



CSU33031 Computer Networks

Assignment 2 Flow Forwarding

Merlin Prasad Std# 19333557

November 20,2021

1 Introduction	2
2.Theory of Topic	2
2.1 Component 1 : Application	2
2.2 Component 2 : Forwarding Service	2
2.3 Component 3 : Controller	3
2.5 Packet Description	3
3.Implementation	3
3.1 Component 1 : Application	5
3.2 Component 2 : Forwarding Service	6
3.3 Component 3 : Controller	8
3.5 User Interaction	10
3.6 Packet Encoding	11
4. Discussion	15
5.Summary	16
6.Reflection	17
7.References	17

1 Introduction

The assignment focused on designing a protocol that forwards packets using forwarding tables based on User Datagram Packets (UDP) in order to learn about decisions made in forwarding flows of packets and what information will need to be kept at network devices to facilitate this. To make sure the quickest route is used and reduce the complexity of the services they contact a controller that has the routing information .

I modeled my forwarding service based on two applications who want to contact each other on trinity's networks. To do this applications need to send to forwarding services until the packet is forwarded successfully to the correct destination. I also aimed to have a forwarding mechanism that was easily scalable and flexible.

2.Theory of Topic

I looked at Internet Protocol Version 4 (IPV4) and Internet Protocol Version 6 (IPv6) header information to consider what to include in my own Type Length Value (TLV) packet [\[1\]](#). I also looked into open flow protocol to understand similar protocols to the one I was designing [\[2\]](#).

2.1 Component 1 : Application

Each end user contains an application that allows the user to decide whether to forward or receive a packet. They are hosted on different ports to the default service port of 51510. Once the user has decided whether to receive or send a packet the application will contact the forwarding service that is on its same Internet Protocol (IP) address.

2.2 Component 2 : Forwarding Service

The forwarding service handles the sending of packets so that it reaches its correct destination. All forwarding services will be on the same port number of 51510. Each forwarding service contains a forwarding table which tells it the quickest route to the desired destination of the packet. The service will access the table and use this information to figure out where to forward the packet to next . If the table does not contain this information the service will contact the controller to ask for a flow modification packet. Once this packet is received this information is used to update the table and an acknowledgement is sent back to the controller.

2.3 Component 3 : Controller

The controller has knowledge of all the connections between forwarding services in the protocol. It has its own forwarding table with the quickest route to get to all the destinations . When it receives a TLV packet from the service it checks its table and sends the destination the service should send the packet to next.

2.5 Packet Description

I decided that all my packets should be encoded as TLV for uniformity. The packets enclose values such as network IDs and container names to support my protocol.

3.Implementation

The code for my protocol is run in 7 separate containers in docker called endpoint, endpointB ,controller , router, routerB ,routerC and routerD .By enabling Xlaunch tcdlib.io terminals are able to launch on windows and the packets are also available to view on Wireshark by searching UDP.

All the routers contain a forwarding service. Each router is connected to multiple subnetworks. Endpoints contain both a forwarding service and an application.They are connected to routers on subnetworks as well. The controller is connected to all the containers on the default docker network. If a forwarding service does not know where to next send its packet to reach its destination it contacts the controller.

My protocol is split into multiple subnetworks and each container contains multiple IP addresses which depend on which network is being used.

Subnet	IP Addresses
net1	172.30.0.0/16
net2	172.40.0.0/16
net3	172.50.0.0/16
net4	172.60.0.0/16

Figure1 : Shows the 4 subnetworks that were created in this protocol

Container	IP Addresses
Endpoint	172.30.0.3 172.17.0.7
Router	172.30.0.2 172.40.0.2 172.17.0.4
RouterB	172.40.0.3 172.50.0.2 172.17.0.5
RouterC	172.50.0.3 172.60.0.3 172.17.0.6
RouterD	172.40.0.3 172.17.0.9
EndpointB	172.60.0.2 172.17.0.8
Controller	172.17.2

Figure 2: Shows the 7 containers in the protocol and their associated IP Addresses. Each container is connected to multiple subnetworks and also the default docker network.

All services and the controller run on port 51510 while application A runs in port 49600 and application B runs in port 53521.

