:ac:14:00:03	02:42:ac:14:00:02	ARP	42	172.20.0.3 is at 02:42:ac:14:00:03
21	33.229322000	172.20.0.2	172.20.0.3	UDP	62	Source port: 50001 Destination port: 50000
22	33.229408000	172.20.0.3	172.20.0.2	ICMP	90	Destination unreachable (Port unreachable)

Frame 12: 55 bytes on wire (440 bits), 55 bytes captured (440 bits) on interface 0  
Ethernet II, Src: 02:42:ac:14:00:04 (02:42:ac:14:00:04), Dst: 02:42:ac:14:00:02 (02:42:ac:14:00:02)  
Destination: 02:42:ac:14:00:02 (02:42:ac:14:00:02)  
Address: 02:42:ac:14:00:02 (02:42:ac:14:00:02)  
... .. = LG bit: Locally administered address (this is NOT the factory default)  
... .. = IG bit: Individual address (unicast)  
Source: 02:42:ac:14:00:04 (02:42:ac:14:00:04)  
Address: 02:42:ac:14:00:04 (02:42:ac:14:00:04)

0000 02 42 ac 14 00 02 02 42 ac 14 00 04 08 00 45 00 .B....B .....E.  
0010 00 29 92 09 40 00 40 11 50 8c ac 14 00 04 ac 14 ..).@.@.P.....  
0020 00 02 c3 52 c3 51 00 15 58 55 ac ed 00 05 77 07 ...R.Q.. XU....w.  
0030 00 00 00 03 00 01 00 .....  
.....

File: /tmp/wireshark\_pcapng\_et... Packets: 22 · Displayed: 22 (100.0%) · Dropped: 0 (0.0%) Profile: Default

