



CSU33031 Computer Networks

Assignment #1: File Retrieval Protocol

Yi Ren, 20304391

November 2, 2022

Contents

1	Introduction	1
2	Theory of Topic	2
2.1	Basic Functionality	2
2.1.1	Packets	3
2.1.2	Workers	3
2.1.3	User Input Handling	3
2.2	Flow Control	3
2.2.1	Stop-and-Wait ARQ	3
3	Implementation	4
3.1	Actor Classes	4
3.2	Packets Classes	5
3.3	Header	5
3.4	Sender Class	5
4	Discussion	6
5	Summary	6
6	Reflection	6

1 Introduction

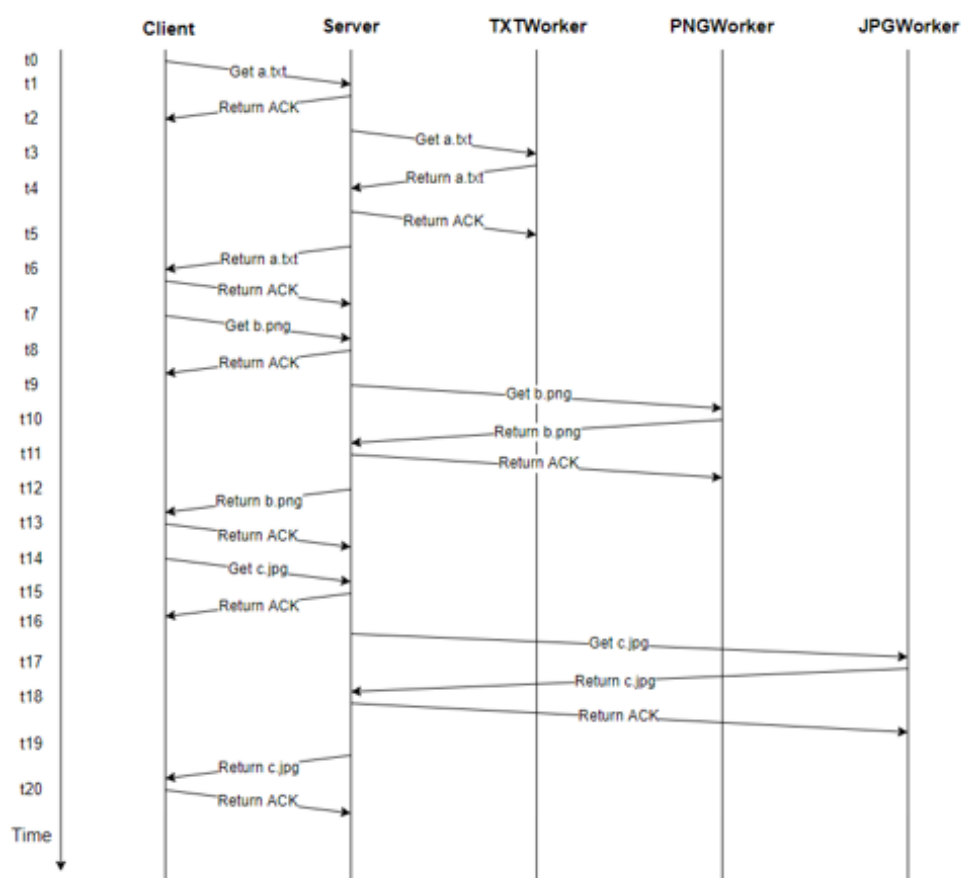
This assignment focuses on learning about protocol design and the information that is kept in a header to support functionality of a protocol. The aim of our protocol is to provide a mechanism to retrieve files from a service based on UDP datagrams. For my protocol, it involves three types of actors: one client, one server and three workers. It allows the client to issue requests for several types of files.

2 Theory of Topic

In this section, I will start with discussing the basic functionality. The additional functionality for the sake of efficiency or flow control will be discussed afterwards.

