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

CSU33031 Computer Networks Assignment #2: Flow Forwarding

Faith Olopade
November 25, 2023

1 Introduction

In this second assignment for CSU33031 Computer Networks, I embarked on a journey to deepen my understanding of network protocols, specifically focusing on the design and implementation of a flow forwarding mechanism. Building upon the foundational knowledge and skills I gained in Assignment 1, this task presented an opportunity to explore the complexities involved in managing packet flows within a network.

Objectives of the Assignment

The primary objective of this assignment was to develop a reactive routing approach for a network overlay. The challenge lay in creating a program that could efficiently establish routing information by broadcasting requests for destinations not already listed in the current routing table. This approach mirrors real-world scenarios in ad hoc networks, particularly useful in areas without traditional infrastructure, such as those affected by natural disasters.

Linking Assignment 1 and 2

My journey from Assignment 1 to this current assignment has been marked by an evolution in both complexity and application. While the first assignment focused on creating a publish/subscribe protocol for the distribution of audio, video, and text content, this assignment required a shift in focus towards network dynamics. The task at hand necessitated a deeper dive into network topologies, addressing schemes, and the nuances of reactive routing.

Importance of Network Protocols

Understanding and implementing network protocols is crucial in today's interconnected world. Protocols are the backbone of communication networks, dictating how data is transmitted and received across various devices and systems. The knowledge of creating and managing these protocols is invaluable, as it enables the creation of efficient, flexible, and fault-tolerant networks. This assignment served as a practical application of these concepts, allowing me to directly engage with the intricacies of network communication and problem-solving.

Personal Approach to the Assignment

In my solution, I utilised a combination of Docker and Python to simulate a realistic network environment. My protocol, designed to handle the dynamic flow of packets, comprised various network elements including multiple routers and endpoints. Through this, I aimed to create a system capable of efficiently managing packet routing, learning from traffic, and adapting to changes within the network. This hands-on experience provided me with a deeper appreciation of the role and complexity of network protocols in facilitating seamless communication.

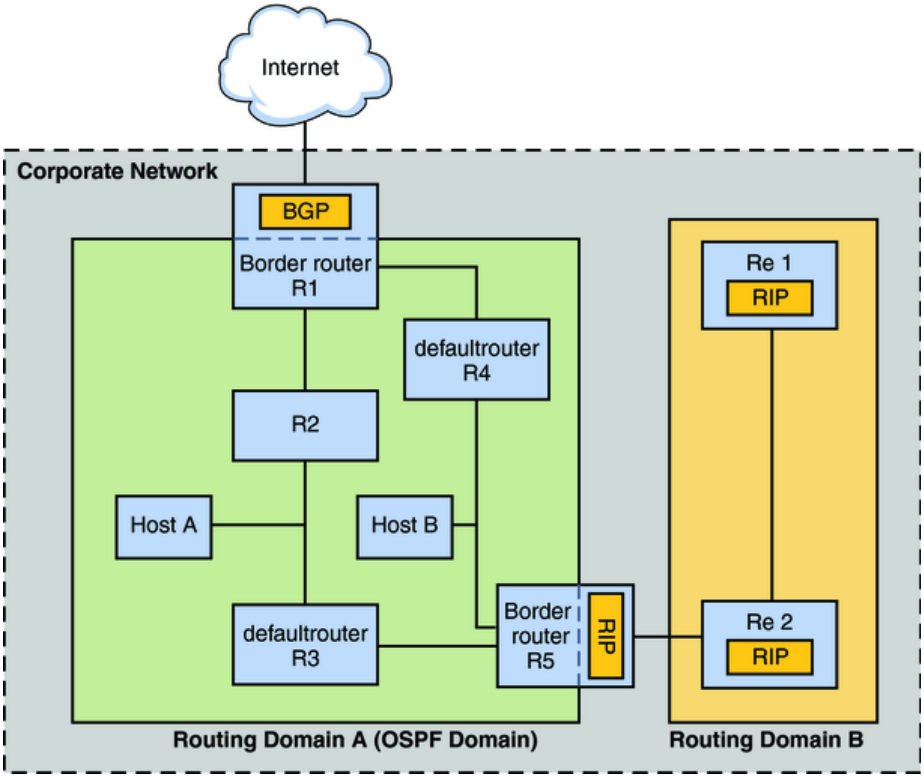

Figure 1: A high-level representation of the flow forwarding protocol. Packet forwarding and routing on IPv4 networks - system administration

2 Background

The development of this assignment's solution is deeply rooted in the technologies and frameworks that I previously encountered and those that I ventured into for the first time. My journey through this project has been shaped by an array of tools and concepts, with each playing a pivotal role in realising the final outcome.