Figure 12: Same as Figure 9

Wireshark 1.12.1 (Git Rev Unknown from unknown) @1796a3fce61

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help

Filter: Expression... Clear Apply Save

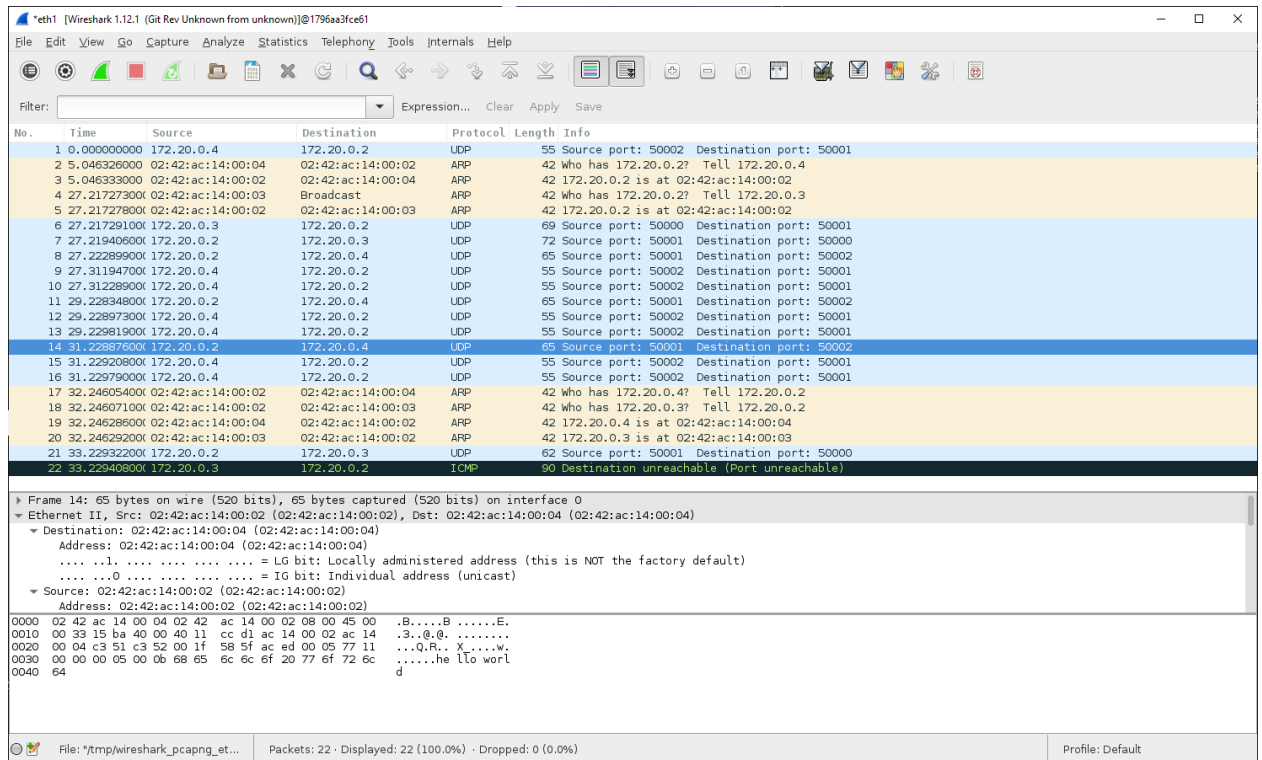
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
2	5.046326000	02:42:ac:14:00:04	02:42:ac:14:00:02	ARP	42	Who has 172.20.0.2? Tell 172.20.0.4
3	5.046333000	02:42:ac:14:00:02	02:42:ac:14:00:04	ARP	42	172.20.0.2 is at 02:42:ac:14:00:02
4	27.217273000	02:42:ac:14:00:03	Broadcast	ARP	42	Who has 172.20.0.2? Tell 172.20.0.3
5	27.217278000	02:42:ac:14:00:02	02:42:ac:14:00:03	ARP	42	172.20.0.2 is at 02:42:ac:14:00:02
6	27.217291000	172.20.0.3	172.20.0.2	UDP	69	Source port: 50000 Destination port: 50001
7	27.219406000	172.20.0.2	172.20.0.3	UDP	72	Source port: 50001 Destination port: 50000
8	27.222899000	172.20.0.2	172.20.0.4	UDP	65	Source port: 50001 Destination port: 50002
9	27.311947000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
10	27.312289000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
11	29.228348000	172.20.0.2	172.20.0.4	UDP	65	Source port: 50001 Destination port: 50002
12	29.228973000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
13	29.229819000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
14	31.228876000	172.20.0.2	172.20.0.4	UDP	65	Source port: 50001 Destination port: 50002
15	31.229208000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
16	31.229790000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
17	32.246054000	02:42:ac:14:00:02	02:42:ac:14:00:04	ARP	42	Who has 172.20.0.4? Tell 172.20.0.2
18	32.246071000	02:42:ac:14:00:02	02:42:ac:14:00:03	ARP	42	Who has 172.20.0.3? Tell 172.20.0.2
19	32.246286000	02:42:ac:14:00:04	02:42:ac:14:00:02	ARP	42	172.20.0.4 is at 02:42:ac:14:00:04
20	32.246292000	02:42:ac:14:00:03	02:42:ac:14:00:02	ARP	42	172.20.0.3 is at 02:42:ac:14:00:03
21	33.229322000	172.20.0.2	172.20.0.3	UDP	62	Source port: 50001 Destination port: 50000
22	33.229408000	172.20.0.3	172.20.0.2	ICMP	90	Destination unreachable (Port unreachable)

Frame 13: 55 bytes on wire (440 bits), 55 bytes captured (440 bits) on interface 0  
Ethernet II, Src: 02:42:ac:14:00:04 (02:42:ac:14:00:04), Dst: 02:42:ac:14:00:02 (02:42:ac:14:00:02)  
Destination: 02:42:ac:14:00:02 (02:42:ac:14:00:02)  
Address: 02:42:ac:14:00:02 (02:42:ac:14:00:02)  
... .. = LG bit: Locally administered address (this is NOT the factory default)  
... .. = IG bit: Individual address (unicast)  
Source: 02:42:ac:14:00:04 (02:42:ac:14:00:04)  
Address: 02:42:ac:14:00:04 (02:42:ac:14:00:04)

0000 02 42 ac 14 00 02 02 42 ac 14 00 04 08 00 45 00 .B....B .....E.  
0010 00 29 92 0a 40 00 40 11 50 8b ac 14 00 04 ac 14 ..).@.@.P.....  
0020 00 02 c3 52 c3 51 00 15 58 55 ac ed 00 05 77 07 ...R.Q.. XU....w.  
0030 00 00 00 03 01 00 01 .....  
.....