2.1 Basic Functionality

The basic functionality for this protocol is to complete the file retrieval process among the client, the server and the workers. The client issues file request to the server, the server receives the request and forwards it to the corresponding worker, and finally, transfers the file sent from the worker to the client. The client can then ask for another file or exit the retrieval system. Acknowledgement messages(ACK) are sent after each packet-transfer so that we are able to monitor the transfer from the console. For a special case of searching a nonexistent file, an error message will be forwarded to the client instead of the file.



Flow diagram for the overview of the protocol

```

root@5ccb3ed62154: /compnets/asmt1
root@5ccb3ed62154:/compnets/asmt1# java -cp . Client
Welcome to the file searching system!
Please enter the full file name for searching: b.png
[Received packet] ACK: OK - Received request
[Received packet] Error: File 'b.png' not found
Please enter the full file name for searching:
  
```

Console screenshot for the error message

2.1.1 Packets

There are four types of packets that will be sent among actors:

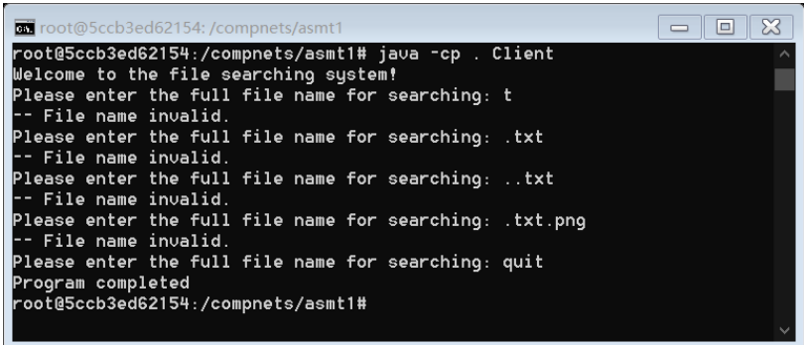
- A File Request Packet which contains the request issued by the client
- An Acknowledgement(ACK) Packet which includes the acknowledgement message of the packet receipt
- A File Info Packet which contains the file required by the client
- A String Packet which should be sent back to the client instead of a File Packet when the required file is not found

2.1.2 Workers

There are currently three workers involved in my program, each is in charge of a single type of files, one for TXT files, one for PNG files and another one for JPG files. The server determines which worker should be responsible for the incoming request by checking the file type.

2.1.3 User Input Handling

The program only allows user to search with full file names. Any input that does not ends with ".txt", ".png" or ".jpg" will be rejected on the client window. The request will not be packed into a File Request Packet until the user input is confirmed to be valid, so that it will not produce unnecessary work for the server and the workers.



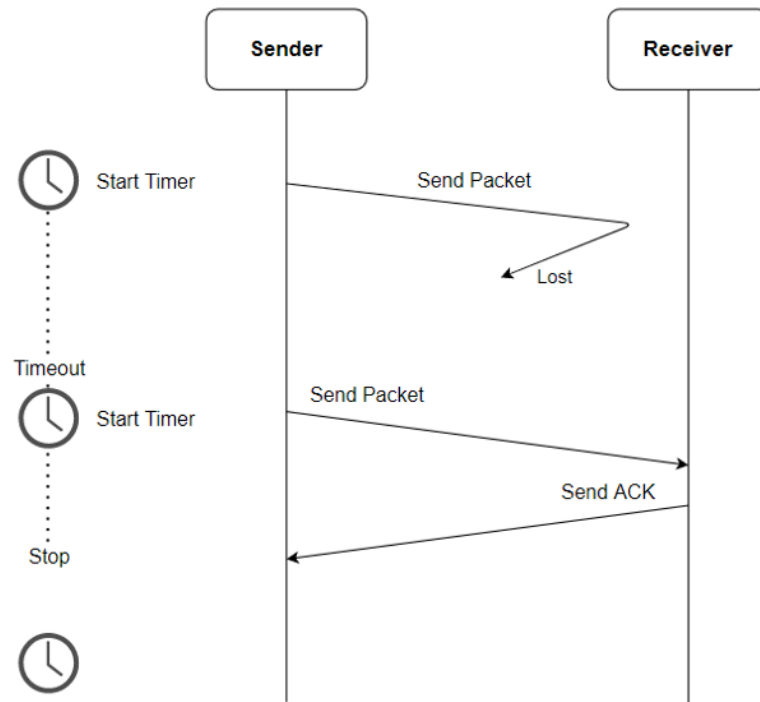
```
root@5ccb3ed62154: /compnets/asmt1
root@5ccb3ed62154:/compnets/asmt1# java -cp . Client
Welcome to the file searching system!
Please enter the full file name for searching: t
-- File name invalid.
Please enter the full file name for searching: .txt
-- File name invalid.
Please enter the full file name for searching: ..txt
-- File name invalid.
Please enter the full file name for searching: .txt.png
-- File name invalid.
Please enter the full file name for searching: quit
Program completed
root@5ccb3ed62154: /compnets/asmt1#
```