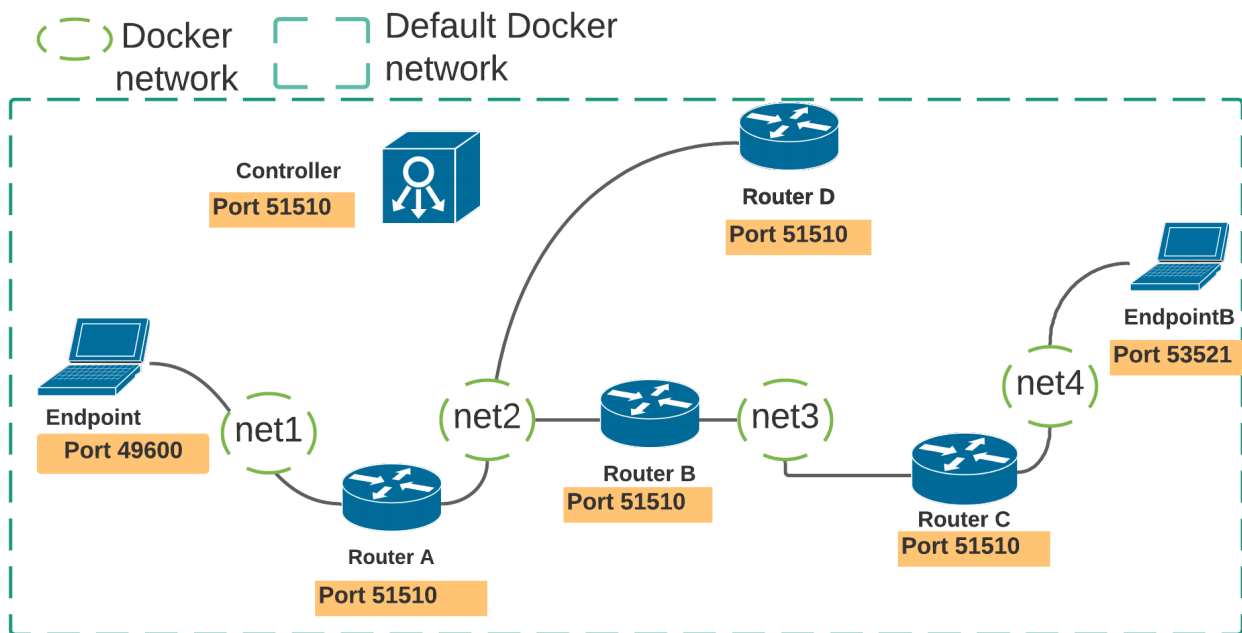


Figure 2: Shows the connection of routers and endpoints in my protocol. Each router and endpoint contain a forwarding service that is on port 51510. Communication occurs on different subnetworks in the protocol and on the default docker network.

3.1 Component 1 : Application

Each application has its own distinct port number. Application A runs on port 49600 while Application B runs on port 53521. There are currently only two applications that exist in my protocol however it is easy to implement more applications using the same code. Application A runs on scss.tcd network ID while application B runs on maths.tcd network. All user interaction is handled by the application.

If the user wants an application to receive packets the application sends the port number and network ID to the forwarding service that is on its own container. If the forwarding service on this endpoint gets a packet that matches this string it forwards it to the application.

On the other hand if the user wishes to send a packet the application will ask the user for which application or network to send the packet towards. The user is also prompted to input a message to attach to the payload. Once the packet is sent the application can send further packets as required to the same or a different destination.