File: /tmp/wireshark\_pcapng\_et... Packets: 22 · Displayed: 22 (100.0%) · Dropped: 0 (0.0%) Profile: Default

Figure 13: Same as Figure 10

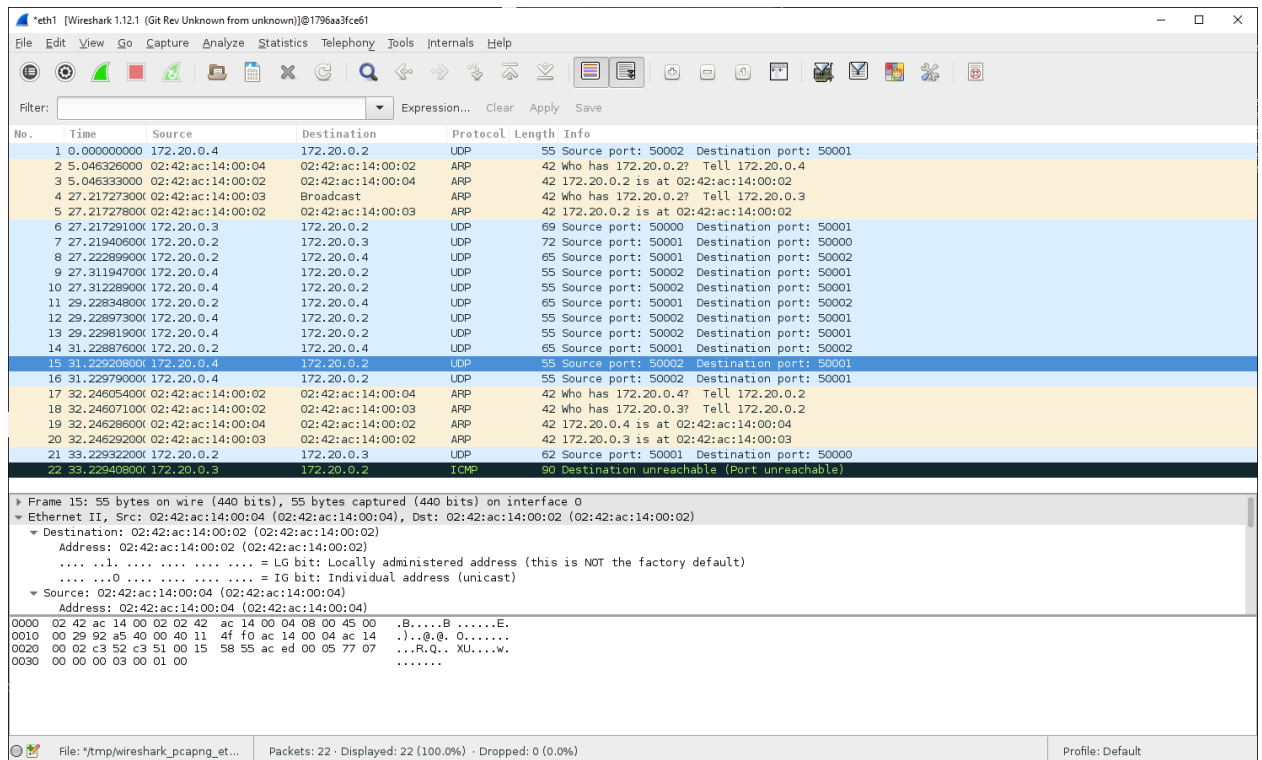


No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
2	5.046326000	02:42:ac:14:00:04	02:42:ac:14:00:02	ARP	42	Who has 172.20.0.2? Tell 172.20.0.4
3	5.046333000	02:42:ac:14:00:02	02:42:ac:14:00:04	ARP	42	172.20.0.2 is at 02:42:ac:14:00:02
4	27.217273000	02:42:ac:14:00:03	Broadcast	ARP	42	Who has 172.20.0.2? Tell 172.20.0.3
5	27.217278000	02:42:ac:14:00:02	02:42:ac:14:00:03	ARP	42	172.20.0.2 is at 02:42:ac:14:00:02
6	27.217291000	172.20.0.3	172.20.0.2	UDP	69	Source port: 50000 Destination port: 50001
7	27.219406000	172.20.0.2	172.20.0.3	UDP	72	Source port: 50001 Destination port: 50000
8	27.222899000	172.20.0.2	172.20.0.4	UDP	65	Source port: 50001 Destination port: 50002
9	27.311947000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
10	27.312289000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
11	29.228348000	172.20.0.2	172.20.0.4	UDP	65	Source port: 50001 Destination port: 50002
12	29.228973000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
13	29.229819000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
14	31.228876000	172.20.0.2	172.20.0.4	UDP	65	Source port: 50001 Destination port: 50002
15	31.229208000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
