Telecoms II - Assignment II Report

Open Flow

Samuel Petit

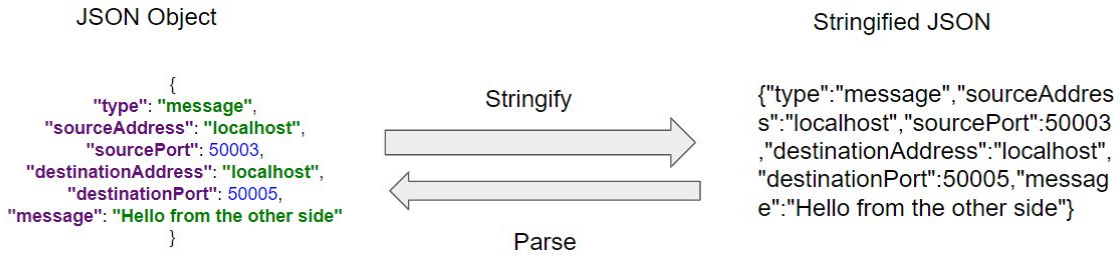
petits@tcd.ie - 17333946

Contents

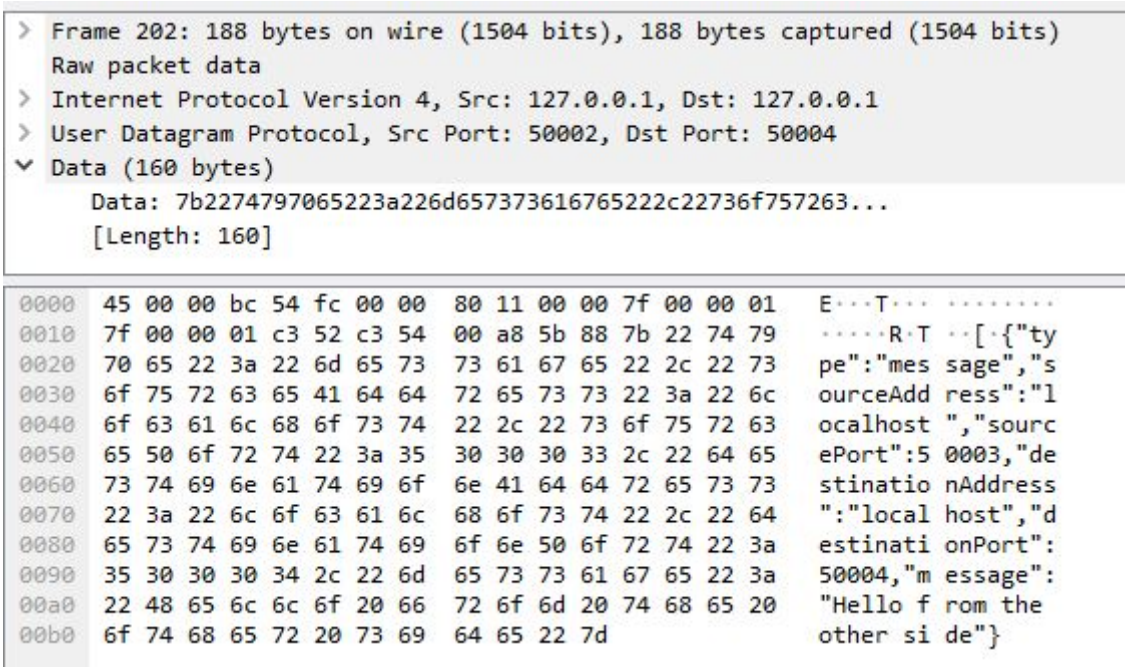
1. The core of my program
    a. Packet structure
    b. Ports
    c. Program Structure
    d. Controller
    e. Router
    f. Client
    g. Messaging Flow
2. Additional Material and personal opinion
    a. Advantages & Disadvantages
    b. Personal Opinion and design overview

1. The core of my program
    a. Packet structure

The way I structure my packets is by using JSON. This makes communication very easy as I can create a JSON object and set a specific type depending on the kind of packet to send, the receiver can then parse the JSON string received into a JSON object and access the different elements very easily, and make decisions based on the “type”.



