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

School of Computer Science and Statistics

Flow Forwarding Report

Student: TIANZE ZHANG

Contents

1 Introduction

This report gives a detailed description of designing and implementing a forwarding mechanism which supporting carrying flow of packets from senders to receivers through the predefined simulated network of routers. The roles played by the key components: central controller, routers and their interactions are outlined. The design and implementation of the message protocol, which is TLV (type length value) UDP (User Datagram Protocol) based and its functionalities are illustrated and analysed. The protocol of exchange of messages between controller and routers in the routing process is also designed and implemented.

2 Design Consideration

The challenging of the forwarding mechanism is to efficiently link different networks and form the optimized routes from source to destination in a security manner.

In this solution, a logically centralized approach is adopted to update routing info from controller to the routers forwarding table. In this way, the controller has global knowledge of the network, and the optimised routing algorithms can be used for the packet forwarding decision. Thus, the protocol for communicating between controller and router are needed.

2.1 Protocol and Forwarding table

The communication protocol between controller and routers supports requesting routing info from central controller and populating the forwarding table in the routers in an on-demand approach. Ideally, the controller should support pushing the updated route info to the routers. Forwarding table in the router is the first contact to get the route info for the next hop, while the table in the controller is the master which holds the global view of the network routers info.

2.2 Key components

Controller is the manager of the networking which holds all the routing info, and it is the source of the network topology knowledge. The routers request the routing info if it’s not available in the forwarding table.

Router makes the forward decision and get the next hop based on the forward table, then forwarding the packet.

Application and forwarding service is responsible to send the UDP datagram on the specific port number. The packet is created based on the TLV-header plus payload protocol, and send to the forwarding service.

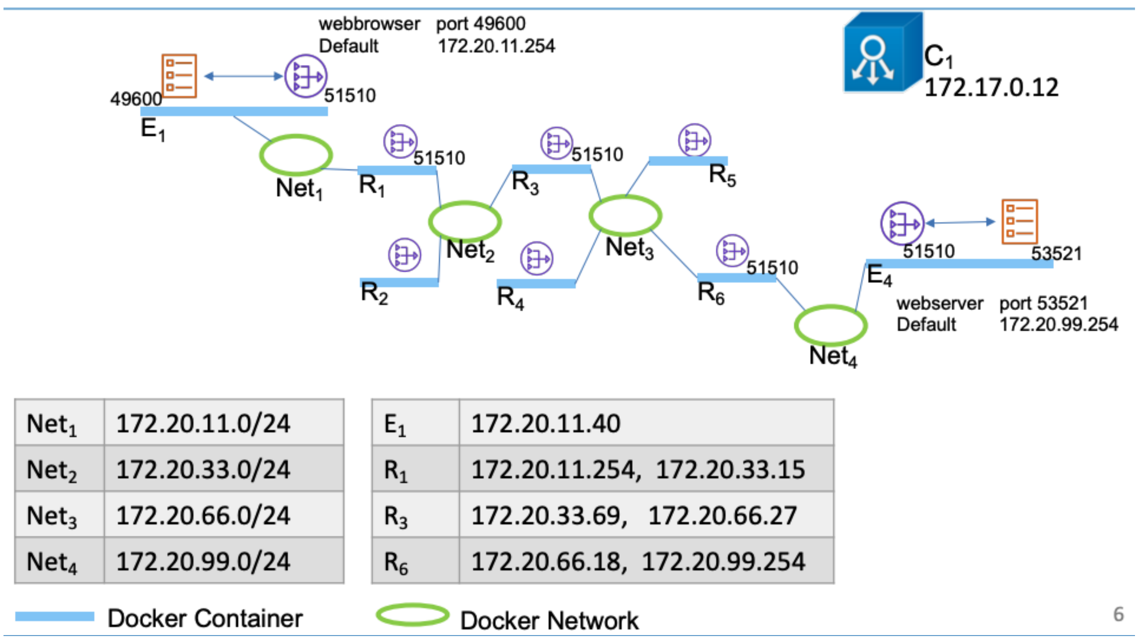
2.3 Environment Setup

Docker: utilize docker containers and network to create simulated environment on the local machine. According to the predefined network topology, subsets are created and containers are allocated across them. The running containers are the simulation of the network nodes. (p52, Rajdeep)

Wireshark for packet sniff, and protocol analysis.