16	31.229790000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
17	32.246054000	02:42:ac:14:00:02	02:42:ac:14:00:04	ARP	42	Who has 172.20.0.4? Tell 172.20.0.2
18	32.246071000	02:42:ac:14:00:02	02:42:ac:14:00:03	ARP	42	Who has 172.20.0.3? Tell 172.20.0.2
19	32.246286000	02:42:ac:14:00:04	02:42:ac:14:00:02	ARP	42	172.20.0.4 is at 02:42:ac:14:00:04
20	32.246292000	02:42:ac:14:00:03	02:42:ac:14:00:02	ARP	42	172.20.0.3 is at 02:42:ac:14:00:03
21	33.229322000	172.20.0.2	172.20.0.3	UDP	62	Source port: 50001 Destination port: 50000
22	33.229408000	172.20.0.3	172.20.0.2	ICMP	90	Destination unreachable (Port unreachable)

Frame 14: 65 bytes on wire (520 bits), 65 bytes captured (520 bits) on interface 0  
Ethernet II, Src: 02:42:ac:14:00:02 (02:42:ac:14:00:02), Dst: 02:42:ac:14:00:04 (02:42:ac:14:00:04)  
Destination: 02:42:ac:14:00:04 (02:42:ac:14:00:04)  
Address: 02:42:ac:14:00:04 (02:42:ac:14:00:04)  
.....1. .... = LG bit: Locally administered address (this is NOT the factory default)  
.....0. .... = IG bit: Individual address (unicast)  
Source: 02:42:ac:14:00:02 (02:42:ac:14:00:02)  
Address: 02:42:ac:14:00:02 (02:42:ac:14:00:02)

0000 02 42 ac 14 00 04 02 42 ac 14 00 02 08 00 45 00 .B....B .....E.  
0010 00 33 15 ba 40 00 40 11 cc d1 ac 14 00 02 ac 14 .3..@. ....  
0020 00 04 c3 51 c3 52 00 1f 58 5f ac ed 00 05 77 11 ...Q.R.. X....w.  
0030 00 00 00 05 00 06 68 65 6c 6c 6f 20 77 6f 72 6c .....he llo worl  
0040 64 d