This little diagram shows how we can stringify a JSON Object into text and then parse it back. It is a sample packet than might be sent when a client wants to send a message to another client. Here is that very same packet captured with wireshark



:

    b. Ports

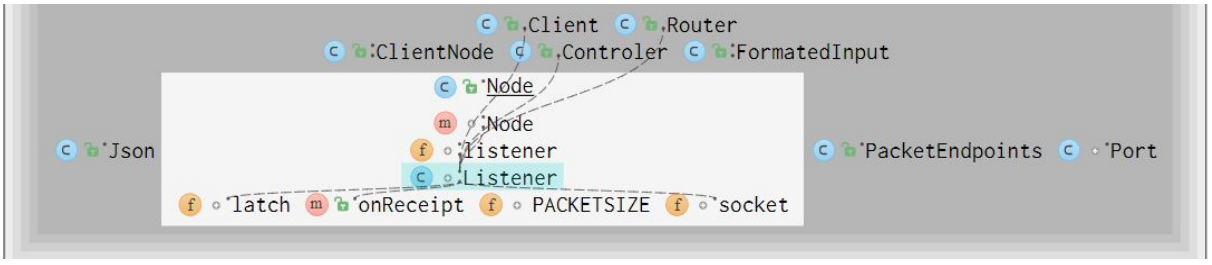
The controller is the only system which has a set port : 50001. Every single other client will ask for a port in the command line when launching it before starting. This means that every separate router or client will require a separate port for communication, my program then asks for a port to send a message to when a client has decided to send a message. To make sure this is somewhat controller, my program will only accept **valid** ports. For instance if the program is asking for a port to use to launch a new router instance, my program will make

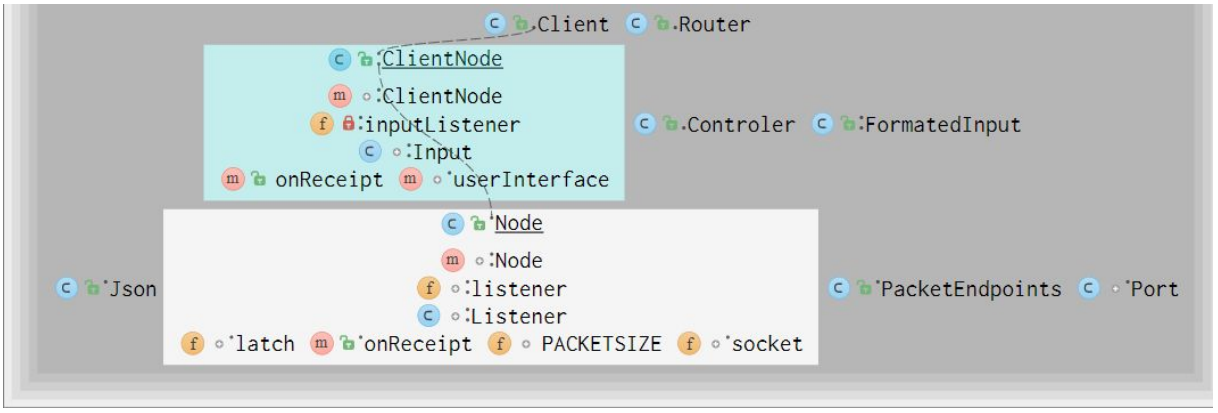
sure that the user input is a valid integer and that the port is not currently in use. When asked for a port to send a message to it will also only accept valid integers as well as

c. Program Structure

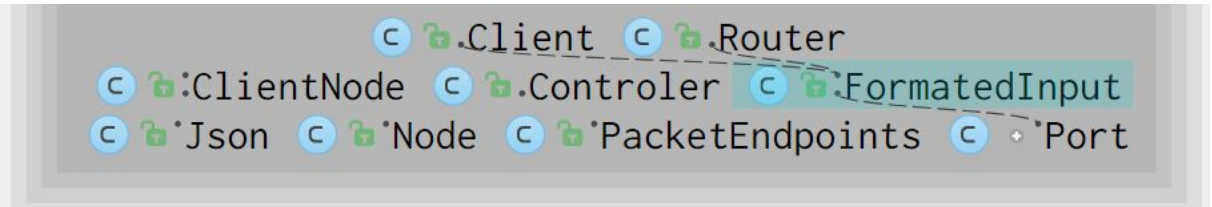
There are a few different types of classes in my assignment, here's a quick high-level run through illustrated with a few screenshots of the architecture of my program, taken with structure101 :

