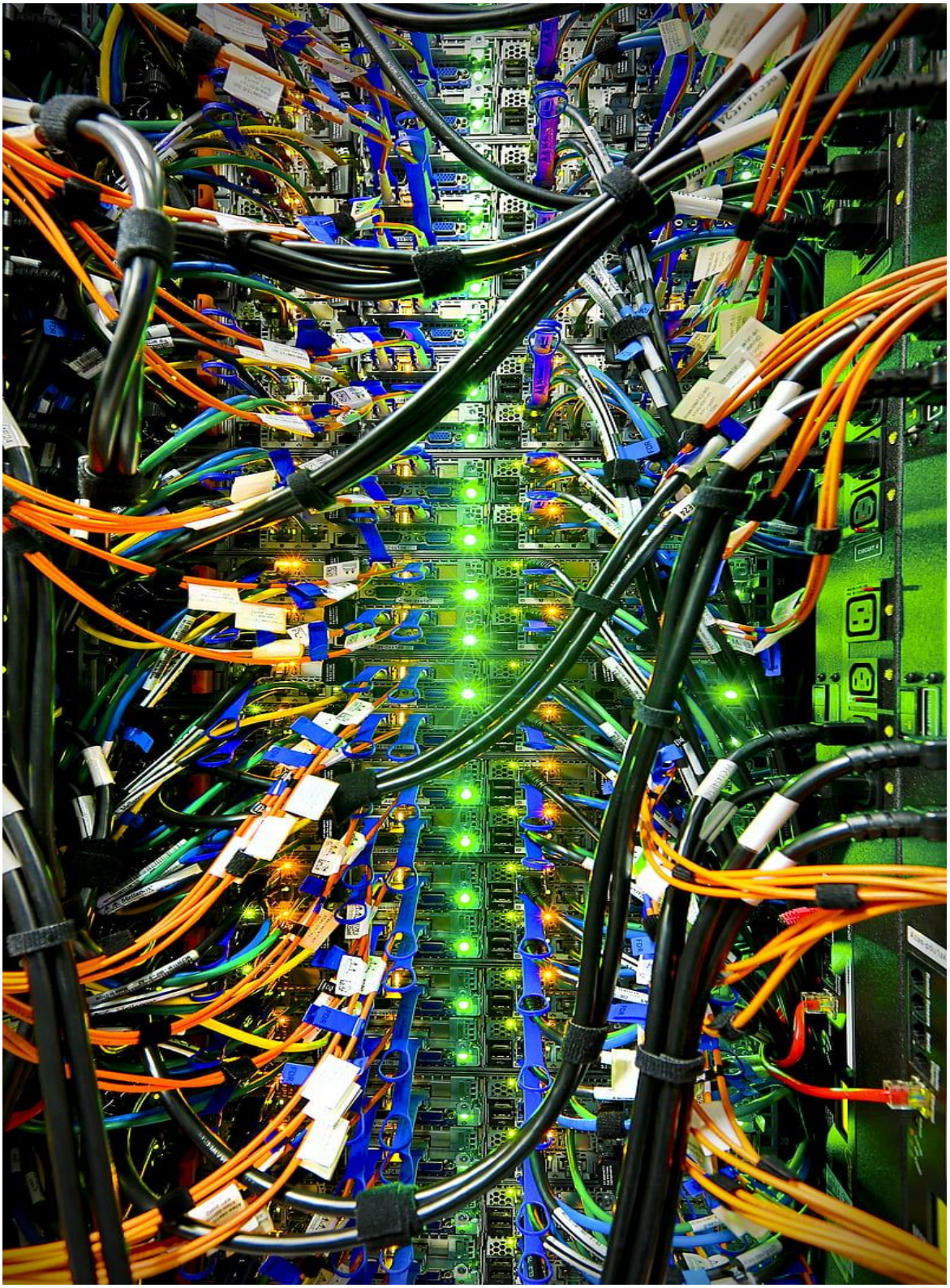
CSU 33031 ASSIGNMENT 2 REPORT



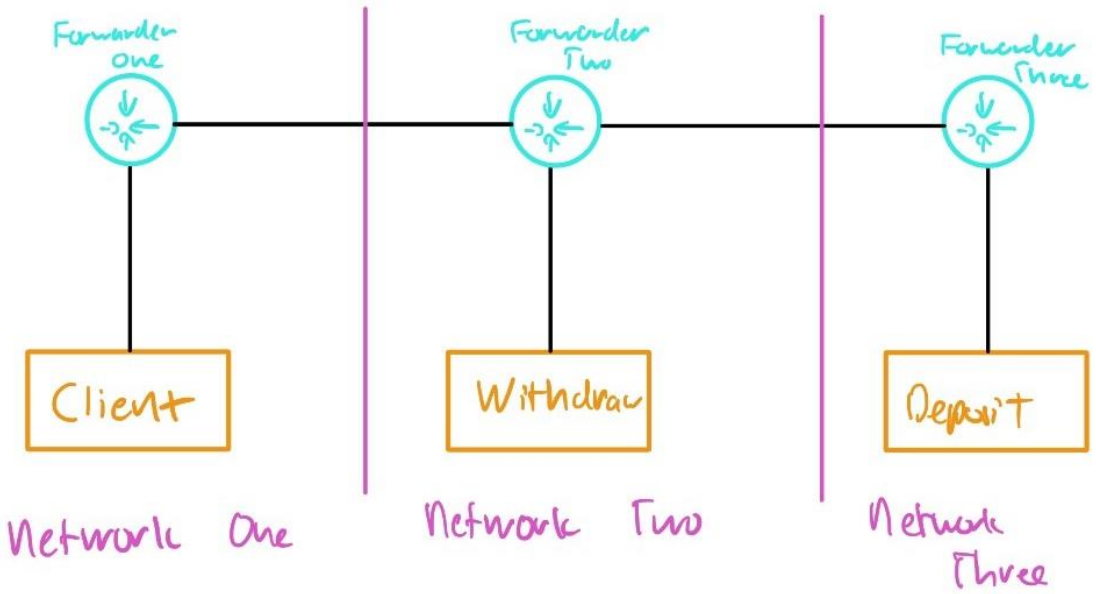
Elliot Lyons, Student ID: 20333366

Contents

Initial Approach

Starting this assignment was a lot easier than the last, due to the knowledge I had garnered. A decision I initially had to make was whether to build on my code from that assignment or to start afresh. I decided to do the latter as I wasn't sure how compatible my previous code for communicating across multiple networks was.

My initial approach was to simply establish inter-network UDP communication between nodes. See below my original flow diagram:

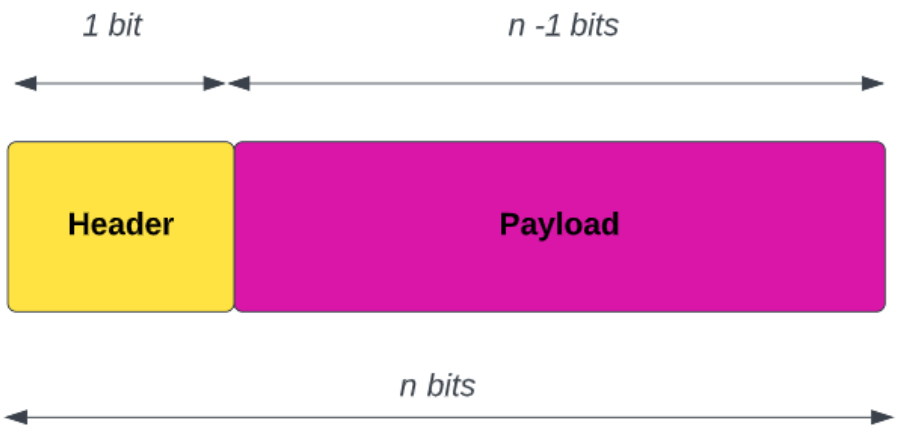


(Figure 1: Initial Approach Flow Diagram)

***Note**: there is an element of artistic license here. 'ForwarderOne' is also in Network Two, 'ForwarderTwo' is in Network One, Two and Three and 'ForwarderThree' is in Network Two and Three. This enables the forwarders to communicate with each other. The same is the case in Figure 3.*

I decided to base my code on a simple banking system, similar to the assignment brief. A client would input into the system whether they wanted to withdraw or deposit money to their account. Depending on their choice, the request would be sent to the relevant node to carry out this operation. A packet would also be sent to the other operator, to tell them their services were not required.

For the time being, the port numbers and packet destinations were hard coded into each forwarder. There was no controller nor forwarding table at this stage. The forwarders would know where to send packets based on the header value in the data.