Figure 14: Same as Figure 8



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
2	5.046326000	02:42:ac:14:00:04	02:42:ac:14:00:02	ARP	42	Who has 172.20.0.2? Tell 172.20.0.4
3	5.046333000	02:42:ac:14:00:02	02:42:ac:14:00:04	ARP	42	172.20.0.2 is at 02:42:ac:14:00:02
4	27.217273000	02:42:ac:14:00:03	Broadcast	ARP	42	Who has 172.20.0.2? Tell 172.20.0.3
5	27.217278000	02:42:ac:14:00:02	02:42:ac:14:00:03	ARP	42	172.20.0.2 is at 02:42:ac:14:00:02
6	27.217291000	172.20.0.3	172.20.0.2	UDP	69	Source port: 50000 Destination port: 50001
7	27.219406000	172.20.0.2	172.20.0.3	UDP	72	Source port: 50001 Destination port: 50000
8	27.222899000	172.20.0.2	172.20.0.4	UDP	65	Source port: 50001 Destination port: 50002
9	27.311947000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
10	27.312289000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
11	29.228348000	172.20.0.2	172.20.0.4	UDP	65	Source port: 50001 Destination port: 50002
12	29.228973000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
13	29.229819000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
14	31.228876000	172.20.0.2	172.20.0.4	UDP	65	Source port: 50001 Destination port: 50002
15	31.229208000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
16	31.229790000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
17	32.246054000	02:42:ac:14:00:02	02:42:ac:14:00:04	ARP	42	Who has 172.20.0.4? Tell 172.20.0.2
18	32.246071000	02:42:ac:14:00:02	02:42:ac:14:00:03	ARP	42	Who has 172.20.0.3? Tell 172.20.0.2
19	32.246286000	02:42:ac:14:00:04	02:42:ac:14:00:02	ARP	42	172.20.0.4 is at 02:42:ac:14:00:04
20	32.246292000	02:42:ac:14:00:03	02:42:ac:14:00:02	ARP	42	172.20.0.3 is at 02:42:ac:14:00:03
21	33.229322000	172.20.0.2	172.20.0.3	UDP	62	Source port: 50001 Destination port: 50000
22	33.229408000	172.20.0.3	172.20.0.2	ICMP	90	Destination unreachable (Port unreachable)

Frame 15: 55 bytes on wire (440 bits), 55 bytes captured (440 bits) on interface 0  
Ethernet II, Src: 02:42:ac:14:00:04 (02:42:ac:14:00:04), Dst: 02:42:ac:14:00:02 (02:42:ac:14:00:02)  
Destination: 02:42:ac:14:00:02 (02:42:ac:14:00:02)  
Address: 02:42:ac:14:00:02 (02:42:ac:14:00:02)  
.....1. .... = LG bit: Locally administered address (this is NOT the factory default)  
.....0. .... = IG bit: Individual address (unicast)  
Source: 02:42:ac:14:00:04 (02:42:ac:14:00:04)  
Address: 02:42:ac:14:00:04 (02:42:ac:14:00:04)

0000 02 42 ac 14 00 02 02 42 ac 14 00 04 08 00 45 00 .B....B .....E.  
0010 00 29 92 a5 40 00 40 11 4f f0 ac 14 00 04 ac 14 ..@. @. 0.....  
0020 00 02 c3 52 c3 51 00 15 58 55 ac ed 00 05 77 07 ...R.Q.. XU....w.  
0030 00 00 00 03 00 01 00 .....  
0040