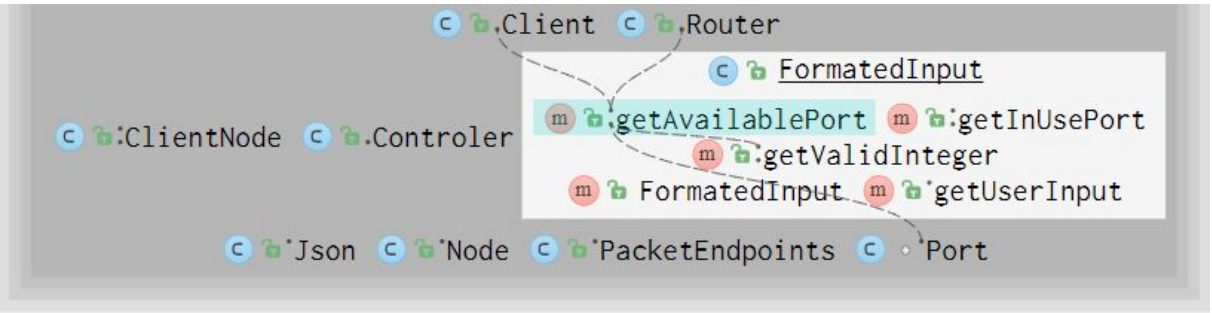
- Main classes : Client, Controller, Router. These are the classes to run when testing the program and they represent what all of the other classes are based on.
- Subclasses : There are 2 classes that the 3 main classes mentioned above extend. Node and ClientNode, the router and controller class both extend Node and the Client class extends the ClientNode class. That ClientNode class actually extends the Node and adds the ability to keep reading user input while the other 2 don't need this feature. The node class then is used as it : extends thread, making it possible to run multiple instances concurrently, has methods used to receive packets.



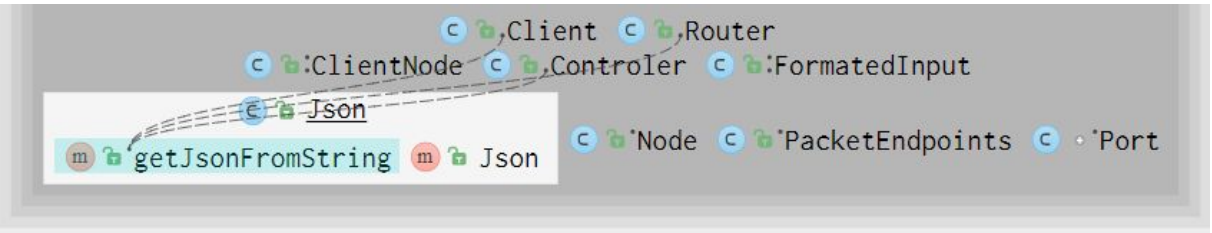


- Utilitary classes : FormatedInput, Json, Port are all classes that only contain static methods as these methods are used by all 3 of my main classes. Here is a quick overview of what features these classes have :
  - Port only has one method and it is pretty straight forward, it will get an integer value and check if that port is currently used or not. It will return a boolean value.





- FormatedInput revolves around user input in the command line. Its methods include getting user input (any string), getting an integer from the user, getting an available port and finally a used port (by using the method from the Port class above).
- Json : this class also contains a single method which will parse a json formatted string into a Json Object.



- The last class I didn't cover is PacketEndpoints, this class is only used in Hashmaps that are used when forwarding packets. I will dive deeper into this class in the coming paragraphs.

d. Controller

The controller as you might've guessed manages how packets are sent from one endpoint to another through multiple routers. It's only role is to acknowledge new router connections and create paths upon requests. Essentially it keeps a list (hashset) with the ports of the routers that have requested connections. Upon the receipt of a "path request" packet this is how my algorithm handles the creation of new routes.

