Telecommunications Assignment 2

Name: Marek Betka
Student Number: 16324334

Introduction:

The goal of this implementation was to create a map of routers and a controller to direct data between those routers. The controller is connected to all routers which allows it to have a complete map and sense of direction when it comes to making out a path for the data to travel.

Setup

Since I was working in python I needed to code everything from scratch again, luckily I was able to use my client and server (from assignment 1) with little modification, and to change my gateway so it acts like a router.

I also started off by drawing a diagram of the routers and their connections/distances so that it would be easier for me to visualize and incorporate into the assignment. This diagram is shown later in this report.

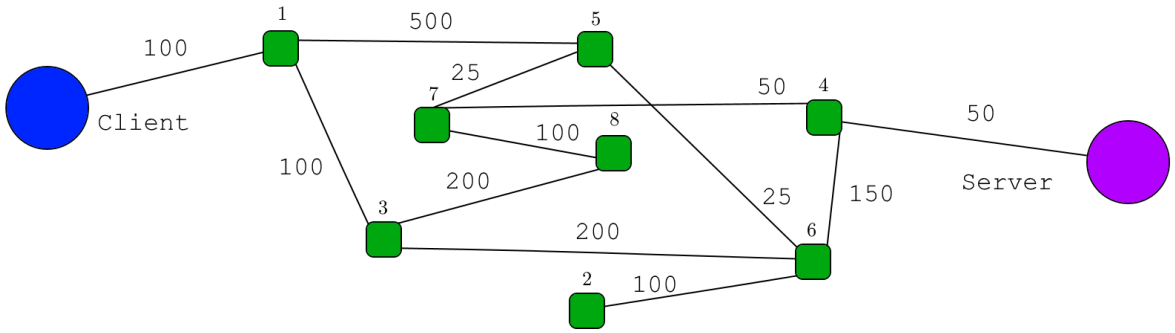
Changes made to client/server/gateway (From assignment 1)

I modified all my code to work with UDP instead of TCP as it did initially, this is because there was a lot more packets that needed to be handled by the routers and it would be easier if a connection didn't need to be established, and instead they were just sent and received as they went out or came in.

The client and server was also changed that it didn't handle errors or acknowledge any incoming messages back to the sender, this is because it was unnecessary and complicated the system more which was not really needed. If something like this were to be applied in the real world this would absolutely need to be included.

The server was never given the functionality to send data, only to receive it as it was not necessary, since we were to just demonstrate the routers and controllers routing data.

Router Map



(Note each green router is also connected to the controller which is not shown on this map)

Router

The most changed file of mine was the gateway class which was almost completely redone. Now instead of taking a domain name and searching for it in and then making a connection to that server, its only job was to forward data from one place to another. This at first seemed easy, and it was but there was a lot more that needed to be changed later on.

Each router had to be modified to know if it was receiving data from another router, the controller, or a client/server. This is because the router needed to know if it was to forward the packet onwards, or if to send it to the controller to get directions on how it should move the data.

This was simply achieved by having the very first character in the packet be set to a letter associated with a different flag. The letters I chose for my flags were “A”, “S” and “D”.

“S” meant the router got data from a client or server and was to forward it to the controller so it would get a route to that location. The message also contains the destination and source of the packet, and the data itself eventually.

After receiving a packet with an “S” flag, the router sends a request to the controller for the path the packet should take. The router holds the original packet and sends the source and destination to the controller. This is to reduce bandwidth usage from the router to the controller as the controller doesn’t care about the contents of the packet in question, just how to get it to the destination. If this were to be implemented properly the packet should be removed after a time of the controller not responding.

After receiving data back from the controller, marked by an “A” flag, the router looks through a list of packets being held until it finds one with the same destination for

one of them, it proceeds to route that packet with the path the controller gave. Again, if this was a real implementation the packets would have a unique id associated with them instead of just checking for the same destination and routing some packets earlier than others.

Finally, if the router receives a packet marked by a "D" flag, it knows the data came from another router, and looks through the initial data to find the next router or client/server to forward the packet to.

Lastly, since it needs to be mentioned, these flags should be the first 2 bits of the entire message, instead of a utf-8 encoded string which would reduce bandwidth usage. However, for the purposes of making it work and to explain it, this is much simpler.

Controller

I believe in this assignment specifications the router was to always send a request to the controller after every movement of the packet. I modified mine slightly so that the entire path is sent back to the router instead of sending a request to the controller each time. While this does have the downside of using more bandwidth between wires, it also has the advantage of being quicker, since the routers would need to talk to the controller only once.

