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

CS2031 Telecommunication II
Assignment #1: Protocols

Euan Leith, Std# 18323530

November 28, 2019

Contents

1 Introduction

The goal of this project is to create a system that transmits packets between nodes using Java.
Packets are sent from a server to a broker, which distributes these to workers that have registered. Workers can volunteer for and withdraw from work, as well as return results from any work they carried out to the broker; see figure 1.

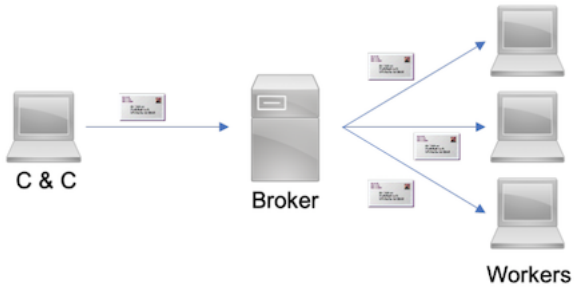

Figure 1: Shows the communication between a Server, a Broker, and a number of Workers.

2 Theory

This section will explain the basics of communication and error detection between nodes, as well as their implementation in Java with the Socket, DatagramPacket and Thread objects.

2.1 Communication

The goal of communication is to send information from one place to another. In telecommunications this is done by way of packets. Java has an object DatagramPacket which contains a byte array of data. These can be sent using the DatagramSocket class. These packets contain information about themselves, where they should go, and the data itself. The packets are sent from one node to another within the system, ultimately leading to the desired endpoint.

2.2 Sockets

In Java, a DatagramSocket can be created to act as a node in the system. This socket is given a destination port and a host name. For this project all of the nodes will be running on one computer, so "localhost" will be its host. The socket can now send and receive packets from other sockets.

2.3 Threads

A computer is able to run multiple threads, each of which are performing separate functions simultaneously. In Java this is done by creating class which extends Thread, with a function 'run' which will run when the thread starts.
As multiple nodes are being run on the same computer, different threads are required for each node. Each of these threads listens for incoming packets, then processes them.
This system also has a thread for each node which runs a constant timer that checks if the most recent frame for each port has been acknowledged before the timeout.
Complications can arise when a thread alters variables which are accessible to other threads, as two threads may want to alter a variable simultaneously. The resulting variable could be any combination of these processes. In Java this is dealt with by a number of features, namely in this case the keyword 'synchronised' which locks the function being run until it is completed. Using this on specific functions where these variables are being altered means that only one thread can change the variable at a time.

2.4 Error Detection and Correction

When communicating between nodes, packets can be lost. This loss has to be processed by the nodes, and the packet resent. There are many different standard protocols for doing this, but this project is only concerned with two; Go Back N, and Selective Repeat.

Go Back N:
Each frame is sent from the sender to the receiver. If a frame isn't lost, the receiver will send an acknowledgement (ACK) back to the sender. If the sender never receives an ACK, either because the frame was lost, or the ACK was lost, it has to resend that frame. However, the order of frames can be important. And so in order to retain order, the sender resends all of the frames from the lost frame onwards.

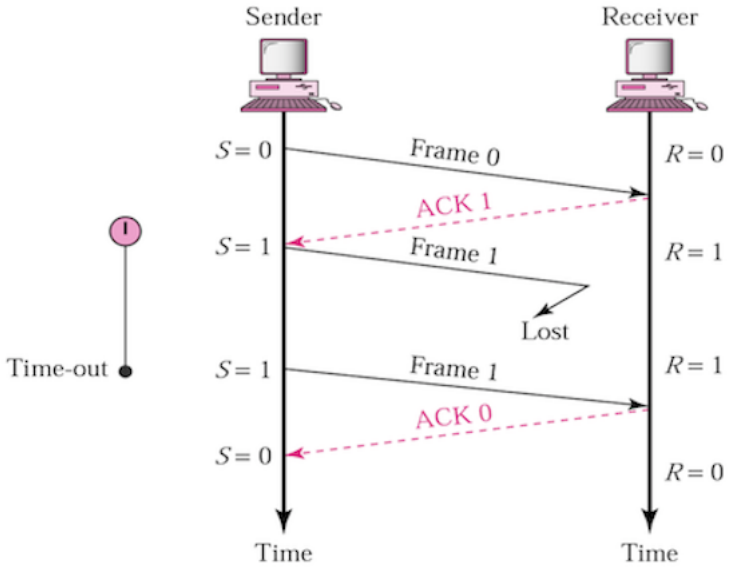

Figure 2: Go Back N Acknowledgement Protocol.