The JSON string received when a router is requesting a new path might look like this :

```
{
  "type": "unknown packet path",
  "sourceAddress": "localhost",
  "sourcePort": 50003,
  "destinationAddress": "localhost",
  "destinationPort": 50004
}
```

The information about source and destination of the actual packet is used to send

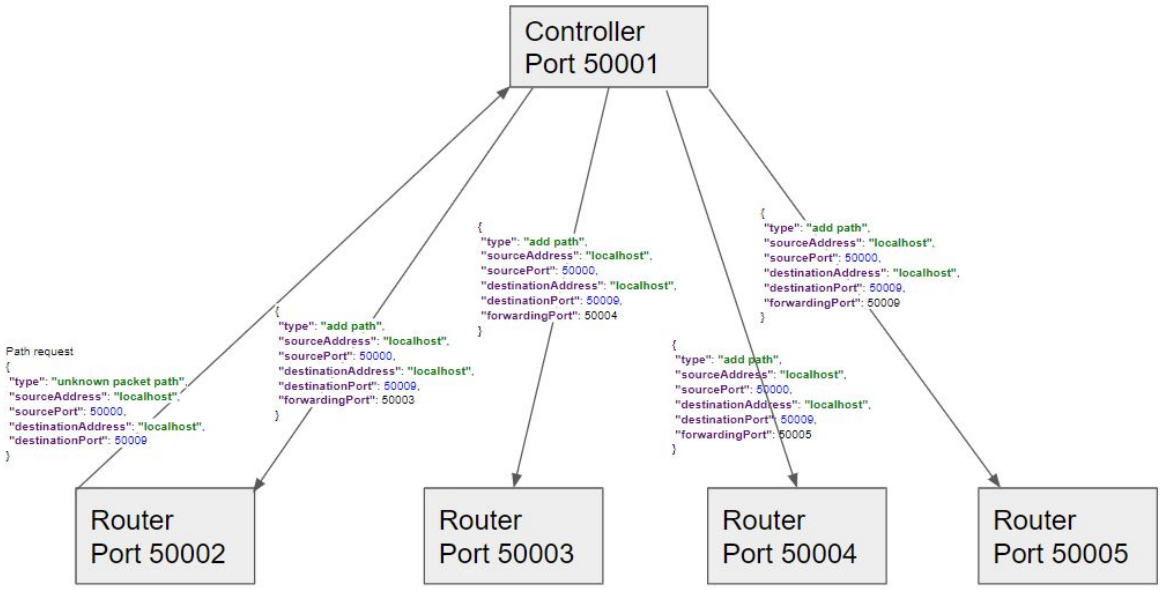
forwarding information to the routers. Here is the breakdown of what happens :

1. The controller receives a packet
2. The packet is converted into a JSON Object
3. The type of packet is checked and the correct method is called
4. The controller starts by considering the origin of the packet (that is the port of the router that requested a path for a packet).
5. If the controller doesn't have any other routers connected, a packet will be sent to the router that sent the request packet. Here is what it might look like (simplified an captured on wireshark) :


```
{
  "type": "add path",
  "sourceAddress": "localhost",
  "sourcePort": 50003,
  "destinationAddress": "localhost",
  "destinationPort": 50004,
  "forwardingPort": 50004
}
```


   The source and destination data is sent as this information is used by the routers - again, more on this later. The port that the specific router will then send this packet to is also added at the very end, in this case we can see that only 1 router will be used in this communication and that the router will forward the packet directly to the destination endpoint.