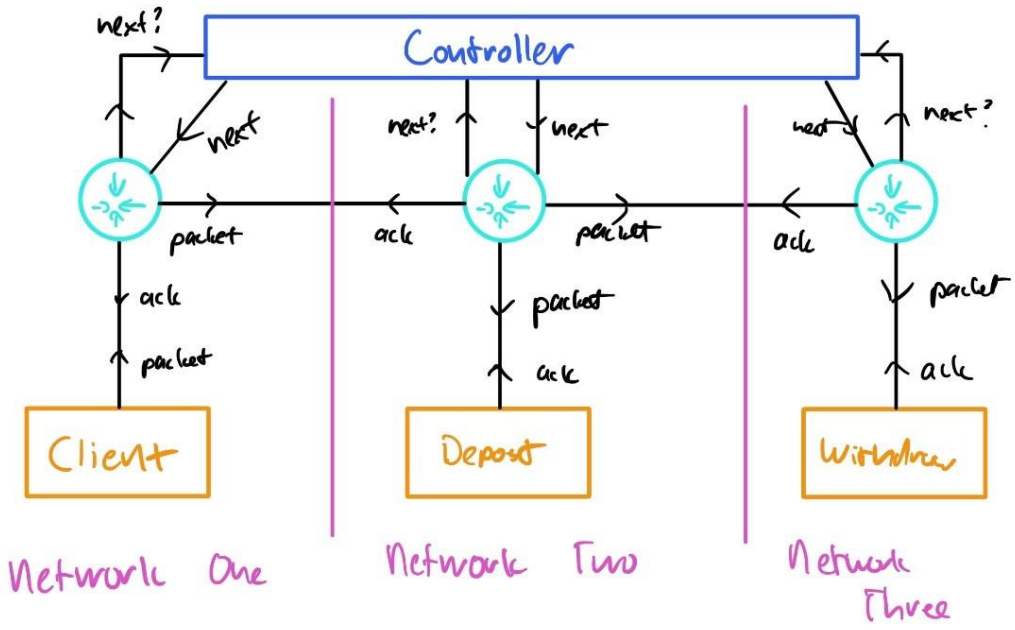


(Figure 2: Packet Structure)

Initially if the header value told 'ForwarderOne' a withdrawal was wanted, the forwarder would send that packet to 'ForwarderTwo'. 'ForwarderTwo' would then send that request to the withdrawal node where it would carry out that action. 'ForwarderThree' would then receive information that the deposit services were not wanted and terminate. At this point there was no acknowledgements being sent between nodes; the client wouldn't actually know if their request had been successful. There was some error checking implemented in the UI of the application. This meant that the user would not be able to input a false request, such as withdrawal of funds greater than the account balance. My next step was to implement the sending of acknowledgements and a controller which would store a forwarding table that the forwarders would query when they needed to forward a packet, simulating a Software-Defined Wide-Area Network (SD-WAN).

Progression of Approach

My next step was to add a controller and acknowledgements to the system. See below the flow diagram of this being put into practice:



(Figure 3: Flow diagram for furthering approach)

Adding a Controller

As you can see, the forwarders no longer have the next destination hard coded. They instead each query the controller where their next hop is. The controller has the forwarding table for the entire system and will know where a packet needs to go. The packets have the same structure as in Figure 2. This means the controller can identify where a packet needs to go based off the header value of that packet. It can then relay the relevant address of the next hop back to the forwarder. The forwarder will then send it on to the next forwarder or to the service in their network. See below the terminal window for such an example and the code to achieve this.

```
Packet received.
Sending to control
Received from control
Next node: ClientTwo
Next port: 3
Forwarding packet.
```

(Figure 4: 'ForwarderTwo' receiving a packet)

```java
xPacket.setSocketAddress(toController);
System.out.println(x: "Sending to control");
controller.send(xPacket);

byte[] fromControl = new byte[1024];
DatagramPacket yPacket = new DatagramPacket(fromControl, fromControl.length);
controller.receive(yPacket);
System.out.println(x: "Received from control");
controller.close();

byte[] received = yPacket.getData();

byte[] r = new byte[received.length - 1];

for (int i = 0; i < r.length; i++)
{
    r[i] = received[i+1];
}

int nextPort = received[0];         // next port number is received in the header of the file

String nextNode = new String(r);    // next node is detailed in the body of the packet
System.out.println("Next node: " + nextNode);
System.out.println("Next port: " + nextPort);
```

(Figure 5: Forwarder.java receiving a packet from controller to see where to go next)

'ForwarderTwo' receives a packet from 'ForwarderOne'. It sends a query to control as to where it needs to go next. As you can see the next step is 'ClientTwo'. The controller will then pass on the port for that node. 'ForwarderTwo' then forwards the packet to that address.

I added in system timeouts for nodes throughout the network. If a deposit was wanted the withdrawal service would timeout instead of listening for a packet that would never come.

