Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

CSU33031 Computer Networks
Assignment #1: File Retrieval Protocol

Dylan Fitzpatrick, Std# 20331794

October 27, 2022

Contents

1 Introduction

The aim of this assignment is to provide a means for one or more clients to retrieve files from one or more workers based on User Datagram Protocol (UDP) packets. This is done through the use of an ingress node which communicates with both the client and worker nodes.

2 Theory of Topic

In this section, I will describe the concepts and protocols that were used to realise a solution. This section will outline the role of each of the components in achieving the final product as well as the theory behind how packets are created and sent.

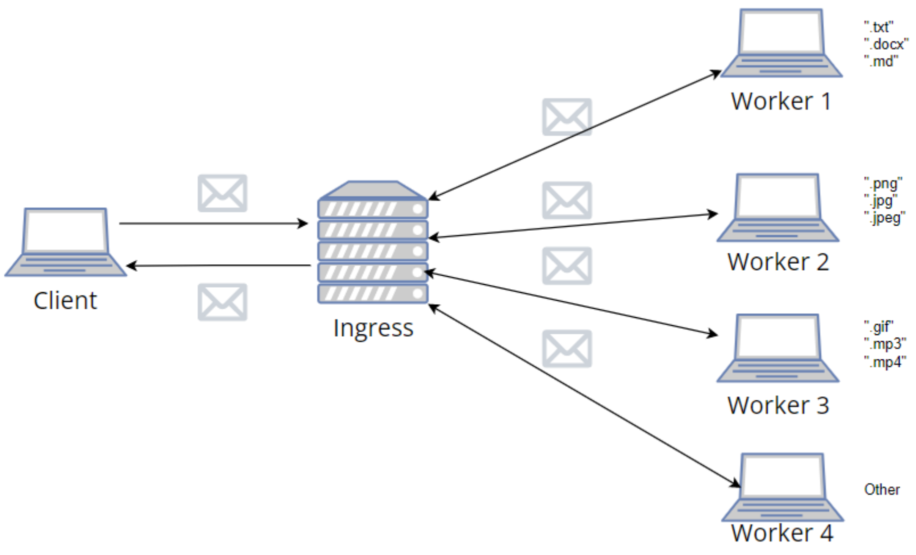


Figure 1: The topology of the final product. The figure shows the client sending the request for the file and receiving the file from ingress. The figure also shows the file types which correspond to each worker (e.g. '.txt', '.docx', '.md' for worker 1.)

2.1 User Datagram Protocol (UDP)

User Datagram Protocol (UDP) is a method for transferring data between two computers across a network. User Datagram Protocol (UDP) does not require previous communication, which is beneficial for time-sensitive communications.

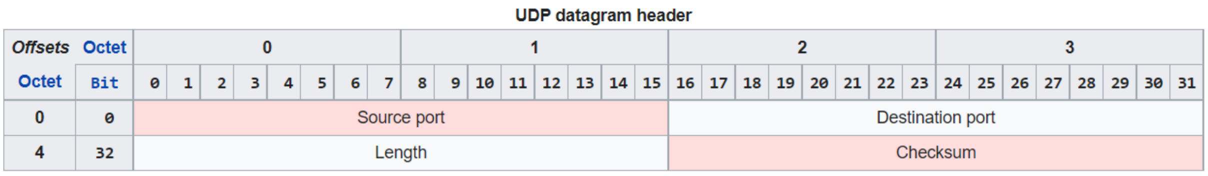
User Datagram Protocol (UDP) uses Internet Protocol (IP) to get a datagram packet between two computers. The data is encapsulated in a packet and header information is added. This consists of the source port, destination port, packet length and checksum. See Figure 2.

User Datagram Protocol (UDP) allows for packets to be sent and received in a different order than they were transmitted in.

2.1.1 Applications of UDP

- Voice and video traffic
- Lossless packet transmission
- QUIC protocol is built on top of UDP

- Gaming



UDP datagram header

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Source port																Destination port															
4	32	Length																Checksum															

Figure 2: The composition of a UDP header. Source: Wikipedia

2.1.2 Advantages of UDP

- Does not need a connection to be established/maintained
- Checksum used for error detection
- Fast with no need to wait on acknowledgements
- Small packet sizes and small header (8 bytes)

2.1.3 Disdvantages of UDP

- Connectionless and unreliable
- No guarantee packets will be sent
- No flow control or acknowledgements
- No error control so packets may be silently dropped

2.2 Client

The client node issues requests for a file to the ingress node. The request is sent in the form of a packet. The client will then receive the requested file from the ingress node upon completion.

2.3 Ingress

The ingress node processes the request made by the client, in this solution, it examines the file type that has been requested, and then decides which workers to send the request to based on the file type. If the file is found by the worker, ingress receives the file first, in the form of a packet, and then sends it on to the client.