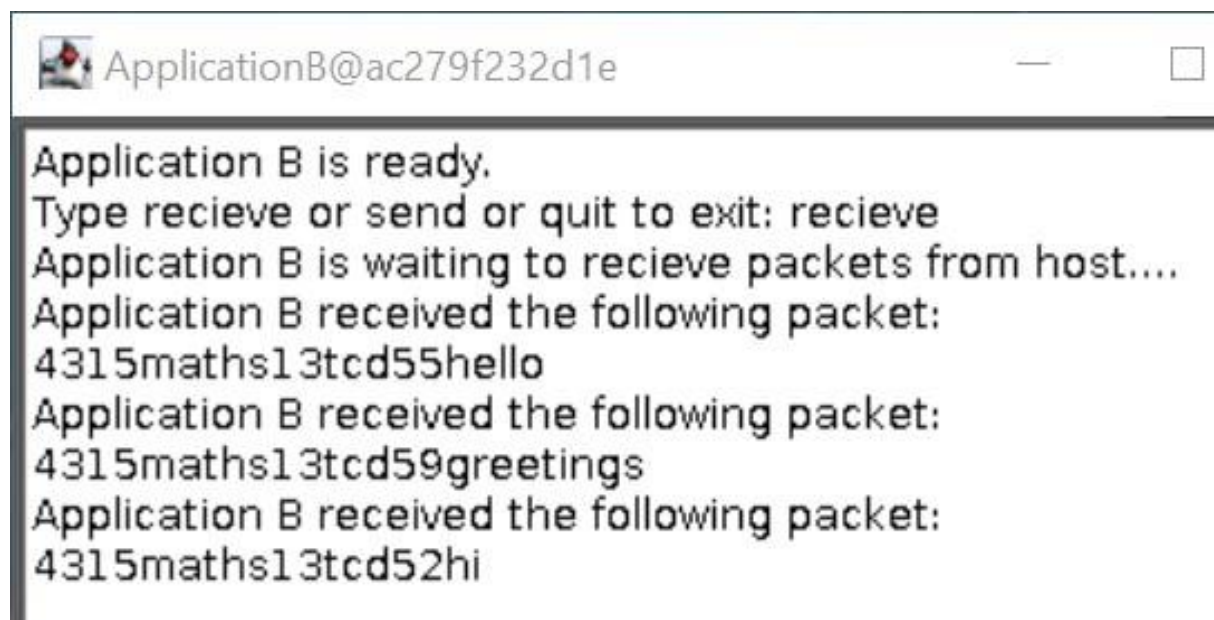


Figure 3 : application B in endpoint B is now ready to receive packets. It has received 3 separate packets which contain the messages “hello” , “hi” and “greetings” from another application.

3.2 Component 2 : Forwarding Service

The forwarding services are all bound to port 51510. Each router and endpoint contain a forwarding service. Once a forwarding service is run it contacts the controller with an initial receive packet to signal that the service is ready for communication .

In my protocol I connect the different services to different subnetworks so each service has its own IP address. I use container names to send packets between services in my subnetworks as docker can figure out the IP address of a container using the port and container name.

When an application wishes to receive a packet, the service that is on the same container will receive this message. The service will examine the header and keep track of the string and the port number from the packet it received.

I implemented the forwarding table as a hashmap where the key is the destination the packet wants to reach and the value is the container name of the next container to hop towards to get closer to the desired location . Initially the values in the hashmap are empty . Each service analyses the headers of TLV packets it receives and then checks its own table to figure out where to hop to next . If the values for the destinations are unknown the service forwards the packet it received to the controller. When it receives the flow modification packet from the controller in TLV format it updates the hashmap with the container name. It also sends an acknowledgement (ACK) to the controller to let it know that the flow modification packet was successfully received . The service will then forward the packet to the correct service .

If the container name received is "appA " or "app B" it means that the service needs to now forward the packet to the appropriate application. The service will check the previously stored string and if they match it will use the port number that was enclosed in the header to send the packet from the service to the application.



Service@7b6df28b7a6a

Service is waiting for contact....

Service recieved the following packet 4315maths13tcd55hello
Packet recieved from network: routerB.net3
Destination is unknown contacting controller

Service recieved the following packet 39endpointB
Packet recieved from network: 172.17.0.2
Service send acknowledgment
Quickest route to maths is through: endpointB
Packet sent by Service

Service recieved the following packet 4315maths13tcd59greetings
Packet recieved from network: routerB.net3
Quickest route to maths is through: endpointB
Packet sent by Service

Service recieved the following packet 4315maths13tcd52hi
Packet recieved from network: routerB.net3
Quickest route to maths is through: endpointB
Packet sent by Service

Figure 4: Shows the forwarding service that is in the container router C. The forwarding service waited for contact . Once it received the packet from routerB it checked its own forwarding table. As the destination was unknown it contacted the controller on how to get to the network ID “maths”. It received a packet with the container name “endpointB”. This information was then used to update the services forwarding table and an ACK was sent to the controller. The packet was then successfully forwarded to the container endpointB. When further packets were received for the same destination the service was able to forward the packet directly to the location without needing to contact the controller as the forwarding table contained the information required.