3 Implementation

My Implementation contains 11 components: 6 routers (Router1, Router2, Router3, Router4, Router5, Router6), 2 applications with 2 forwarding services (ApplicationE1 with Service1 and ApplicationE4 with Service2) and a controller (Controller). My implementation allows applications to communicate which each other (send and receive packets) across different networks.



Net_1	172.20.11.0/24
Net_2	172.20.33.0/24
Net_3	172.20.66.0/24
Net_4	172.20.99.0/24

E_1	172.20.11.40
R_1	172.20.11.254, 172.20.33.15
R_3	172.20.33.69, 172.20.66.27
R_6	172.20.66.18, 172.20.99.254

Figure 1: This diagram is an overview of my implementation,

The focus is to enable communication between E1 and E4 which are under different networks.

3.1 Docker Network Setup

Each component runs in a separate container except the application with its forwarding service. To ensure the application and routers are under different networks, docker subnets were used. 4 different subnets were created, and each component was connected to its own subnets as in Figure 1: each router was connected to two docker networks and the application was connected to one. The controller was set to connect to all the network as the routers and the forwarding services need to communicate with the controller.

```
docker network create -d bridge --subnet 172.20.11.0/24 N1
docker network create -d bridge --subnet 172.20.33.0/24 N2

docker network connect N1 r1
docker network connect N2 r1
```

Figure 2: The diagram shows the creation of the networks N1 and N2 in docker, and connecting the container for router 1 to the two networks. This is a general method also used for the creation of the rest of the networks and connecting to these networks

3.2 Routers

Each of the router has the same functionality and port number, but different addresses. They have the same port number of 510510. The routers first wait to receive any packet sent from other routers or forwarding services, then it would find out the destination address that the packet should be sent to. After knowing the destination address of the packet, it would check which address should the packet go next, in order to reach the destination address. This is achieved by checking its own forwarding table. The forwarding table of the routers is represented as a 2D array, which the first column represents the destination address, and the second column represents the address where the router should forward next. In my implementation, each router has an empty forwarding table at the beginning. When the routers do not know where to forward the packet, it sends a packet to the controller requesting the address to forward next. When the controller sends back the packet providing the address, the router’s own local forwarding table also gets updated by adding the information to its table. Therefore, routers won’t ask for the same address for a destination twice.

3.3 Applications

Each Application has the same functionality. However, port numbers for applications are different. For example, in my implementation, ApplicationE1 has a port number of 49600, and ApplicationE4 with a port number of 53521. The application firstly asks whether the user wants the application to send or receive. When the application is used for sending, it will ask for the application the user wants to send to, and finally encode the packet then send to its forwarding service. If the application is used for receiving, it will wait for a packet to come in from its forwarding service and decode the payload of the packet.

3.4 Forwarding services

The forwarding services run on local host: same container of an application. It is used for the application to connect with the routers part at port 510510. The forwarding services accept incoming packets from an application or a router. It does a similar job as the router which it also has its own forwarding table initialized empty. It needs to determine which router to forward the packet. If the application was set as receiving, it needs to send the packet to the application instead. The forwarding service also contacts the controller if it’s local forwarding table is empty or there is not enough information.

3.5 Controller

The controller acts like a manager. It has a hardcoded master forwarding table that gives all the information of the entire system. The controller is connected to all the docker networks, therefore, any router or forwarding service are able to communicate with the controller. The controller first waits to receive any packet. When it receives a packet, it decodes the packet and records the destination address the packet is going to. It also stores the address of the sender making the request, it then checks its local forwarding table which is also a 2D-Array using the senders address and the destination address to find the address required. The controller finally makes a new packet with the address required and sends it back to the router or forwarding table making the request.