2.1 Technical Background

Python and Docker: Python, a versatile and high-level programming language, served as the backbone of my implementation. Its simplicity and powerful libraries, especially for socket programming, were instrumental in developing the network communication components of my project. Docker, a platform that enables the creation and use of software containers, provided the perfect environment to simulate a network of multiple elements. Docker's ability to isolate and manage different components within containers facilitated a more realistic network testing and deployment scenario, allowing me to replicate complex network topologies and interactions.

UDP Sockets: UDP (User Datagram Protocol) sockets form the crux of the communication in my solution. Building on the knowledge from Assignment 1, where UDP's lightweight, connectionless nature was harnessed for a publish-subscribe model, I further explored its capabilities in reactive routing. UDP's efficiency in transmitting data without the overhead of establishing and maintaining a connection proved invaluable in quickly disseminating routing information and handling dynamic network configurations.

2.2 Reactive Routing and Its Application in Ad Hoc Networks

The concept of reactive routing, unlike other types of routing, involves creating routes on-demand. This approach is particularly beneficial in ad hoc networks, which are characterised by their spontaneous formation and lack of fixed infrastructure. In such networks, the ability to dynamically discover and maintain routes as network topology changes is critical. Reactive routing protocols, such as the one developed for this assignment, enable efficient data forwarding by broadcasting requests only when the route to a particular destination is unknown, thereby reducing unnecessary network traffic and adapting quickly to changes.

2.3 Integration of Previous Assignment Knowledge

The foundations laid in Assignment 1, particularly the understanding of network communication protocols and header design, significantly influenced my approach to this project. The ability to design efficient headers and understand their role in routing and forwarding mechanisms was directly applicable. The insights gained from handling UDP communication were expanded upon to manage the more intricate and dynamic routing requirements of this assignment. Additionally, the experience of working with Docker in creating isolated network environments was crucial in simulating the complex network topologies required for testing the reactive routing solution.

The technologies and concepts discussed here not only provided the tools necessary to build the solution but also shaped my understanding of network communications in dynamic environments. The integration of knowledge from the previous assignment with new learnings in this project exemplifies the continuous learning process in the field of computer networking. The subsequent sections will delve deeper into the design and implementation details, highlighting how these technical components were orchestrated to fulfil the assignment's objectives.

3 Problem Statement

This assignment presented a multifaceted challenge centred around the development and implementation of a reactive routing approach within a dynamic network environment. My task was to create a system capable of forwarding flows of packets with minimal communication overhead, while maintaining flexibility and fault tolerance in the network.

Challenges in Implementing Flow Forwarding:

The primary challenge was to design and implement a reactive routing mechanism that dynamically determines routes in the network. This approach required the network elements to broadcast requests for the location of destinations unknown in their current routing tables. The intricacy of this task was compounded by the requirement to work with a network topology comprising nodes with multiple interfaces and networks, which added layers of complexity to the routing logic.

Another significant challenge was ensuring the efficient establishment of routing information. The system had to be capable of rapidly adapting to changes in the network topology, such as the addition or removal of nodes, without compromising data transmission efficiency. Additionally, addressing exceptional cases like non-existent destinations or the handling of a high volume of packets during route establishment was critical to ensure robustness and reliability in the network's operation.

Objectives:

1. **Establishing Efficient Routing Information:** The core objective was to develop a routing mechanism capable of establishing efficient routing paths within a network. This involved creating a system where routers could dynamically discover routes to destinations, reducing the need for pre-established, static routing tables.

2. **Handling Exceptional Cases:** Addressing edge cases such as non-existent destinations in the network or coping with a sudden influx of packets was essential. The system needed to be resilient and capable of maintaining effective routing without significant performance degradation in such scenarios.

3. **Creating a Flexible and Fault-Tolerant Network:** Another key objective was to build a network that could adapt to changes and faults without significant disruptions. This involved developing routing mechanisms that could recalibrate and find alternate paths when usual routes were unavailable or compromised.

To achieve these objectives, my approach was to leverage the lessons learned from Assignment 1, particularly in the regards of UDP communication and Docker-based network simulation. The experience gained from implementing a publish-subscribe protocol based on UDP datagrams provided a solid foundation for tackling the challenges of reactive routing in a complex network topology. This assignment represented an opportunity to expand upon my existing knowledge and skills, applying them to a new context with its unique set of challenges and requirements.