3.3 Component 3 : Controller

The controller contains information about the forwarding tables for each service in the protocol. The controller sits on the default docker network on IP address 172.0.0.2 in my protocol and in this way it is already connected to all the other services without needing to create a separate subnetwork . Services contact the controller with its specific IP address. By checking the IP address of the packet that was received the controller knows which service wants to contact it. It then examines the destination in the header and sends a flow modification packet which contains the container name of where the service should forward the packet towards next. It will receive an ACK when this was successful .

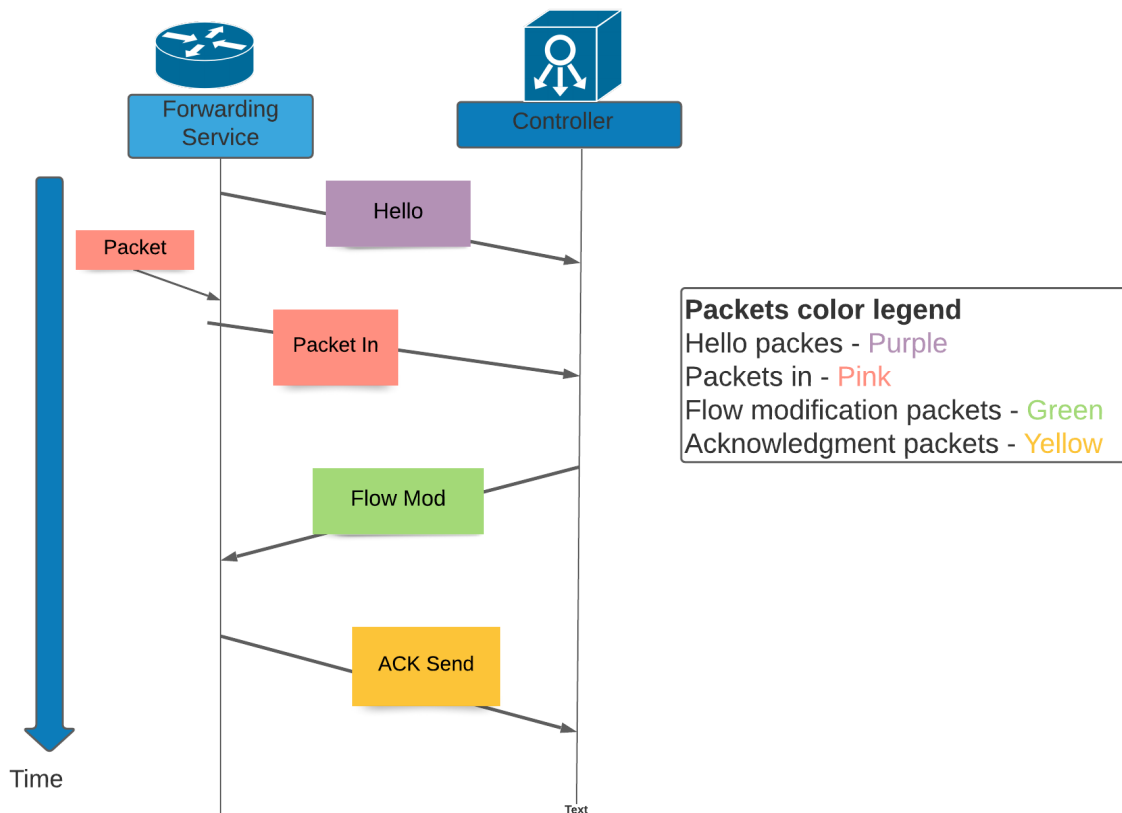


Figure 5: Flow diagram explaining the communication between forwarding services and controller. Services send an initial packet to establish connection. If a service receives a packet to an unknown destination it contacts the controller. The controller will then send a flow modification packet to the service . Services send ACK's to the controller to show it received the flow modification packet successfully.