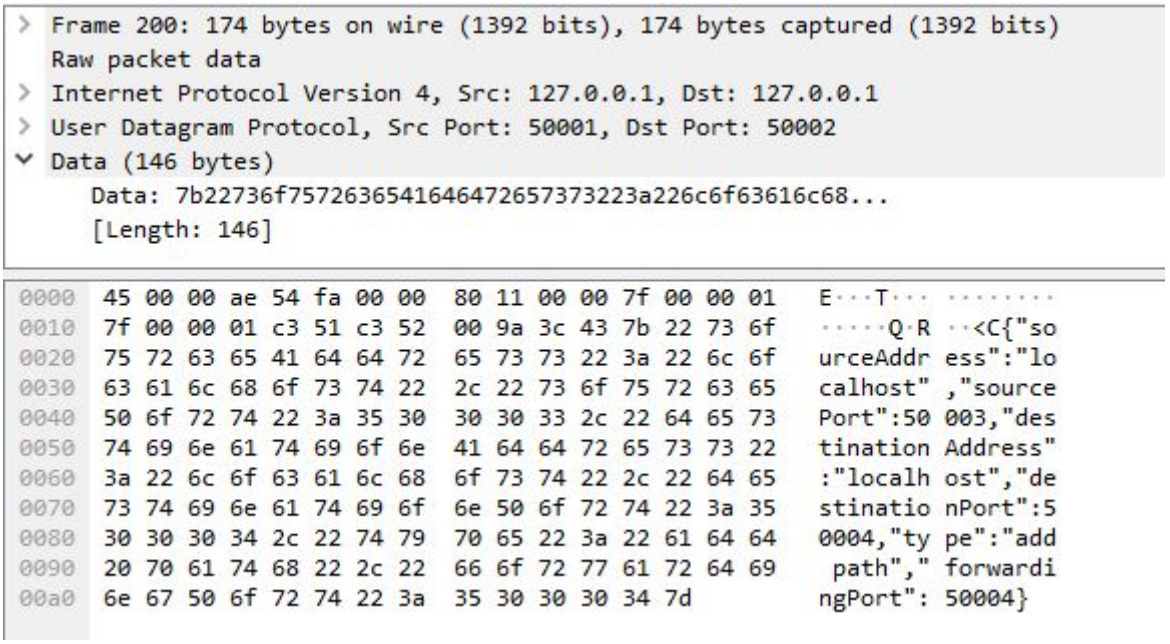
6. If there are more routers connected, then similar packets will be sent to every single router. Linking the first one to the next one, until it gets to the last port, and send the actual endpoint as the "forwardingPort". Here is a diagram of how this may look like:



Visualisation of a controller assigning paths to routers (acknowledgement packets ignored)

As we can see here, the controller is essentially linking together all of the routers until it assigns the last router to the actual endpoint of the packet.

Here is one of those packets, captured on wireshark :



We've covered everything the controller does, let's now move on to my router class.

e. Router

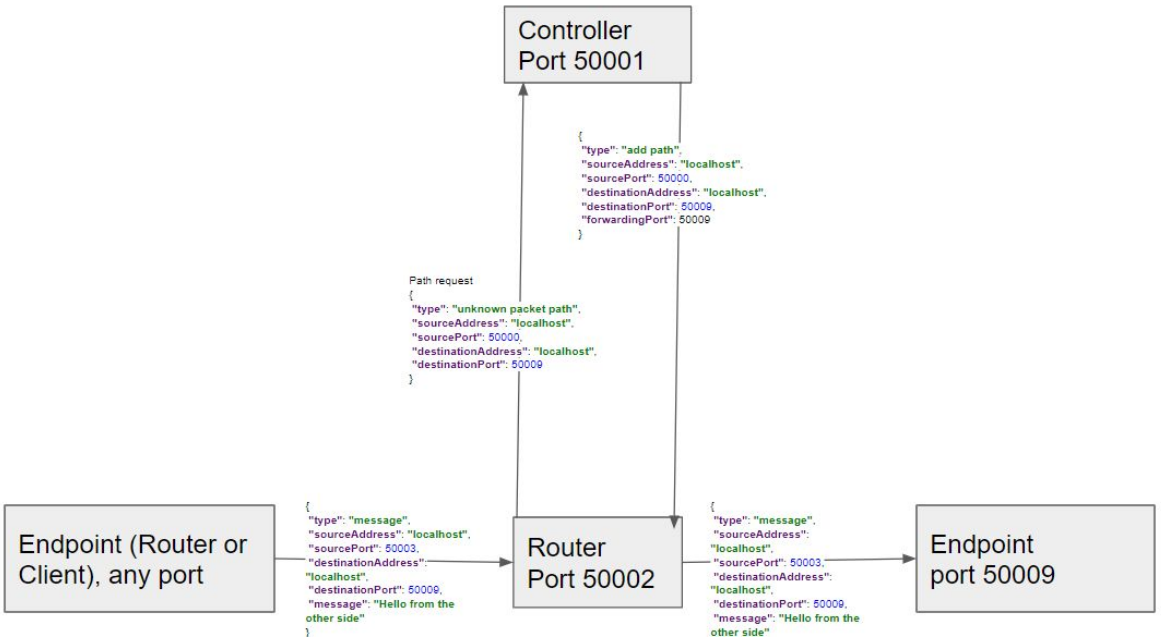
The router has one role which includes different types of communication. The first type is forwarding a packet. When it received a packet from an endpoint of type "message" it will look in its table if it knows where to send that packet to, if it does it will simply forward that packet to the assigned port in his table. If it does not know where to send that packet, we then get on to the second type of communication the router does.