2.4 Workers

The worker nodes process the request sent by the ingress node. If the file is found, the worker will send it to the ingress node. If the file size is very large, the worker splits the file up into several smaller packets, which then get reassembled after they have been successfully sent.

2.5 Communication and Packet Description

The data is "packetised" by encapsulating it into a frame or cell. A header is added to the data. The header consists of information about the packet and goes in front of the packet. The checksum allows the integrity of the packet header and payload to be verified. The packets are then sent across the network, the destination of the packet is known due to the content in the header.

3 Implementation

This section will outline how the client, ingress and worker nodes were implemented. Each of the three were implemented using their own Java classes. As well as this there are two types of packet content which should be noted. The type FILEREQ represents the requesting of a file. The type FILEINFO represents the actual file content in the form of a packet. This section will explain how the project was programmed successfully.

3.1 Client

```
request = new GetPacketContent(fileName).toDatagramPacket();
request.setSocketAddress(dstAddress);
Thread.sleep(500);
socket.send(request);
```

Listing 1: fileName is set as type FILEREQ and converted to a packet. The socket address is then set to ingress using the container name and the port number. The thread then sleeps to implement a delay and the request is then sent to ingress.

The first thing that is seen when running the client is the user being asked the name of the file they would like to retrieve. If the user types "exit", the program is terminated. Once the file name is typed, the String is converted to a datagram packet using a method called toDatagramPacket(). This method uses ByteArrayOutputStream and ObjectOutputStream and writes the file name to a byte array. A packet is then created from this byte array with header content containing the byte array and the length of the byte array. The socket address is then set to ingress using the setSocketAddress() method from the DatagramPacket library. The socket address is set using the Docker container name as well as ingress' port number which is 50000. See Listing 1.

The client will know it has received the file once it receives a packet of type FILEINFO. A message is then displayed to the user confirming the file has been received.

3.2 Ingress

```
if(content.getType()==PacketContent.FILEINFO){
    System.out.println("———\n\nReceived packet from worker " + workerNumber +
                                                                        "\n");
    packet.setSocketAddress(dstAddressClient);
    socket.send(packet);
}
```

Listing 2: Ingress receiving file from worker. Worker sends a packet of type FILEINFO to ingress. A message is printed confirming that a packet has been received from one of the workers. The socket address is set to client and the packet is sent to client.

Ingress first determines whether the packet it receives is of type FILEREQ or of type FILEINFO.

If a packet of type FILEREQ is received, ingress knows that it must read in the file name and determine the file type, which will then tell it to which worker the file request should be sent. This is done by converting the packet content to type String and then determining what the String ends with. The socket address for the correct worker is then set. Each worker has the same port number (20000), the difference is found in the container name for each worker.

If a packet of type FILEINFO is received, ingress has received the file. Ingress then forwards the file onto client, the socket address for client is set using port number 14430 and client's container name.

3.3 Workers

```
fileContent = new FileInfoContent(fileName, size, buffer);
Thread.sleep(500);
System.out.println("——\n\nSending file " + fileContent.getFileName()
                   + " to ingress\n");
Thread.sleep(500);
DatagramPacket response;
response = fileContent.toDatagramPacket();
response.setSocketAddress(dstAddressIngress);
Thread.sleep(500);
socket.send(response);
System.out.println("——\n\nPacket sent \n");
```

Listing 3: Worker sending file to ingress in a single packet. The file is stored in variable fileContent. fileContent is packetised and sent to ingress.

There are a total of 4 workers, each containing different file types:

- Worker 1: ".txt", ".docx", ".md"
- Worker 2: ".png", ".jpg", ".jpeg"
- Worker 3: ".gif", ".mp3", ".mp4"
- Worker 4: any other file type

The workers take in a FILEREQ packet and search for the file. If the file cannot be found, "File (file name) not found." is printed to the console. If found, the name of the file is saved into a variable of type File and a buffer is created. The buffer is a byte array the same length as the file. The size of the file is read using FileInputStream on the buffer. If the file size is bigger than 65536 minus the length of the file name minus the size of the file (i.e. smaller than 65536 minus the header content), the file is split into several smaller packets, which are reconstructed after being sent. For readability purposes, the amount of split packets which could be sent was limited to 10, if more than 10 packets attempt to be sent, a message saying "File too big." is displayed. If the file is not to big it is sent all as one packet.