The screenshot shows a terminal window titled "Controller@23a5b3d90c97". The log messages are as follows:

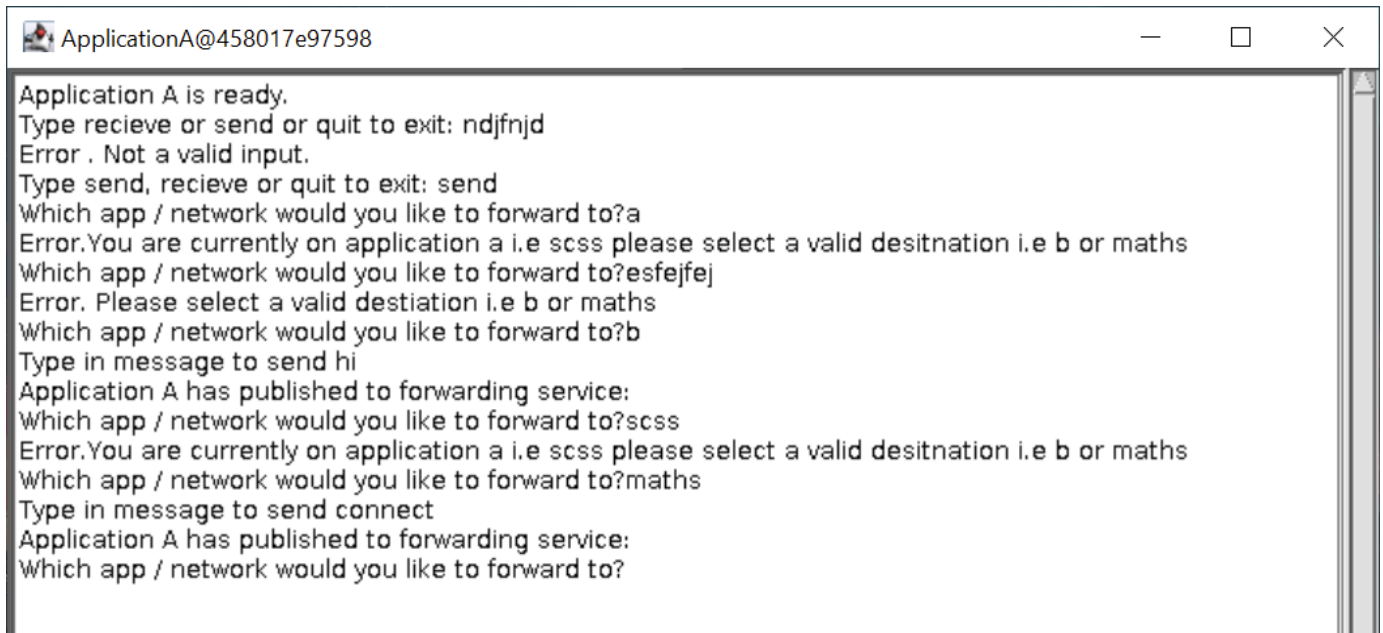
```
Controller is waiting for contact...
Controller recieved the following packet 67connect from172.17.0.3
Controller recieved the following packet 67connect from172.17.0.6
Controller recieved the following packet 67connect from172.17.0.5
Controller recieved the following packet 67connect from172.17.0.4
Controller recieved the following packet 67connect from172.17.0.7
IP Address of packet recieved : 172.17.0.7
Controller send update to forwarding table.
Controller recieved the following packet 23ACK
IP Address of packet recieved : 172.17.0.4
Controller send update to forwarding table.
Controller recieved the following packet 23ACK
IP Address of packet recieved : 172.17.0.5
Controller send update to forwarding table.
Controller recieved the following packet 23ACK
IP Address of packet recieved : 172.17.0.6
Controller send update to forwarding table.
Controller recieved the following packet 23ACK
IP Address of packet recieved : 172.17.0.3
Controller send update to forwarding table.
Controller recieved the following packet 23ACK
```

Figure 6: The first 5 messages show the initial connection message from each of the services in TLV format. Each service sends the packet “6 7 connect” on start-up to the controller to establish connection. Type 6 stands for connect to controller , 7 is the length of the value “connect”. The further messages show different services contacting the controller to figure out where to forward the packet to next. They are all connected to the docker bridge network . IP address 172.17.0.7 stands for the service in endpointA contacting it. The controller informed the service to next forward the packet towards the service in the container called “router”. Once the service received this packet it sent an ACK towards the controller. All the other services go through a similar sequence of steps.

3.5 User Interaction

All user interaction is confined to the application . The user is able to decide on the message and application it wishes to send a packet towards. If the user decides to send a packet in application A they are able to send to the destination by either type “b” or appb for application B or maths. Similarly in application B packets can be sent by typing “a” or “appa” for application A or scss . They can send as many packets to a destination as they like. The user can also decide to make an application receive a packet instead.