Selective Repeat:
Each frame is sent from the sender to the receiver. If a frame isn't lost, the receiver will send an ACK back to the sender. If the frame is lost, the sender will continue and send the next frame, but the receiver will send a negative acknowledgement (NAK) to say that it never received the previous frame. The sender then resends this lost frame. However unlike Go Back N, the subsequent frame doesn't need to be resent as the sender can be sure that it wasn't lost since, since the receiver responded to that frame.

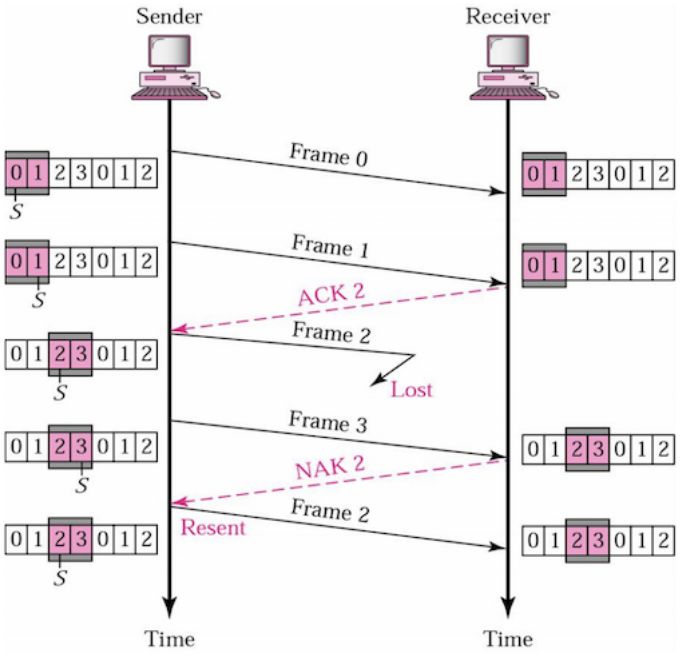

Figure 3: Selective Repeat Acknowledgement Protocol.

3 Implemenation

3.1 Start-up

When a node is run the listener and timer are initialised; see listing 1. Then a socket is created at a given port. A listener thread is then run, which waits to receive packets. A timer thread is also run which will deal with acknowledgements timing out; see listing 2 The Server and Worker will add the Broker's port to their timer. The Broker won't add ports to its timer until a Worker volunteers.

```
Node() {
        latch = new CountDownLatch(1);
        listener = new Listener();
        listener.setDaemon(true);
        listener.start();

        clock = Clock.systemDefaultZone();
        queue = new ConcurrentHashMap<>();
        timer = new AckTimer();
        timer.start();
}
```

Listing 1: Standard constructor for a Node, which initialises the listener and timer threads.

```
Server() {
        timer.addPort(DEFAULT_DST_PORT);
        try {
            socket= new DatagramSocket(DEFAULT_SRC_PORT);
            listener.go();
        } catch(java.lang.Exception e) {e.printStackTrace();}
}
```

Listing 2: Additional constructor for each subclass of Node, in this case Server. This creates a socket and starts the listener, and adds the Broker's port to the timer.

3.2 Listening for Packets

Each node has a class Listener which runs an endless loop waiting for any packets to arrive. When a packet is received, it creates an object for it and runs the function onReceipt, which will process the packet according to its contents. Each Listener is a thread so that multiple can be run on the same computer.

```
// Listen for incoming packets and inform receivers
public void run() {
      try {
         latch.await();
         // Endless loop: attempt to receive packet
         while(true) {
            DatagramPacket packet = new DatagramPacket(
               new byte[PACKET_SIZE], PACKET_SIZE);
            socket.receive(packet);

            onReceipt(packet);
          }
      } catch (SocketException e) {
      } catch (Exception e) {e.printStackTrace();}
}
```

Listing 3: Listener class waits for packets to be received, then processes them.