3.6 Process and Interaction

The system works as follow:

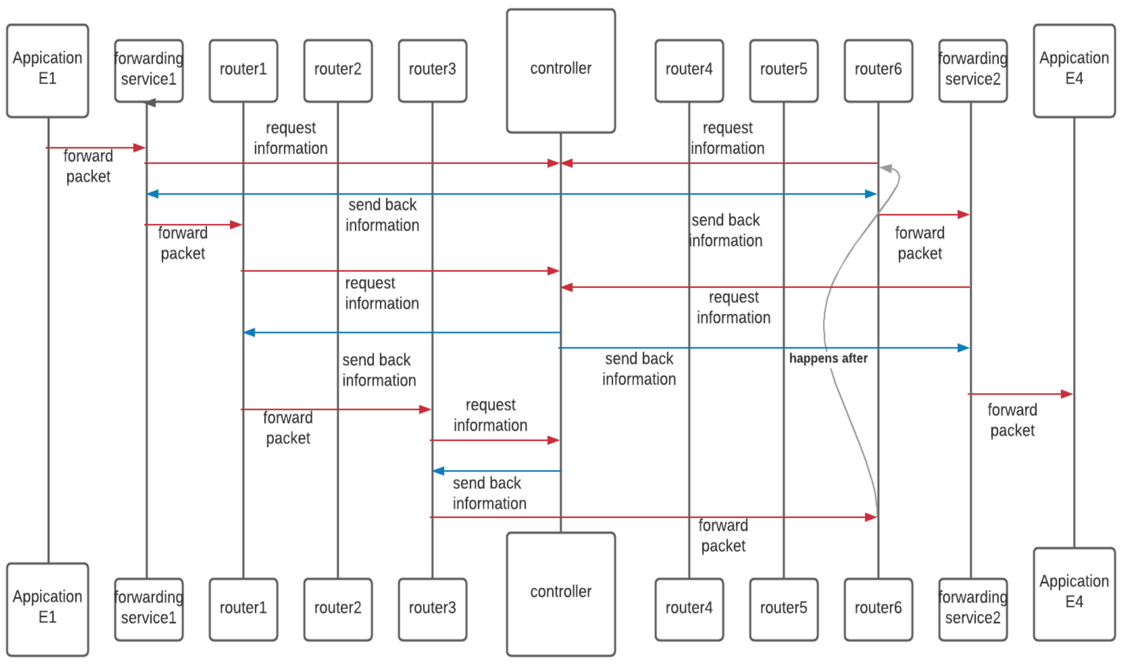


Figure 3: The diagram above is a flow diagram of Figure 1 that demonstrates the overall process and interaction between systems: AppicationE1 creates a packet and forward to service1.Service1 asks controller where to forward next. controller tells to forward to R1 (Router1). Service1 sends to R1. R1 asks controller where to forward next. controller tells to forward to R3 (Router3). R1 sends to R3. R3 asks controller where to forward next. controller tells to forward to R6 (Router6). R3 sends to R6. R6 asks controller where to forward next. controller tells to forward to Service2. R6 sends to Service2. Service2 asks controller where to forward next. controller tells to forward to ApplicationE4. Service2 sends to ApplicationE4.

I would like to present another example after running the previous example. Suppose ApplicationE1 wants to send to ApplicarionE4 again. Since the forwarding table are all updated, they no longer need to send to the controller anymore:

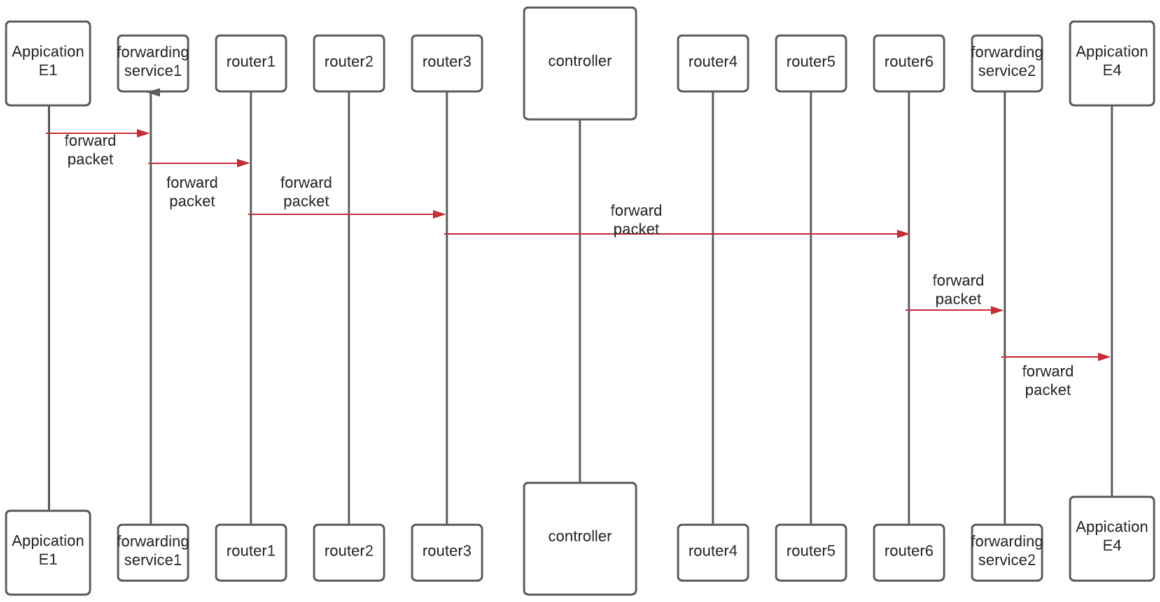
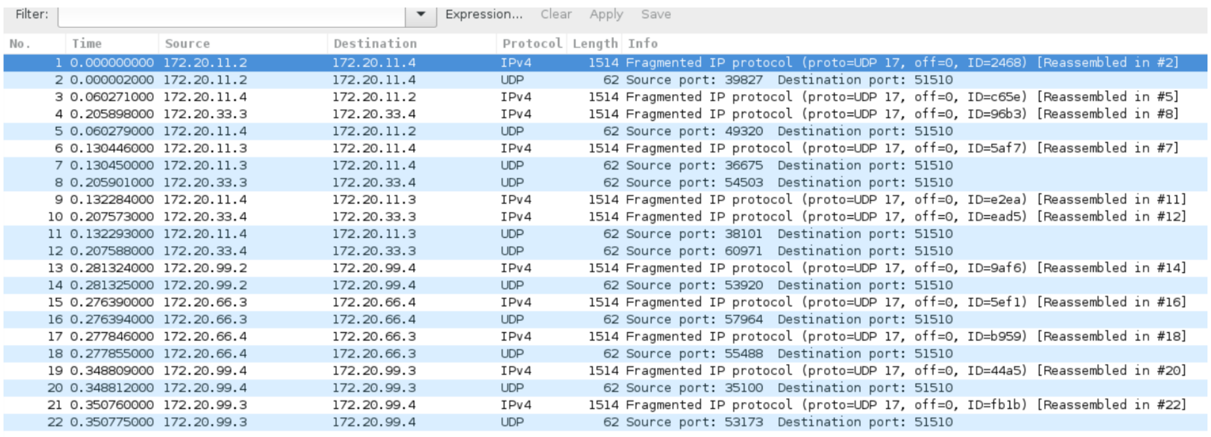