As there is a chance for human error in inputting values I implemented error handling in my application so mistakes would not be issued. Precise error messages are also displayed which allow the user to know exactly how to avoid the same mistake.



```
ApplicationA@458017e97598
Application A is ready.
Type recieve or send or quit to exit: ndjfnjd
Error . Not a valid input.
Type send, recieve or quit to exit: send
Which app / network would you like to forward to?a
Error.You are currently on application a i.e scss please select a valid desitnation i.e b or maths
Which app / network would you like to forward to?esfeifej
Error. Please select a valid destiation i.e b or maths
Which app / network would you like to forward to?b
Type in message to send hi
Application A has published to forwarding service:
Which app / network would you like to forward to?scss
Error.You are currently on application a i.e scss please select a valid desitnation i.e b or maths
Which app / network would you like to forward to?maths
Type in message to send connect
Application A has published to forwarding service:
Which app / network would you like to forward to?
```

Figure 7: Demonstrates error handling in applications . Application A was ready to send or receive packets. When the user inputed an invalid input an error message is displayed informing the user of their mistake and allowing them to retry. When they tried to forward to the destination of the network they are currently in, the user was also informed this was wrong .

3.6 Packet Encoding

I encoded all my packets in TLV format to simulate real world protocols better. There are 7 different types of values. By allowing for combination types I can add numerous TLV values together in one packet.

The following table explains the 7 different types of information enclosed in my TLV packet.

Type	Description
1	Network ID
2	Acknowledgements (ACK)
3	Container name
4	Combination
5	Message
6	Connect to controller
7	Port number

Figure 8: shows the table of types in the TLV encoding scheme.

4	3	1	4	scss	1	3	tcd	5	2	hi
---	---	---	---	------	---	---	-----	---	---	----

Figure 9 : shows an example of a packet with TLV encoding. The 4 at the start indicates this is a combination type which can include any number of further TLV in this case there are 3. The first TLV is 14scss . The 1 shows its a network ID type , with a length of 4 and its value is "scss". The second TLV is 13tcd. This is to allow for hierarchical network IDs which in this example is scss.tcd. Finally the last TLV contains 52hi. The 5 shows this is a message type which is of length 2 with the value "hi". This message length and value will depend on what the user inputs in the application.

TLV Packet Structure

To allow for the communication of key information between applications , I created the following TLV packet . Maximum size of packet : 65535 bytes .

Title	Bytes	Description
Type	1 to 3 bytes	Type of TLV packet . If it is a combination type it will also include the number of TLVs it will contain, and the type of the first packet.
Length	1 byte	Length of first value.
Value	4 to 5 bytes	The first value. My packet will always include the network ID as the first value. Currently only maths and scss are implemented.
Type2	1 byte	The type of the second value. Currently the second value is always 1 for network ID.
Length2	1 bytes	Length of the second value.
Value2	3 byte	The tlv packets all include tcd as their second network ID currently in the protocol.
Type3	1 byte	The type of the third value. Currently the third value is always 5 for message type.
Length3	1 byte	Length of the third value.
Value3	Variable	The user defined message

Figure 10 : explains the structure of TLV packets sent by applications to forwarding services.

The packet content class deals with how to create UDP packets.

The packet content class has been extended to include flow modification packets , acknowledgement packets, receive packets and the standard TLV packet . All packets are encoded in TLV for standardisation

The TLV packets are sent from applications to forwarding services when they want to send a packet. There are currently two network IDs in my protocol . They are scss and maths as they are both departments at Trinity . With the use of combination type hierarchical network IDs can be built. In my protocol I created scss.tcd and maths.tcd however someone else in the future can use my TLV to create more complex hierarchies easily.

I also included the option for the user to type in a message to be enclosed in the payload of the packet with the message type.

1661	10.434918	192.168.65.5	172.17.0.7	DNS	80	Standard query respon
4153	25.012360	172.17.0.7	172.17.0.2	UDP	83	51510 → 51510 Len=41
4181	26.163734	172.17.0.2	172.17.0.7	UDP	68	51510 → 51510 Len=26
4182	26.170303	172.17.0.7	192.168.65.5	DNS	83	Standard query 0x80ec
4211	27.299941	192.168.65.5	172.17.0.7	DNS	83	Standard query respon