3.3 Packet Encoding

Packets are split into two types; FileFuncContent (FFC) and AckPacketContent (APC). FFC packets contain which port(s) to forward itself to (if any), a file, and what to do with it. APC packets contain an acknowledgement message, and are returned whenever a non-acknowledgement packet is received.
The attributes of each packet are assigned using the Java class ObjectOutputStream; see listing 4, and read using ObjectInputStream; see listing 5. These streams are variables which contain an ordered array of items of various Java types.

```
FileFuncContent(ObjectInputStream oin) {
        try {
            type= FILEFUNC;
            filename = oin.readUTF();
            size = oin.readInt();
            ports = oin.readUTF();
            func = oin.readUTF();
        } catch(Exception e) {e.printStackTrace();}
}
```

Listing 4: Constructor which reads and assigns the classes attributes from an Object-InputStream, in this case for the FileFuncContent class.

```
protected void toObjectOutputStream(ObjectOutputStream oout) {
        try {
            oout.writeUTF(filename);
            oout.writeInt(size);
            oout.writeUTF(ports);
```

```
            oout.writeUTF(func);
        } catch(Exception e) {e.printStackTrace();}
}
```

Listing 5: Function for implementation of PacketContent, in this case FileFuncContent, which writes its attributes to an ObjectOutputStream.

These are then converted to and from byte arrays, which can then be sent and received as packets; see listing 6.

```
public DatagramPacket toDatagramPacket() {
        DatagramPacket packet= null;

        try {
            ByteArrayOutputStream bout;
            ObjectOutputStream oout;
            byte[] data;

            bout= new ByteArrayOutputStream();
            oout= new ObjectOutputStream(bout);

            oout.writeInt(type);          // write type to stream
            toObjectOutputStream(oout);   // write content to stream by type

            oout.flush();
            data= bout.toByteArray(); // convert content to byte array

            // create packet from byte array
            packet= new DatagramPacket(data, data.length);

            oout.close();
            bout.close();
        } catch(Exception e) {e.printStackTrace();}

        return packet;
}
```

Listing 6: Function which writes a PacketContent's attributes to an ObjectOutputStream, which is then converted to a byte array. A packet is then created from this byte array.

3.4 Sending & Receiving Packets

Once a packet is received, it goes through a number of switch statements to determine what to do with it; see listing 7.

```
void onReceipt(DatagramPacket packet) {
        PacketContent content = packet.toDatagramPacket();
        switch(content.type) {
                case FILEFUNC:
                        switch (content.func) {
                                //do stuff
                        }
                case ACKPACKET:
                        //do stuff
                default:
                        //invalid type
```

```
        }
}
```

Listing 7: Function onReceipt is run when a packet is received and performs the appropriate action given the contents of the packet.

In this system, the nodes receive packets from a User node, but normally each node would perform its own action. As a result sending these packets is done in the onReceipt function.
The Worker can send packets to the Broker to volunteer and withdraw from work, and well as return results for assignments. It can also receive work assignments from the Broker.
The Broker receives volunteer and withdraw requests from Workers, as well as results for assignments; see figure 4. It can also receive assignments from the Server, and distributes them to the given workers who have volunteered; see figure 5.