Figure 15: Same as Figure 9



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
2	5.046326000	02:42:ac:14:00:04	02:42:ac:14:00:02	ARP	42	Who has 172.20.0.2? Tell 172.20.0.4
3	5.046333000	02:42:ac:14:00:02	02:42:ac:14:00:04	ARP	42	172.20.0.2 is at 02:42:ac:14:00:02
4	27.217273000	02:42:ac:14:00:03	Broadcast	ARP	42	Who has 172.20.0.2? Tell 172.20.0.3
5	27.217278000	02:42:ac:14:00:02	02:42:ac:14:00:03	ARP	42	172.20.0.2 is at 02:42:ac:14:00:02
6	27.217291000	172.20.0.3	172.20.0.2	UDP	69	Source port: 50000 Destination port: 50001
7	27.219406000	172.20.0.2	172.20.0.3	UDP	72	Source port: 50001 Destination port: 50000
8	27.222899000	172.20.0.2	172.20.0.4	UDP	65	Source port: 50001 Destination port: 50002
9	27.311947000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
10	27.312289000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
11	29.228348000	172.20.0.2	172.20.0.4	UDP	65	Source port: 50001 Destination port: 50002
12	29.228973000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
13	29.229819000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
14	31.228876000	172.20.0.2	172.20.0.4	UDP	65	Source port: 50001 Destination port: 50002
15	31.229208000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
16	31.229790000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
17	32.246054000	02:42:ac:14:00:02	02:42:ac:14:00:04	ARP	42	Who has 172.20.0.4? Tell 172.20.0.2
18	32.246071000	02:42:ac:14:00:02	02:42:ac:14:00:03	ARP	42	Who has 172.20.0.3? Tell 172.20.0.2
19	32.246286000	02:42:ac:14:00:04	02:42:ac:14:00:02	ARP	42	172.20.0.4 is at 02:42:ac:14:00:04
20	32.246292000	02:42:ac:14:00:03	02:42:ac:14:00:02	ARP	42	172.20.0.3 is at 02:42:ac:14:00:03
21	33.229322000	172.20.0.2	172.20.0.3	UDP	62	Source port: 50001 Destination port: 50000
22	33.229408000	172.20.0.3	172.20.0.2	ICMP	90	Destination unreachable (Port unreachable)

Frame 16: 55 bytes on wire (440 bits), 55 bytes captured (440 bits) on interface 0  
 Ethernet II, Src: 02:42:ac:14:00:04 (02:42:ac:14:00:04), Dst: 02:42:ac:14:00:02 (02:42:ac:14:00:02)  
 Destination: 02:42:ac:14:00:02 (02:42:ac:14:00:02)  
 Address: 02:42:ac:14:00:02 (02:42:ac:14:00:02)  
 .... 1. .... = LG bit: Locally administered address (this is NOT the factory default)  
 .... 0. .... = IG bit: Individual address (unicast)  
 Source: 02:42:ac:14:00:04 (02:42:ac:14:00:04)  
 Address: 02:42:ac:14:00:04 (02:42:ac:14:00:04)

0000 02 42 ac 14 00 02 02 42 ac 14 00 04 08 00 45 00 .B....B .....E.  
 0010 00 29 92 a6 40 00 40 11 4f ef ac 14 00 04 ac 14 ..@. ....  
 0020 00 02 c3 52 c3 51 00 15 58 55 ac ed 00 05 77 07 ...R.Q.. XU...W.  
 0030 00 00 00 03 01 00 01 .....  
 .....

File: \*tmp/wireshark\_pcapng\_et... Packets: 22 · Displayed: 22 (100.0%) · Dropped: 0 (0.0%) Profile: Default

Figure 16: Same as Figure 10

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
2	5.046326000	02:42:ac:14:00:04	02:42:ac:14:00:02	ARP	42	Who has 172.20.0.2? Tell 172.20.0.4
3	5.046333000	02:42:ac:14:00:02	02:42:ac:14:00:04	ARP	42	172.20.0.2 is at 02:42:ac:14:00:02
4	27.217273000	02:42:ac:14:00:03	Broadcast	ARP	42	Who has 172.20.0.2? Tell 172.20.0.3
5	27.217278000	02:42:ac:14:00:02	02:42:ac:14:00:03	ARP	42	172.20.0.2 is at 02:42:ac:14:00:02
6	27.217291000	172.20.0.3	172.20.0.2	UDP	69	Source port: 50000 Destination port: 50001
7	27.219406000	172.20.0.2	172.20.0.3	UDP	72	Source port: 50001 Destination port: 50000
8	27.222899000	172.20.0.2	172.20.0.4	UDP	65	Source port: 50001 Destination port: 50002
9	27.311947000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
10	27.312289000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
11	29.228348000	172.20.0.2	172.20.0.4	UDP	65	Source port: 50001 Destination port: 50002
12	29.228973000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
13	29.229819000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
14	31.228876000	172.20.0.2	172.20.0.4	UDP	65	Source port: 50001 Destination port: 50002
15	31.229208000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
16	31.229790000	172.20.0.4	172.20.0.2	UDP	55	Source port: 50002 Destination port: 50001
17	32.246054000	02:42:ac:14:00:02	02:42:ac:14:00:04	ARP	42	Who has 172.20.0.4? Tell 172.20.0.2
18	32.246071000	02:42:ac:14:00:02	02:42:ac:14:00:03	ARP	42	Who has 172.20.0.3? Tell 172.20.0.2
19	32.246286000	02:42:ac:14:00:04	02:42:ac:14:00:02	ARP	42	172.20.0.4 is at 02:42:ac:14:00:04
20	32.246292000	02:42:ac:14:00:03	02:42:ac:14:00:02	ARP	42	172.20.0.3 is at 02:42:ac:14:00:03
21	33.229322000	172.20.0.2	172.20.0.3	UDP	62	Source port: 50001 Destination port: 50000
22	33.229408000	172.20.0.3	172.20.0.2	ICMP	90	Destination unreachable (Port unreachable)