Frame 4153: 83 bytes on wire (664 bits)		83 bytes captured (664 bits)	
00	02 42 ac 11 00 02 02 42 ac 11 00 07 08 00 45 00	·B·	·B·
10	00 45 cf 6b 40 00 40 11 13 11 ac 11 00 07 ac 11	·E·k@·@·	·
20	00 02 c9 36 c9 36 00 31 58 6e ac ed 00 05 77 23	·	·6·6·1 Xn·
30	00 00 00 1e 00 03 34 33 31 00 01 35 00 05 6d 61	·	·43 1·5·ma
40	74 68 73 00 05 31 33 74 63 64 00 07 35 35 68 65	ths·	13t cd·55he
50	6c 6c 6f		llo

Figure 11 : shows the encoding of TLV packet sent from forwarding service in endpointA to the controller. The packet is encoded as a TLV as follows: 4315maths13tcd55hello.

The 4 at the start indicated this is a combination type which can include any number of further TLV in this case there are 3. The first TLV is 15maths . 1 shows its a network ID type , the network has a length of 5 and its value is “maths”. The second TLV is 13tcd. Finally the last TLV contains 55hello. The 5 shows this is a message type which is of length 5 with the value “hello”.

The forwarding service sends ACK when it receives flow modifications successfully . The payload has a fixed length of 21 bytes and contains the message “23ACK” which make them easily identifiable in wireshark

```
response= new AckPacketContent("2","3","ACK").toDatagramPacket();
response.setSocketAddress(packet.getSocketAddress());
socket.send(response);
terminal.println("Service send acknowledgment ");
```

Figure 12 : shows a code snippet from the Service which shows how an ACK packet is encoded as a TLV. The 2 at the start show its a ACK type with a length of 3 with the value “ACK”.

Receive packets are sent by applications when they wish to accept packets. They contain the string of the network they will receive on as well as the port number of the application itself. The forwarding service that is on the same container as the application will take this packet and will take note of the string and port number. If the forwarding service receives a packet that matches the string it forwards it to the application .

Controller sends flow modification packets enclosed as TLV as well. These packets contain the container name of the next destination the forwarding service should forward towards.

4199	95.758515	172.30.0.3	172.30.0.2	UDP	83 51510 → 51510 Len=41
4200	95.768082	172.17.0.4	172.17.0.2	UDP	83 51510 → 51510 Len=41
4243	99.083529	172.17.0.2	172.17.0.4	UDP	69 51510 → 51510 Len=27
4244	99.088544	172.17.0.4	192.168.65.5	DNS	83 Standard query 0xd437 P
4259	100.244293	172.40.0.2	172.40.0.3	UDP	83 51510 → 51510 Len=41

Frame 4243: 69 bytes on wire (552 bits), 69 bytes captured (552 bits)					
Ethernet II, Src: 02:42:ac:11:00:02 (02:42:ac:11:00:02), Dst: 02:42:ac:11:00:04 (02:42:ac:11:00:04)					
Internet Protocol Version 4, Src: 172.17.0.2, Dst: 172.17.0.4					
User Datagram Protocol, Src Port: 51510, Dst Port: 51510					
Data (27 bytes)					
00	02 42 ac 11 00 04 02 42 ac 11 00 02 08 00 45 00	.B....B.....E.			
10	00 37 93 8b 40 00 40 11 4f 02 ac 11 00 02 ac 11	.7..@.@.O.....			
20	00 04 c9 36 c9 36 00 23 58 5d ac ed 00 05 77 15	...6.6.#X]...w.			
30	00 00 00 28 00 01 33 00 01 37 00 07 72 6f 75 74	...(.3.7..rout			
40	65 72 42 00 00	erB..			

Figure 13 : shows TLV flow modification packet received by router A . The controller sends the message “37routerB”. The service will update its forwarding table with this value and then send the packet to routerB which you can see on the last line. 172.40.0.2 is the IP address of router A and 172.40.03 is the IP address of router B.

4. Discussion

There are various choices I made when designing this protocol. I used Docker as it allowed me to create multiple subnetworks and run my components on different hosts to better simulate real life flow forwarding protocols.

I implemented a hashmap for the forwarding table and this let me ensure only one unique destination was being stored for each network. I stored container names instead of IP addresses in the forwarding table to help improve readability. This also meant docker dealt with the resolving of IP addresses allowing for a more flexible solution for when my forwarding services were communicating between each other. If someone wished for IP addresses to be sent instead they would just need to modify the forwarding table in the controller slightly.