When a router receives a packet and does not know where to send it, it will create a new packet containing the source and destination data, set the type to "unknown packet path", send that on to the controller and add the packet to forward to a queue.

When the router then receives a packet from the controller with type "add path", the following actions take place :

1. The router creates a new PacketsEndpoint instance with the source and destination data from the received packet.
2. A string is made from the instance of the PacketsEndpoint, that string is used as the key of the router's hashmaps (both the one containing the forwarding data and the queued packets). A new entry is added into the hashmap with the forwarding details, the key is the string we just mentioned and the value is the "forwardingPort" assigned by the controller.
3. After this is executed, the router checks if it has packets waiting to be sent based of that same string (used as the key once again) and sends those packets to the allocated destination (if there are any).

Here is a diagram showing this exact process :



f. Client

The client can send and receive messages. When starting an instance, the program will ask for multiple information through the command line :

- A port for the client to use (must be available, my program will check for this).
- A router port to use (where to send packets to). This port must currently be used (the program will check this and proceed to the next step or ask again).

The program will then keep listening for user input in the command line. The only option in this case is to enter “1”, that will trigger the “send a message” method.

When that method is called, here is the flow of events :

- The program asks for a port to send the message to (must be in use).
- The program asks for the user to input the message to send
- A packet is created and sent to the router that the user originally specified. The router will then forward that packets depending on the route given by the controller.

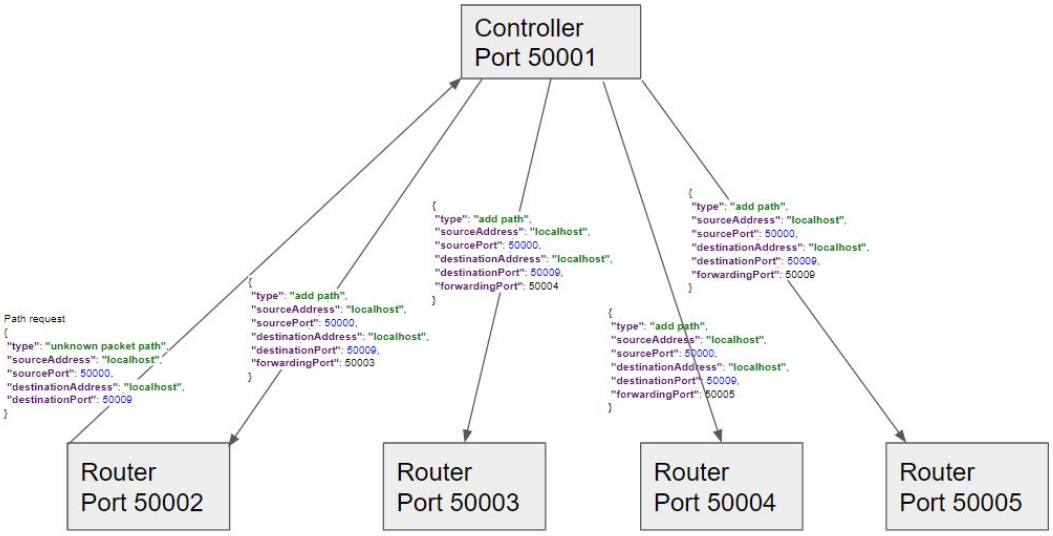
When a client receives a packet of type “message”, that message is printed in the command line along side information about where the message comes from.