Frame 21: 62 bytes on wire (496 bits), 62 bytes captured (496 bits) on interface 0  
 Ethernet II, Src: 02:42:ac:14:00:02 (02:42:ac:14:00:02), Dst: 02:42:ac:14:00:03 (02:42:ac:14:00:03)  
 Destination: 02:42:ac:14:00:03 (02:42:ac:14:00:03)  
 Address: 02:42:ac:14:00:03 (02:42:ac:14:00:03)  
 .... 1. .... = LG bit: Locally administered address (this is NOT the factory default)  
 .... 0. .... = IG bit: Individual address (unicast)  
 Source: 02:42:ac:14:00:02 (02:42:ac:14:00:02)  
 Address: 02:42:ac:14:00:02 (02:42:ac:14:00:02)

0000 02 42 ac 14 00 03 02 42 ac 14 00 02 08 00 45 00 .B....B .....E.  
 0010 00 30 13 07 40 00 40 11 cf 88 ac 14 00 02 ac 14 ..@. ....  
 0020 00 03 c3 51 c3 50 00 1c 58 5b ac ed 00 05 77 0e ...Q.P.. X[...W.  
 0030 00 00 00 01 00 08 63 6f 6d 70 6c 65 74 65 .....complete

File: \*tmp/wireshark\_pcapng\_et... Packets: 22 · Displayed: 22 (100.0%) · Dropped: 0 (0.0%) Profile: Default

Figure 17: The Broker then communicates with the Command and Control saying that the program has completed.

## 4 Summary

I was able to demonstrate my knowledge in the theory behind packets being sent from one machine into another and subsequently implementing that knowledge to create a program that allowed for a user to send out a task from a Command and Control to a Broker that then divvied up the tasks to a scalable amount of Workers.

I was also able to capture the data passing through the different packets and analyze them.

Through this process I became comfortable using docker to create multiple virtual instances that communicated over a virtual interface. I also became comfortable writing in latex as it was something I had never

## 5 Reflection

This was the first assignment done to this scale that I have ever done. I spent about 12 hours on the theory behind the code, wrapping my head around what exactly I needed to have my program do. I then spent another 10 or so hours writing the code while also debugging any errors I had and learning how threads worked in Java. Then I spent about 6 hours writing this document. That makes about 28 hours in total on this assignment.

Clearly thing through the problem at the beginning helped tremendously when writing my actual code as I knew what I needed to do and could focus on writing and then debugging. I didn't need to spend time thinking about the problem and so I was more efficient. What I should've done is as I was working on the different sections, I should've worked a little bit on this write-up as it would've been easier to write things had I worked on it progressively. Next time, I will change that and work on it as I go along.

Overall I learned a lot when it comes to Java, networking and communication, Docker, and Latex. I thoroughly enjoyed this assignment.

## References

- [1] Microsoft Encarta. Node (networking), 2011.
- [2] Microsoft. Windows Network Architecture and the OSI Model, 2017.
- [3] William Stallings. *Business Data Communication*. Upper Saddle River, N.J. : Prentice Hall, 4th edition, 2001.