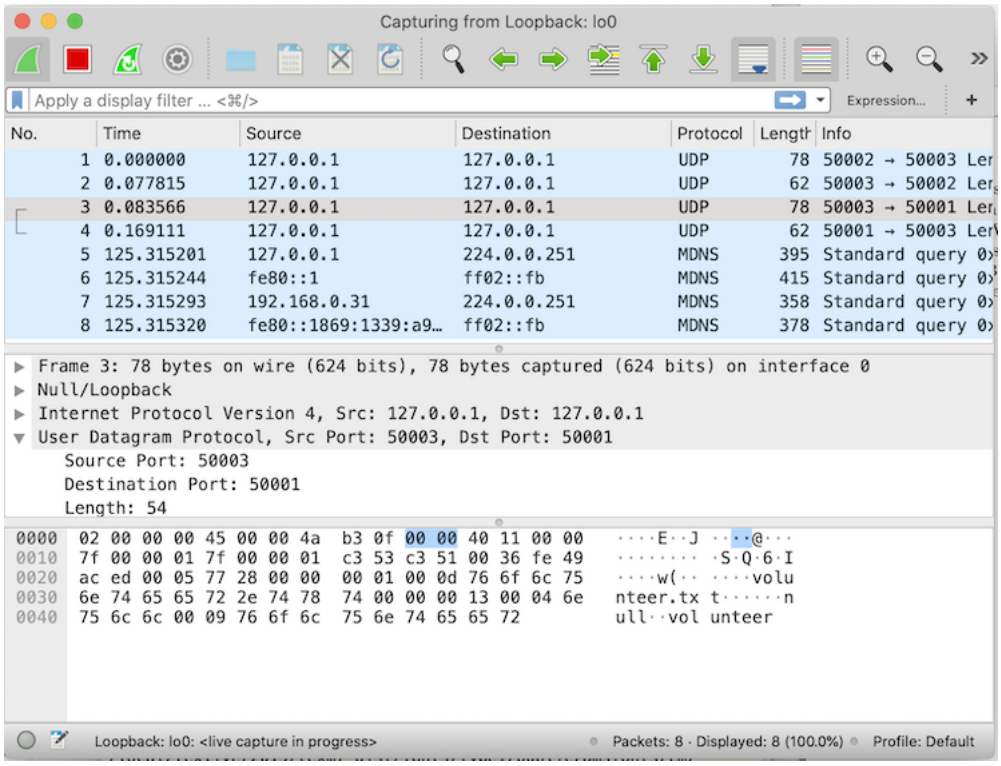

Figure 4: Broker at port 50001 receives a packet from a Worker at port 50003. This contains the file 'volunteer.txt', the function 'volunteer', and the ports to forward to 'null'. This would be similar for the Worker withdrawing and returning results.

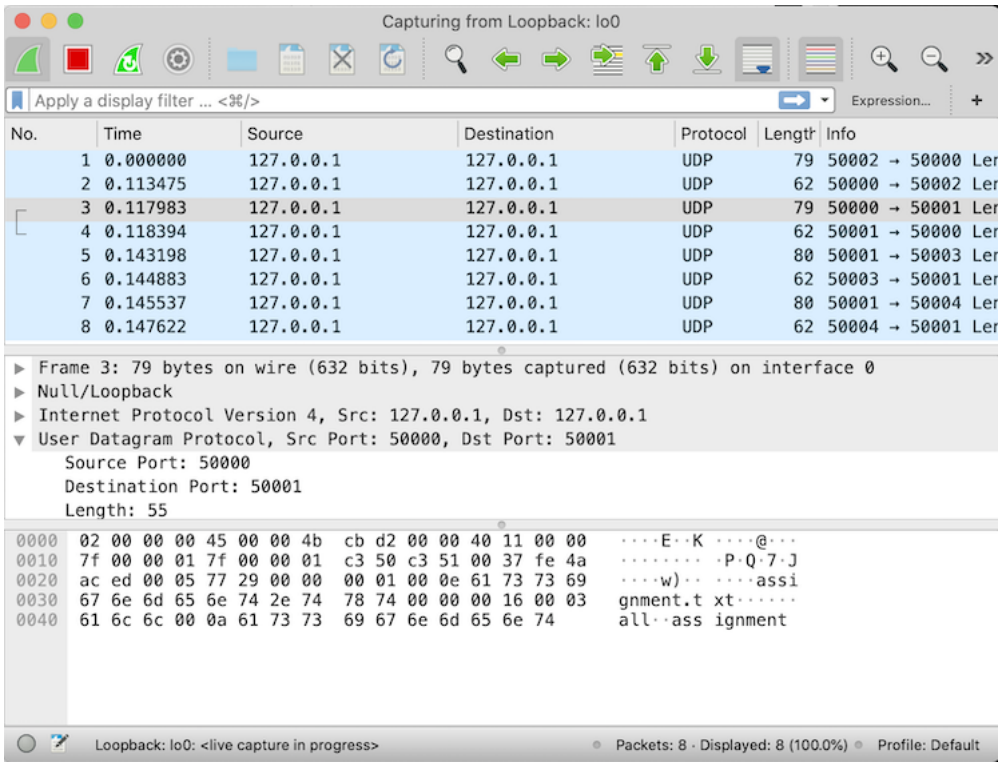

Figure 5: Broker at port 50001 receives a packet from the Server at port 50000. This contains the file 'assignment.txt', the function 'assignment', and the ports to forward to 'all', meaning all the Workers that have volunteered.