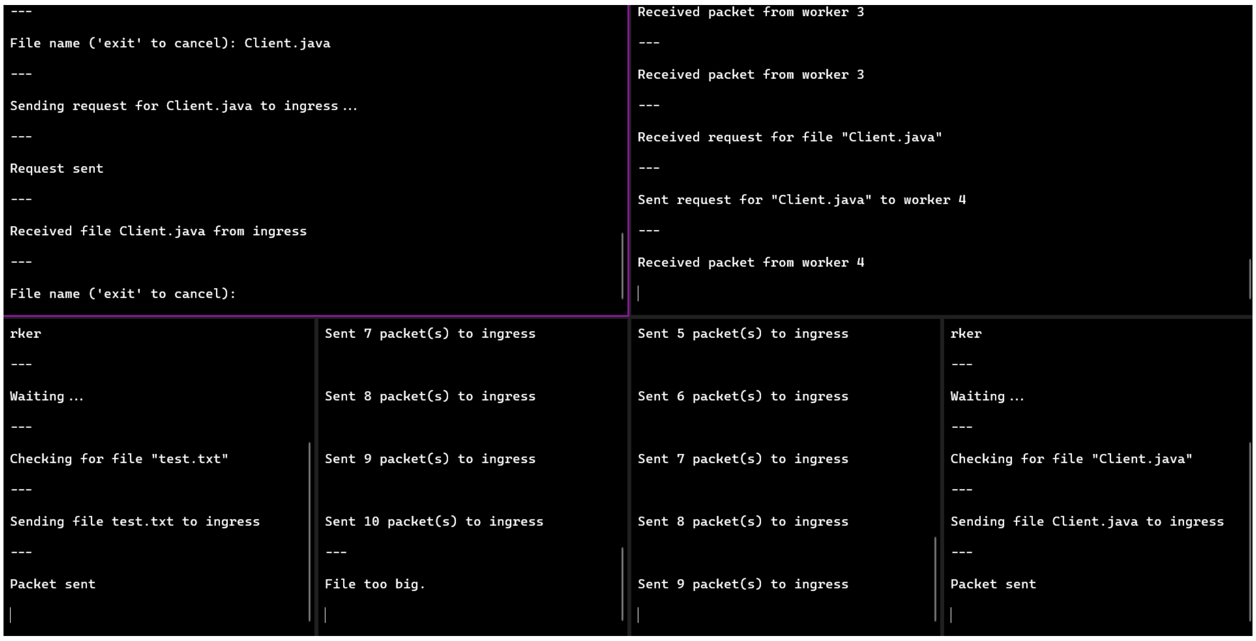
3.4 User Interaction

Upon starting the program, the user is greeted with a message asking them to input the name of the file, upon entering a valid file name, each component will display messages saying when it has successfully sent or received packets. A delay was implemented in the code to enhance readability, so the user can see each part of the process happening one by one rather than all at the same time. Figure 3 visualises the user experience. Ingress can be seen printing "packet received from worker X" each time a packet is sent from a worker to it. The terminal for worker 2 shows what happens when more than ten packets try to send, a message saying "File too big." is displayed. Worker 3 shows nine packets being sent. Workers 1 and 4 only send a single packet each.

3.5 Packet Encoding

In the class PacketContent, the type of packet (FILEREQ or FILEINFO) is decided in the method fromDatagramPacket() using ByteArrayInputStream and ObjectInputStream. ObjectInputStream determines the type from the header of the packet, 10 represents FILEREQ and 100 represents FILEINFO. The method toDatagramPacket() takes data and converts it to a Datagram Packet. This is done by writing the data to a byte array using ByteArrayOutputStream and ObjectOutputStream. The data is converted to a byte array and then converted to a Datagram Packet, where the header content is added.

The constructor in class FileInfoContent takes in the file name, the size of the file and the buffer. The buffer represents the byte array containing the content of the file. In the Worker class, the file is converted to a packet using the toDatagramPacket() method.



```
---
File name ('exit' to cancel): Client.java
---
Sending request for Client.java to ingress...
---
Request sent
---
Received file Client.java from ingress
---
File name ('exit' to cancel):
```

```
Received packet from worker 3
---
Received packet from worker 3
---
Received request for file "Client.java"
---
Sent request for "Client.java" to worker 4
---
Received packet from worker 4
```

```
rker
---
Waiting...
---
Checking for file "test.txt"
---
Sending file test.txt to ingress
---
Packet sent
```

```
Sent 7 packet(s) to ingress
Sent 8 packet(s) to ingress
Sent 9 packet(s) to ingress
Sent 10 packet(s) to ingress
---
File too big.
```

```
Sent 5 packet(s) to ingress
Sent 6 packet(s) to ingress
Sent 7 packet(s) to ingress
Sent 8 packet(s) to ingress
Sent 9 packet(s) to ingress
```

```
rker
---
Waiting...
---
Checking for file "Client.java"
---
Sending file Client.java to ingress
---
Packet sent
```

Figure 3: This is what the user sees in their terminal. Client can be seen on the top left. Ingress is seen on the top right. All 4 workers can be seen on the bottom in order from 1 to 4.