Adding Acknowledgements

The addition of node timeouts meant I had no choice but to implement acknowledgements being sent. This was not a huge issue as you can simply send a response back on a socket once a connection has been established in Java UDP programming. The acknowledgement would also include whether an action had been done or not. For example, the withdrawal node would send an acknowledgement message that the withdrawal had occurred. It would include the updated bank balance within that message. This acknowledgement would be sent back to all nodes that forwarded the request to the destination. This means that the client would be told that their request had been successful and be updated with their new bank balance. See below for an example:

```
Packet received.
Sending to control
Received from control
Next node: ClientTwo
Next port: 3
Forwarding packet.
$200 deposited. New balance: $10200.
Sending back to client.
Program completed.
```

(Figure 6: 'ForwarderTwo' receiving and sending acknowledgements)

```java
int bal = 10000 + value;

String response = "$" + value + " deposited. New balance: $" + bal + ".";
System.out.println(response + " Responding to client.");

byte[] toClient = response.getBytes();
DatagramPacket toC = new DatagramPacket(toClient, toClient.length);     // sending ack back to forwarder
toC.setSocketAddress(packet.getSocketAddress());
socket.send(toC);
socket.close();
```

(Figure 7: ClientTwo.java completing request and sending an acknowledgement)

Figure 7 describes how 'ClientTwo' sends an acknowledgement back to 'ForwarderTwo'. Figure 6 builds upon Figure 4. You can see 'ForwarderTwo' receives the acknowledgement saying that $200 has been deposited from 'ClientTwo'. It then will send an acknowledgement to the node that sent it the original request. Eventually the acknowledgement will be sent to the client as below.

```
Account balance = $10000.
Press 1 to withdraw funds or 2 to add funds to your account.
2
Input amount to add or withdraw.
200
Packet sent.
Response received: $200 deposited. New balance: $10200.
```

(Figure 8: Client sending original request and receiving acknowledgement)

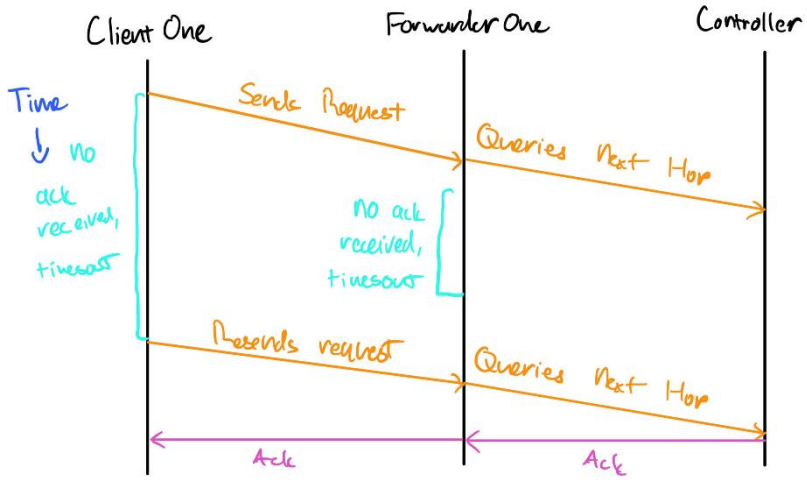
Stop-and-Wait ARQ

I decided to implement the stop-and-wait ARQ (Automatic Repeat Request) as I was satisfied with the utility and reliability it offered in the first assignment. If an acknowledgement wasn't received in a certain period, a node would timeout. The user would be alerted that there was a timeout within the system and would instruct them to restart the system to resend their request as below:

```
Account balance = $10000.
Press 1 to withdraw funds or 2 to add funds to your account.
1
Input amount to add or withdraw.
100
Packet sent.
Timeouts occurred. Restart application.
```

(Figure 9: Stop-and-wait ARQ implementation)

The flow diagram for such a sequence is as below:



(Figure 10: Stop-and-wait ARQ flow diagram)

Concluding Thoughts

Overall, I enjoyed doing this assignment. I successfully implemented a multi-network system that dealt with forwarding queries through the use of a forwarding table as per the assignment brief. I believe my system does so in a sophisticated manner as well which I am proud of.

I'm also happy with my progress from my first assignment. I was complimented on the use of my ARQ so it made sense to implement it again for this assignment. I was recommended to include acknowledgements as well as having a sender timeout in my first assignment. Unfortunately, I didn't manage to do either for the final submission of said assignment so I am glad I got to both for this one.

My knowledge of computer networks, starting this report, was far greater than when I started the previous one. Of course, my knowledge was gained primarily from doing assignment one but I found it a lot simpler to start this time around. This meant I was able to implement the above features easier and I believe my knowledge of forwarders and routers is now far greater than it was when I set out to build this system.

Cover image source: https://www.peakpx.com/en/hd-wallpaper-desktop-kvaaz

Source code may also be found at: https://github.com/elliot-lyons/comp-networks-assmt-2