The Server sends work assignments to the Broker, containing information about which workers to forward the packet to.

3.5 Error Detection and Correction

Each node returns acknowledgements for any frames it receives; see figure 6.

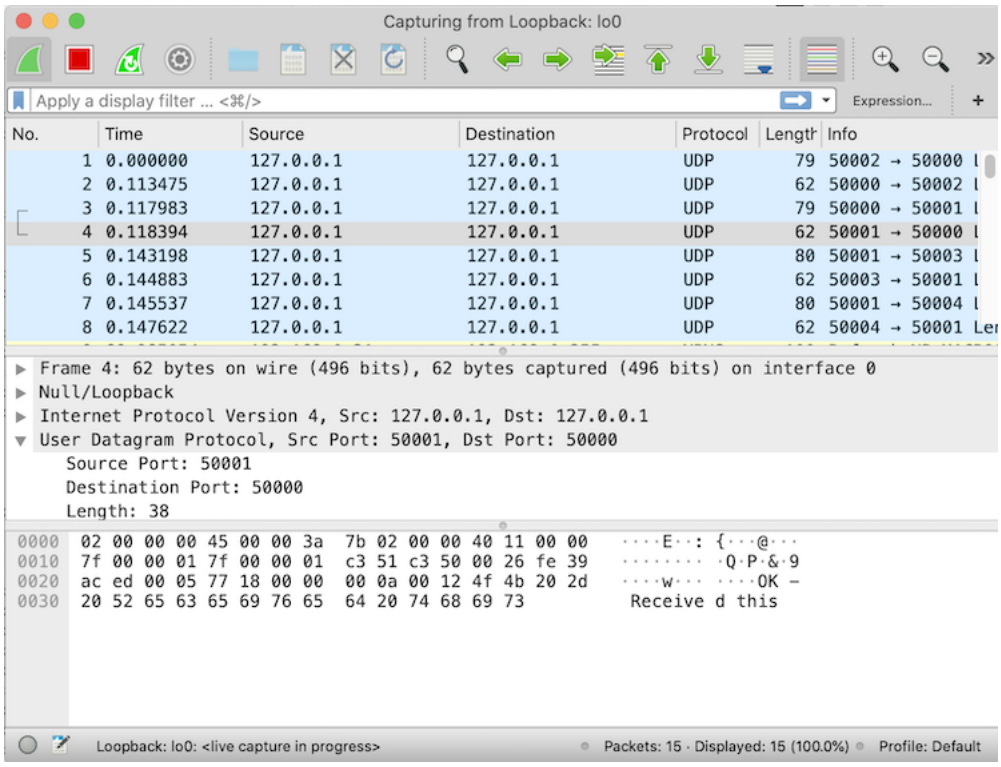

Figure 6: Broker at port 50001 returns an acknowledgement the Server at port 50000 with the message "Ok - Received this".

Each node has a list of frames sent and which port they were sent to that have yet to be acknowledged. This is done with a ConcurrentHashMap which maps a port to a frame; see listing 8. This is done so that if a frame has to be resent, only the subsequent frames for that frame's port are resent. This list is concurrent as its altered by multiple threads, so that altering it in one frame will update the value in the others.

```
private ConcurrentHashMap<Integer, Queue<PacketTimer>> queue;

private class PacketTimer{
        PacketContent content;
        Instant timeSent;
}
```

Listing 8: Each node contains a list of frames to be acknowledged. This is done with a ConcurrentHashMap which maps a port to an list of PacketTimer's, each of which has a frame and the time it was sent.