The controller itself has a map of all the routers and clients/servers and is also connected to each of these (Except the clients and servers). This is because it needs to figure out which router is the closest to the destination, if there is more than one router available.

After a lot of experimentation with various functions and algorithms, I decided to use Dijkstra's algorithm to find a route between the source and destination. I used code I found on stackoverflow[1]. This is because while I did understand how the algorithm worked, I could not figure out how to incorporate it with graphs in python.

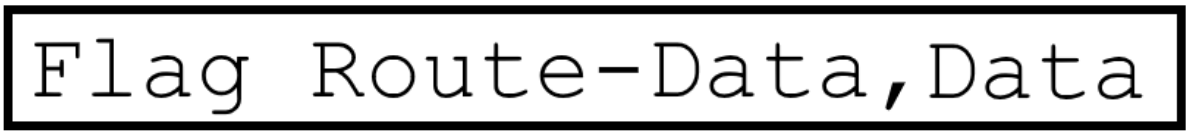
Dijkstra's algorithm for finding the shortest path works by procedurally checking every path one could take, updating the most efficient one as it goes along. It works well for small scales of routers such as this, but for a larger scale it would be better for a powerful computer to make a list of paths one could take when going to a certain destination, and building a large list of them.

Or at least keep that path in the controller for a certain time period, since the server will probably want to talk to the client again and it would now have an efficient path to take. If the path wasn't being used it would eventually be timed out and deleted, but say for example if someone used a website such as facebook frequently, it would be more useful to keep that path instead of calculating it every time they want to visit that site.

[1] https://stackoverflow.com/questions/22897209/dijkstras-algorithm-in-python/22899400

Examples of packets sent

The basic structure of the packet flowing through the system looks as follows:



The flag is always the very first piece of the packet, and since it is the very first piece of data with only length one, it doesn't have to be separated by a delimiter. After this follows the routing data which may contain the source and destination, or a path of routers to take. Finally comes the data which needs to be transferred, this is the only real changeable piece of info by the client/server and is separated by a comma.

Here are a following of packets assisted by a diagram of the path the packets take. The Blue packets are from the client, the Green packets are from the routers, the Red packets are from the controller and finally the Purple packets are from the server. The user input "Hello" into the client as the message. Each packet was decoded using UTF-8 standard decoding. (Note these packets were taken from wireshark and rewritten here for clarity)

Sent slocalhost:10101:localhost:10100,Hello World
Received slocalhost:10101:localhost:10100,Hello World
Sent slocalhost:10101:localhost:10100
Received slocalhost:10101:localhost:10100
Sent a10101:10001:10003:10006:10004:10100:
Received a10101:10001:10003:10006:10004:10100:
Sent d:10006:10004:10100,Hello World
Received d:10006:10004:10100,Hello World
Sent d:10004:10100,Hello World
Received d:10004:10100,Hello World
Sent d:10004:10100,Hello World
Received d:10004:10100,Hello World
Sent d:10100,Hello World
Received d:10100,Hello World
Sent d,Hello World
Received d,Hello World

The client automatically encodes the data entered by the user after adding a flag, and the source and destination. Note that the flag only takes up one character and is not separated by a delimiter since it is predictably the first one only.

The first router (Labelled 1 in the diagram), receives this data and sends a packet to the controller, with only the source and destination, while the user data is held.

After receiving this data, the controller finds the shortest path and changes the flag so the router knows to match the data with a packet.

Since the full path is sent only the next parts of the path are resent by router 1. (As in, the source 10101, itself, 10001, and the next router 10003 is not sent to the next router) The router also modifies the flag so other routers know that it has to continue pushing the data along.

This continues until the server finally receives the data.

Conclusion

In conclusion, this assignment helped me in understanding the things you might need to carry out when setting up a routing system such as this, even if not all of them were done they were definitely acknowledged in this report. The thing that was the most difficult to do was to find the shortest path of the routers as there were many potential algorithms to do so, and the implementation was hard.

The things that could have been improved have already been mentioned but to re iterate: The flags and route data could have been incorporated on the bit level, instead of an encoded string, routers should have a unique id associated with the packet being held, error handling and correction could be added, and finally the controller could temporarily hold the shortest path to/from a destination instead of recalculating them every time.

In total, including the documentation, the assignment took me around 18 hours to complete. The largest chunk of this was figuring out how to structure my routers in python and find the shortest path for them.