g. Messaging Flow

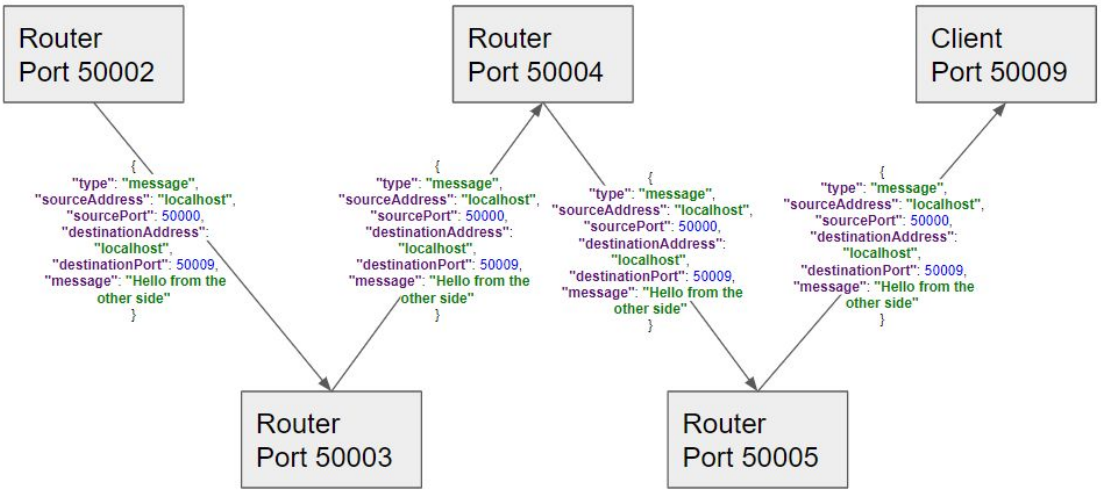
This part just links what I just explained together. Here is the flow for the sending of a message from one client to another when the routers do not already know the path.

Step 1 : A client sends a message to its allocated router.

Step 2: The router asks for information on where to send that packet to the controller. The controller sends information to every router involved.



Step 3: The routers send the message to their allocated endpoint. The packet eventually is received by the endpoint specified by original sender.



In the case that the client sends a second message to the very same endpoint, the routers will already have information on how to forward that packet so step 2 will be ignored.

It is also important to note that these diagrams ignore acknowledgement packets completely. They are more of a high level view of what the process looks like.

2. Personal opinion and design overview

a. Advantages & Disadvantages

First of all, I chose not to implement acknowledgements properly. This would be crucial if I were to spend more time on this assignment, however I have decided not to. However, since this program sends packet locally it isn't a huge issue in my opinion. It will send acknowledgement messages, however if no acknowledgement packet is received after sending a packet then nothing will be done by my program to send that packet again. Since UDP is the protocol used this is indeed something I should have implemented myself.

Second of all, every single client, router and controller require a separate port, this limits the scalability of my program quite heavily.

This program will also literally just transfer a packet from one client to another through routers. No list of packets or messages is ever kept anywhere, since this assignment leaned more towards implementing the system rather than having some specific messaging features I think this is fine.

Now onto the advantages of my design, I think a big advantage is that it will work with any amount of connected routers (well as many as there are available ports), whether it be 1 or 10 routers, the controller will find a path to link all currently connected routers together. An idea could have been to generate a random number between 1 and the total amount of connected routers and have this amount of routers involved in the forwarding of the packet but I thought that for the purpose of this assignment it would make more sense to connect all routers together, leading to more interesting demos and easier testing. A disadvantage to

this method is that realistically you would not want all routers involved in every single transaction in a real world situation.

I believe that the way I structure my packets using JSON (the gson library from google is required to run this program as Java does not have such objects natively - no other library was used) makes the formatting of packets the easiest it could be. It is efficient in space, allows for easy access once the JSON data is parsed and that way I do not have to deal with any bit manipulation.

When starting Routers and Clients it might not be the most intuitive to have to input these different ports through the command line, however I much prefer this method to having those hardcoded. Being able to choose which router the client will contact also helps with testing and demoing.

b. Personal Opinion

I have enjoyed working on this assignment, there is a lot I could have improved on but I have made the decision to spend that extra time elsewhere (such as preparing for the schols examination). I ended up finishing this assignment much faster than the first one since I already knew about threads, how to send and receive packets and had also found out that using JSON was a major help when sending packets. As a result I probably spent about 10 hours on this assignment all together.