Whenever a frame is received, it is added to this list. If the frame is acknowledged, it is removed from the list. If the frame doesn't receive an acknowledgement before the timeout, all of the frames in the list for that frame's port are resent. This is done by a thread with an endless loop which checks if the most recent frame has reached the timeout; see figure 9.

```
public void run() {
    // Endless loop: checking packets have received
    // acknowledgements by a given time
    while (true) {
        for (int port : queue.keySet()) {
            queue.computeIfPresent(port, (k, packets) -> {
                if (!packets.isEmpty() &&
                        clock.instant().getNano() -
```

```
                        packets.peek().timeSent.getNano() <
                        WAIT_TIME) { // reached timeout
                    System.out.println("No ack received for "
                        + packets.peek().content.toString());
                    for (PacketTimer packet : packets) { // resend packets
                        onReceipt(packet.content.toDatagramPacket());
                        packets.remove();
                    }
                }
                return packets;
            });
        }
    }
}
```

Listing 9: Function with an endless loop which checks if the most recent frame has reached the timeout. If so, it resends that frame and all subsequent frames for that frame's port.

Only the most recent frame for each port has to be checked with this method, and only one timer thread is required.
Each loop runs a synchronised function so that if a frame is added or removed from the list in the node's thread, it is dealt with after the loop so as to prevent overlapping. The add and remove functions must also be synchronised for this same reason.

4 Choices and Advantages & Disadvantages

4.1 Packet Types

This system has separate Java objects for each different type of packet, however the contents of the packets could be assigned without these.

Advantages:

- Easy to understand: Using classes hides some of the details of creating packets, whereas not doing so requires the programmer to implement those details manually each time the packet's content is parsed.

Disadvantages:

- Over-complication: Using classes for simple things such as converting some Java objects to a byte array might make the process seem more complicated than it actually is, causing any programmers that are trying to understand the system to take longer to understand what is going on.
- Efficiency: Using classes means more jumps in the code, which are less efficient than not doing so. This is because the processor processes more than just one line at a time, but jumping to another section of code means that the information for the subsequent lines that were partially processed must be discarded.

4.2 Acknowledgement Protocol

This system uses the Go Back N acknowledgement protocol, which will be compared to the Selective Repeat protocol:

Advantages;

- Frames received in sequence: The Go Back N protocol resends the frames from the lost frame onwards in order, therefore there the receiver will always receive the frames in order. However, with Selective

Repeat one only resends the frame that was lost. Therefore it will can be received after other frames which were originally sent before it. So the packet must also contain its position in the order so that the receiver can parse this information.

- Cumulative acknowledgements: With Go Back N the receiver can assume that if it receives an acknowledgement for a frame, all the previous frames have also been received. Therefore acknowledgements don't need to be sent for every frame. However using Selective Repeat requires each frame to be acknowledged individually.

Disadvantages;

- More retransmissions: With Selective Repeat only the frame that was lost must be resent. However with Go Back N, all the subsequent frames must also be resent. This can waste time as some of these frames may have already been received by the receiver.

4.3 Timeouts

This system has a constant timer which checks if the most recent frame for each port has been acknowledged before the timeout, as opposed to creating a new timer for each frame that is sent.

Advantages:

- Less complex: This system only has to deal with other threads adding or removing frames from its list of frames. However the other approach has to deal with all of the other timer threads if a frame isn't acknowledged, as the most recent frame's thread must stop all other threads before resending the frames.
- Fewer threads: This system requires only one thread which checks the most recently sent packets, whereas the other approach creates a new timer thread for each frame sent, using more computing power.

Disadvantages:

- Not always in use: In this system the timer is constantly running, but frames are not always being processed. Therefore there is time when the thread is running but doing nothing. However using the other approach would mean that threads only run when a frame is being processed.

5 Summary

This report has presented my attempt at a system which allows for communication between nodes. It highlights the essential components of my solution and demonstrates the execution of the solution.

6 Reflection

I always find that working through ideas by actually implementing them myself useful, and this assignment was no exception. This always me to work out the various kinks of implementation, as well as iron out any issues or questions I might have regarding the topic. In particular I found that I learned a lot about multithreading and the problems that can arise. This assignment took me about 16 hours to complete.