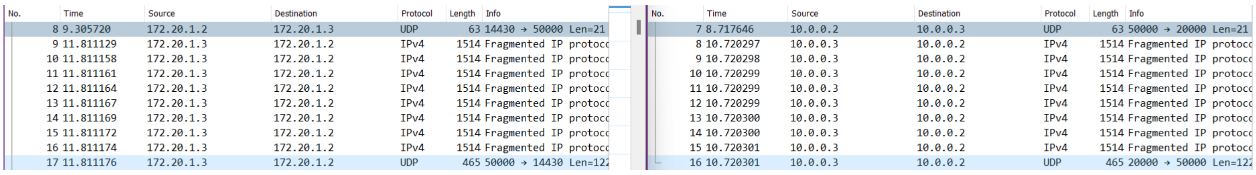
4 Discussion

One key strength of this program is its ability to work across more than one network. A different network is used for transfer of packets between client and ingress than is used for packet transfer between ingress and the workers. Figure 4 shows the packet transfer across different networks, as well as the deconstruction of the file into separate packets due to its large size.

Readability was also enhanced with the use of threads. The method Thread.sleep() is used multiple times in the program to delay the sending of packets, as well as delaying print statements to the console so that is is more clear to the user what way the packets are being sent.

One of the main weaknesses of this program is that it must be restarted if an incorrect file name is entered. This is because my main focus was the key functionality outlined in the assignment brief, error handling was not something I thought very deeply into implementing.

In order to enhance the readability of the pcap files, I limited the amount of packets which could be sent to 10, this could very easily be adjusted to lower the maximum number of packets or to allow all packets to be sent, however for larger files this would make it quite hard to read as the pcap file would become very lengthy.



No.	Time	Source	Destination	Protocol	Length	Info
8	9.305720	172.20.1.2	172.20.1.3	UDP	63	14430 → 50000 Len=21
9	11.811129	172.20.1.3	172.20.1.2	IPv4	1514	Fragmented IP protoco
10	11.811158	172.20.1.3	172.20.1.2	IPv4	1514	Fragmented IP protoco
11	11.811161	172.20.1.3	172.20.1.2	IPv4	1514	Fragmented IP protoco
12	11.811164	172.20.1.3	172.20.1.2	IPv4	1514	Fragmented IP protoco
13	11.811167	172.20.1.3	172.20.1.2	IPv4	1514	Fragmented IP protoco
14	11.811169	172.20.1.3	172.20.1.2	IPv4	1514	Fragmented IP protoco
15	11.811172	172.20.1.3	172.20.1.2	IPv4	1514	Fragmented IP protoco
16	11.811174	172.20.1.3	172.20.1.2	IPv4	1514	Fragmented IP protoco
17	11.811176	172.20.1.3	172.20.1.2	UDP	465	50000 → 14430 Len=122

No.	Time	Source	Destination	Protocol	Length	Info
7	8.717646	10.0.0.2	10.0.0.3	UDP	63	50000 → 20000 Len=21
8	10.720297	10.0.0.3	10.0.0.2	IPv4	1514	Fragmented IP protoco
9	10.720298	10.0.0.3	10.0.0.2	IPv4	1514	Fragmented IP protoco
10	10.720299	10.0.0.3	10.0.0.2	IPv4	1514	Fragmented IP protoco
11	10.720299	10.0.0.3	10.0.0.2	IPv4	1514	Fragmented IP protoco
12	10.720299	10.0.0.3	10.0.0.2	IPv4	1514	Fragmented IP protoco
13	10.720300	10.0.0.3	10.0.0.2	IPv4	1514	Fragmented IP protoco
14	10.720300	10.0.0.3	10.0.0.2	IPv4	1514	Fragmented IP protoco
15	10.720301	10.0.0.3	10.0.0.2	IPv4	1514	Fragmented IP protoco
16	10.720301	10.0.0.3	10.0.0.2	UDP	465	20000 → 50000 Len=122

Figure 4: Wireshark packet capture. Left window shows packets between client and ingress. Right window shows packets between ingress and workers

5 Summary

This report has outlined the theoretical aspects of each of the three key components found in this project, the client, the ingress node and the workers, as well as User Datagram Protocol (UDP) which this project is based on. As well as this, the implementation of each of the components has been described in detail. The user experience and design choices have also been outlined. The final project allows the user to request a file using the client window and retrieve this file from one of the workers, with ingress controlling the communication between the two.

6 Reflection

The recording of videos to show progression was very beneficial for me overall. It ensured that I was able to explain what I was doing correctly and encouraged me to work on the assignment consistently. It also helped me to get used to working with Docker and Wireshark, so now there is much less difficulty in capturing network traffic and obtaining pcap files. This project has also deepened my understanding of UDP greatly. I am now a lot more confident in my knowledge of protocols, packets and networking as a whole. Overall, this assignment took me approximately 30 hours to complete.