Console screenshot for the user input handling

2.2 Flow Control

2.2.1 Stop-and-Wait ARQ

In our basic functionality, we have completed a system using a Stop-and-Wait protocol that waits for and acknowledgement(ACK) after each packet transfer. However, we cannot solve the problem if the packet is dropped. Since the receiver will never get the packet, the ACK will never be sent. Since it never receives the acknowledgement it is expecting, the sender will never know and never take actions to send the packet again. Thus, I applied an automatic repeat-request(ARQ) protocol. For every packet that is sent, a timer will be created. When the timer expires, the corresponding packet will be sent again and a new timer started. The timer is destroyed when the ACK is received.



Flow diagram for ARQ protocol

```
root@5ccb3ed62154: /compnets/asmt1
Please enter the full file name for searching: message.txt
Packet sending failed. Will try again.
Packet sending failed. Will try again.
[Received packet] ACK: OK - Received request
```

Console screenshot for the ARQ protocol

3 Implementation

An overview and a description of the implementation are provided in this section. I employed the provided "Simplistic Java Example" as a starting point, including the Node class, Client and Server classes and the AckPacketContent and FileInfoPacket classes which extend the PacketContent class. They were modified in order to meet the requirements.

3.1 Actor Classes

The existing Client, Server classes and a new Worker class all extend the abstract class Node. And there are three classes that extend Worker, which are TXTWorker class, PNGWorker class and JPGWorkers. Each of the workers is assigned an individual port numbers, from 50002-50004.

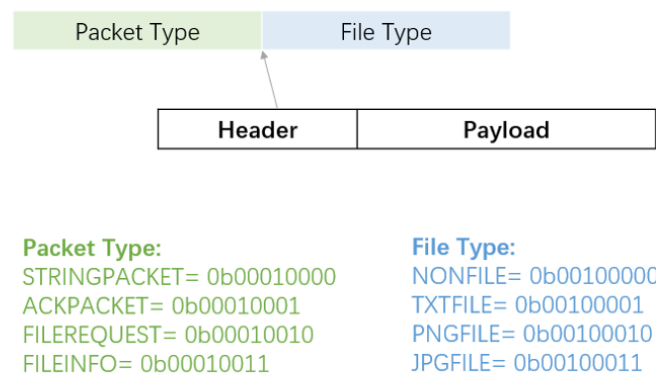
3.2 Packets Classes

There are four classes that extend the PacketContent class, in addition to the existing two classes, I have added a StringPacketContent for the error message and a FileRequestContent for the file request.

3.3 Header

I applied two bytes as the header of a packet.

- **Packet Type**
The first byte of the header plays the role of specifying the packet type. I have added two more types beside ACKPacket and FileInfoPacket to make the sorting of the incoming packets easier.
- **File Type**
The second byte of the header is specially designed for FileRequestPacket. Thus, for all other types of packets, their second byte of header should all be NONFILE(non-file-request). For the FileRequestPacket, the file type is written to the header when the client is packing the file request, specifying the file type it needs. When issuing the file request, the second byte of the header assists the server to decide which worker it should forward the request to.