I used IP addresses to communicate between the forwarding services and the controller over the default network. I think it's important that the controller is connected to every component in the network and this way docker will automatically allocate an IP address for each new container. I chose this approach over creating a separate subnetwork for just the controller and forwarding services because that would require manually configuring the controller and each new forwarding service so it is connected to this network. It would also raise additional problems like ensuring that the forwarding services will only use the network to communicate with the controller and not also use it to communicate with each other. So the use of IP addresses felt like a more simpler and efficient solution to the problem.

Adding more services can be easily facilitated by my implementation as it would require minimal code changes. I decided to limit myself to 4 routers as I did not want to overcrowd my protocol with too many unnecessary routers such as routerD where it is not useful to hop towards currently. Also as each endpoint had a forwarding service as well sending a packet between the two endpoints requires at least 5 hops.

Running all the forwarding services on the same port meant only applications have unique port numbers which allows for easy identification on Wireshark and for forwarding of packets using container names.

I allowed for both applications to be able to send to each other as I think this better mimics real life situations as you would wish to send messages back and forth. You can also send multiple packets to a destination. Currently there are only two destinations in my protocol however by letting the services remember the port numbers when applications wish to receive means adding more applications to the protocol is trivial.

The controller carried out all high level routing decisions in my protocol. This is because I was trying to use open flow as the basis for my protocol. In open flow protocols the switches only carry out forwarding and that is the same way my routers work. Forwarding tables inside the service start off with no information about how to reach the destination. I believe forwarding services should be kept simple and only have a minimal amount of data. The controller should instead keep track of the locations of specific routers and applications and know what quickest way to go through the services to reach the network destination. This also meant that once a service knows what the quickest route is they will keep using that information and do not need to contact the controller again. This better simulates real life flow forwarding protocols as they split up the network devices in a similar fashion.

All packets in my protocol use TLV encoding for standardisation. I send an initial connection packet between services and the controller to show that the services and controllers were connected. As well as this, services send ACKs to controllers when flow modifications are successful further improving the security of my protocol.

The numerous types in my TLV also facilitate sophisticated communication between my protocols such as the use of a message type to allow communication between the two applications. In addition to this I created a combination type for my TLV as this allowed for hierarchies to exist in my networks. While currently there are only two network destinations, someone in the future can take advantage of the already existing features to build their own network hierarchy .

5.Summary

My combination of applications, forwarding services and a controller help me demonstrate a forwarding protocol to a realistic degree. The usage of TLV encoding also enabled me to communicate between my components in a clear way. My forwarding protocol implements features of the open flow protocol . The protocol is designed in scalable and flexible manner aswell.

In conclusion, my forwarding protocol is able to handle communication between multiple applications which allows me to send packets in an efficient manner between networks.

6.Reflection

I am quite proud of the forwarding protocol I designed given the time constraints. If I had more time I would have liked to have created docker compose files to automate some of the tedious process of setting up my containers.

I ended up hardcoding the container names into the forwarding table. While this is satisfactory I believe that a more impressive solution would have involved the controller calculating the shortest path between destinations itself and updating the table as required. Converting the connections between services into edges and nodes and running a dijkstra algorithm such as is done in linked state routing would have created a more impressive controller. This would also mean the controller would send updates to the forwarding services periodically so that it's forwarding table would have the quickest route with any additional services. However real life protocols such as open flow are much more commonly used and trying to do this would have introduced a lot more unnecessary complexity to my code. The main purpose of this assignment was to learn about how to connect components to carry out flow forwarding and this extra addition would have been beyond the scope of the assignment . Another interesting addition would have been to measure latency between different services instead of just the hops to calculate the shortest path.

Trying to decide what to include in my packet as TLV encoding helped me learn more about IPV4 and IPV6 and how IP addresses are used for communication between components in a protocol. If I had more time I would have liked to research more extensively on TLV encoding and include more types in my packets.

Overall I feel like this assignment helped me learn quite a bit about networking and flow forwarding protocols such as open flow.I believe that my final protocol is sophisticated and I have been able to achieve the learning objectives of this project.

7.References

[1]<https://networklessons.com/cisco/ccna-routing-switching-icnd1-100-105/ipv4-packet-header> visited Nov 2021

[2] <https://www.cables-solutions.com/whats-openflow-switch-how-it-works.html> visited Nov 2021