Figure 4: The diagram demonstrates the flow after each component's forwarding tables were updated. If ApplicationE1 wants to send to ApplicationE4, it sends to Service1, Service1 sends to Router1, Router1 sends to Router3, Router3 sends to Router6, Router6 sends to Service2, Service2 to ApplicationE4.



```
Filter:                                        Expression...  Clear  Apply  Save
No.  Time        Source         Destination    Protocol Length Info
  1 0.000000000 172.20.11.2    172.20.11.4    IPv4     1514 Fragmented IP protocol (proto=UDP 17, off=0, ID=2468) [Reassembled in #2]
  2 0.000002000 172.20.11.2    172.20.11.4    UDP        62 Source port: 39827  Destination port: 51510
  3 0.060271000 172.20.11.4    172.20.11.2    IPv4     1514 Fragmented IP protocol (proto=UDP 17, off=0, ID=c65e) [Reassembled in #5]
  4 0.205898000 172.20.33.3    172.20.33.4    IPv4     1514 Fragmented IP protocol (proto=UDP 17, off=0, ID=96b3) [Reassembled in #8]
  5 0.060279000 172.20.11.4    172.20.11.2    UDP        62 Source port: 49320  Destination port: 51510
  6 0.130446000 172.20.11.3    172.20.11.4    IPv4     1514 Fragmented IP protocol (proto=UDP 17, off=0, ID=5af7) [Reassembled in #7]
  7 0.130450000 172.20.11.3    172.20.11.4    UDP        62 Source port: 36675  Destination port: 51510
  8 0.205901000 172.20.33.3    172.20.33.4    UDP        62 Source port: 54503  Destination port: 51510
  9 0.132284000 172.20.11.4    172.20.11.3    IPv4     1514 Fragmented IP protocol (proto=UDP 17, off=0, ID=e2ea) [Reassembled in #11]
 10 0.207573000 172.20.33.4    172.20.33.3    IPv4     1514 Fragmented IP protocol (proto=UDP 17, off=0, ID=ead5) [Reassembled in #12]
 11 0.132293000 172.20.11.4    172.20.11.3    UDP        62 Source port: 38101  Destination port: 51510
 12 0.207588000 172.20.33.4    172.20.33.3    UDP        62 Source port: 60971  Destination port: 51510
 13 0.281324000 172.20.99.2    172.20.99.4    IPv4     1514 Fragmented IP protocol (proto=UDP 17, off=0, ID=9af6) [Reassembled in #14]
 14 0.281325000 172.20.99.2    172.20.99.4    UDP        62 Source port: 53920  Destination port: 51510
 15 0.276390000 172.20.66.3    172.20.66.4    IPv4     1514 Fragmented IP protocol (proto=UDP 17, off=0, ID=5ef1) [Reassembled in #16]
 16 0.276394000 172.20.66.3    172.20.66.4    UDP        62 Source port: 57964  Destination port: 51510
 17 0.277846000 172.20.66.4    172.20.66.3    IPv4     1514 Fragmented IP protocol (proto=UDP 17, off=0, ID=b959) [Reassembled in #18]
 18 0.277855000 172.20.66.4    172.20.66.3    UDP        62 Source port: 55488  Destination port: 51510
 19 0.348809000 172.20.99.4    172.20.99.3    IPv4     1514 Fragmented IP protocol (proto=UDP 17, off=0, ID=44a5) [Reassembled in #20]
 20 0.348812000 172.20.99.4    172.20.99.3    UDP        62 Source port: 35100  Destination port: 51510
 21 0.350760000 172.20.99.3    172.20.99.4    IPv4     1514 Fragmented IP protocol (proto=UDP 17, off=0, ID=fb1b) [Reassembled in #22]
 22 0.350775000 172.20.99.3    172.20.99.4    UDP        62 Source port: 53173  Destination port: 51510
```

Figure 5: This figure shows some of the packets that make up the exchange by starting a capture in Wireshark.

3.7 Protocol and packet encoding

The packet is treated as a byte array and contains two different parts: a TLV (Type length value) header and the payload (data).

3.7.1 TLV header

I included the key features of a TLV header and remodified as follow:

Maximum size of packet: 1500 bytes.

Packet design:

Part	Size(byte)	Description
Type	**1** byte	Type of packet. i.e.from controller or router
Length	**1** byte	Length of destination address
Value	Variable	Destination address
Payload	Variable	Payload of packet i.e. data

TLV header:

Byte	0	1	2-length+1
Bits	0-4	4-8	8-length*4+4
content	Packet Type	Length of des address	Destination address

3.7.2 Encoding

The way encoding works as follows: The data/payload in string format is first converted into a byte array. The TLV header is constructed by converting the destination address in byte array, then calculate the length of this byte array. The calculated length is then inserted before the destination address in byte array format and the type of packet is inserted before length. Finally, the entire TLV header is inserted before the payload's array to form the packet.

3.7.3 Decoding

The way decoding work is as follow: the length of the destination address is first recorded. By knowing the length, the packets can be split into parts and allows components to work on the part which its interested in. For example, if it wants to get the address, the packet could be spilt from 2 to length+1.

4 Evaluation and Discussion

The implementation is simplified without routing algorithm to get the optimised routes. Pushing notification from the central controller is not supported, which means the network change can not synchronize with the routers. The handshake between controller and routers are not implemented due to the time constraint.

To achieving scalability and flexibility, more complex routing protocol including extra cost related info should be added, and the sophisticated routing algorithm to support the calculation of the values to provide to the forwarding table.

5 Summary

In this report, it covers the detailed design and implementation for the problem statement and forwarding use cases. The design approach and considerations are also explained in detail and in-depth. Specifically, the detailed descriptions on the implementation are provided to facility to understand the solutions which have been done. The further evaluation and discussion are also suggested.

Reference

Kurose-Ross, 2017 Computer Networking. A Top-Down Approach, Seventh Edition, Pearson Publishing

Rajdeep Due, 2016 Learning Docker Networking, Packt publishing.

Stefan Weber, 2021 Trinity, CSU33031 slides.