Visualisation of the header

3.4 Sender Class

I have this Sender class to implement the Stop-and-Wait ARQ protocol using concurrency mechanisms. Since wait() method does not return anything, I cannot differentiate whether the wait() exits for being notified or reaching a timeout. Thus, I added a boolean variable that is set to True only when being notified. Below is my pseudo-code for Sender.

```
synchronized run() {
    while(true) {
        try {
            sendPacket(packet);
            wait(WAIT_PERIOD);
            if(sent) {
                return;
            } else {
                print("Packet_sending_failed._Will_try_again.");
            }
        } catch (Exception e) {e.printStackTrace();}
```

```

    }
}

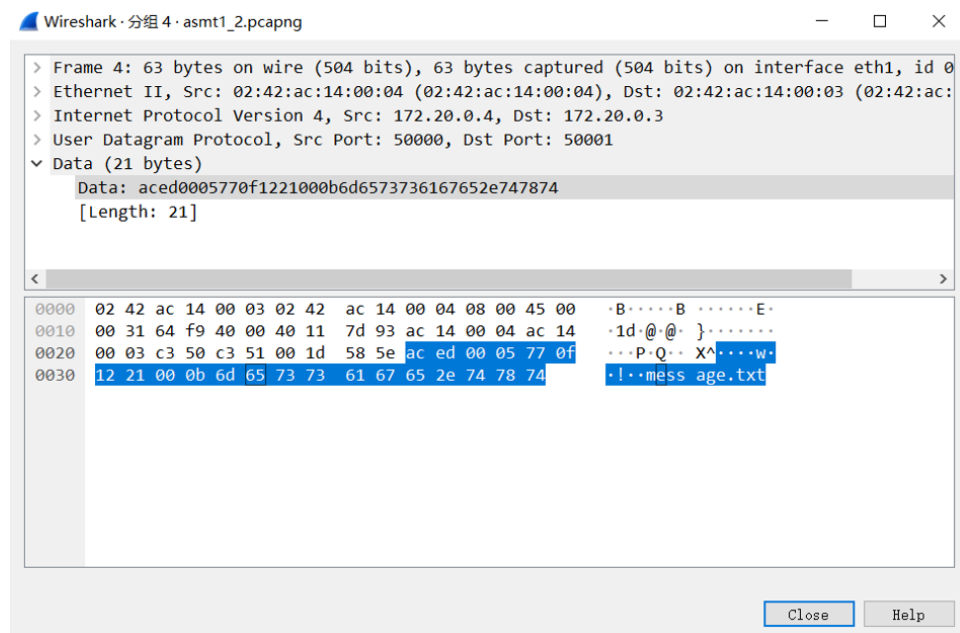
synchronized ackReceipt(){
    sent = true;
    notify();
}

```

Listing 1: Sender Class Pseudo-code

4 Discussion

I have noticed that my header bytes does not appear in the first two bytes but in the 7th and 8th bytes in my wireshark traffic for some reason. This has also appeared in the sample code. As this issue does not cause any problems in the actual program, I have elected to ignore it.



Wireshark traffic screenshot, note that there are six bytes before my header bytes

5 Summary

This report outlines my attempt to design a file retrieval system among three nodes. The description of the implementation in this report highlights the key elements of my solution and shows how the theory is put into practice.

6 Reflection

At the beginning of this assignment, I found that the provided sample code was quite helpful. It served as a good starting point. I had practically finished the basic functionalities during the first phase, so I took my time with the ARQ part. The Stop-and-Wait ARQ, however, was really challenging for me. While debugging, I continued running into issues, but I eventually found a solution that I am pleased with. This is all because of my progress during the first part, allowing me to spend more time and dig into some higher-level functionalities. The three phases of the assignment have given me an opportunity to achieve progresses

step by step during each phase. I therefore think it is crucial to divide tasks into parts and deal with a small portion each time.

I spent roughly 23 hours total on the assignment, not including the time it took to produce the report.