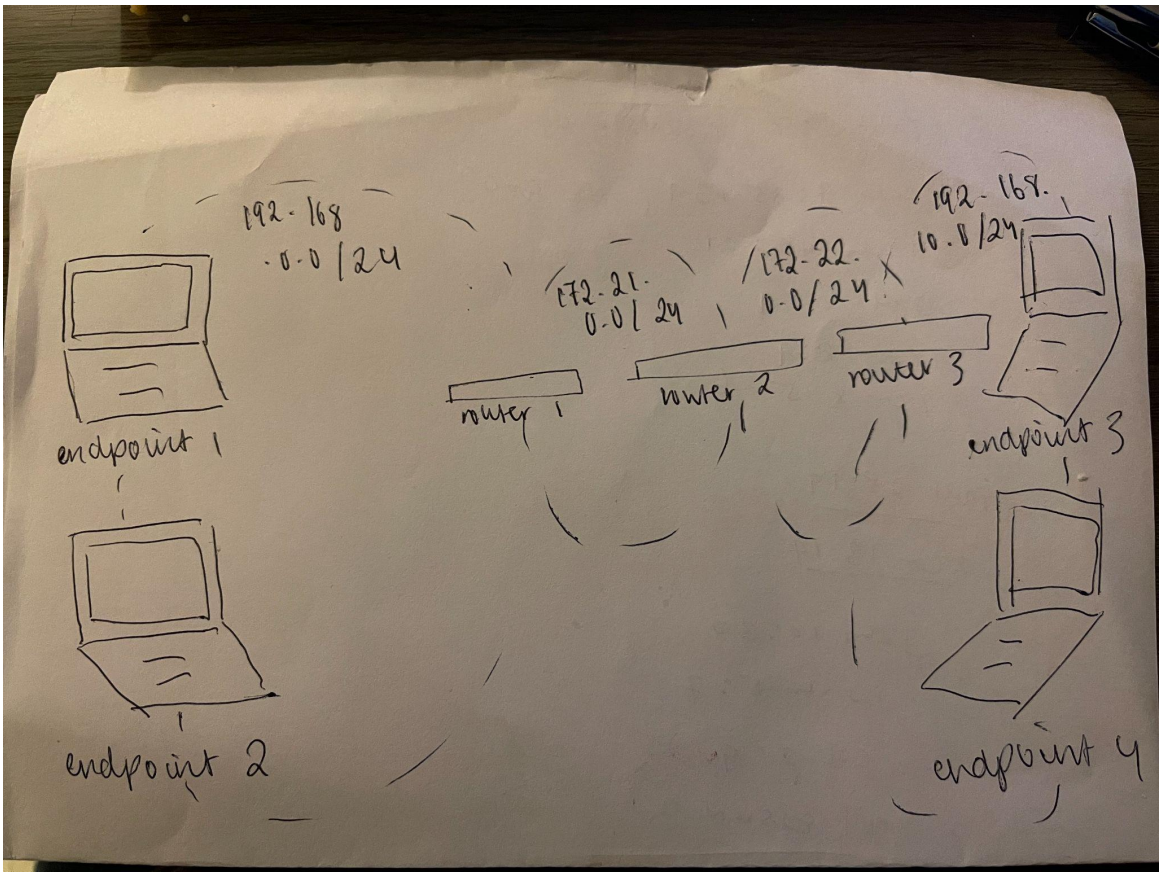

Figure 2: Architectural representation of the protocol's topology, depicting the interaction between Endpoints and Routers across distinct IP networks.

4 Design

In this section, I'll delve into the design aspects of my solution. This design is conceptualised with a focus on creating a reactive routing mechanism within a complex network topology, utilising Docker for virtual network infrastructure and Python for the programming environment.

4.1 Architecture Overview

The architecture of my solution is based on a distributed network of elements, including routers and endpoints, all functioning within a Docker-simulated environment. Each network element is assigned a unique address, not necessarily tied to IP addresses, and can be located anywhere in the topology. The primary focus is on establishing and managing flow forwarding paths dynamically within this network.

4.2 Reactive Routing Mechanism

The cornerstone of my design is the reactive routing mechanism. This approach involves network elements (routers) broadcasting requests to discover routes to destinations not already in their routing tables. When an endpoint initiates communication, the nearest router broadcasts a request for the destination's location. On receiving this broadcast, a router with information about the destination replies,

thus establishing a route. This mechanism is inherently dynamic, allowing for the adaptation to network changes such as the addition or removal of nodes.

4.3 Network Elements and Roles

Routers: Act as the primary agents in establishing routing paths. They broadcast requests for unknown destinations and learn about network topology through the traffic they handle. Each router maintains a forwarding table, dynamically updated based on the traffic and responses received.

Endpoints: These are the source and destination of the packets within the network. They interact with the routing mechanisms to send or receive data.

Docker Containers: Each network element, be it a router or an endpoint, is encapsulated within a Docker container, providing an isolated and controlled networking environment for simulation purposes.

4.4 Header Design in Packet Forwarding

The design of packet headers plays a crucial role in my solution. Each packet header contains essential information for routing decisions, such as the source and destination addresses (randomly assigned and unique within the network), and packet type indicators. This header information is crucial for routers to make forwarding decisions. The routers use this header data to update their forwarding tables, thus enabling the flow of packets back and forth between the original source and destination.

Figure 3: Packet Structure

4.5 Implementation in Python

Python's versatility and robustness in network programming are leveraged for implementing the routing mechanism and handling UDP communications. The design utilises Python's socket programming capabilities for sending and receiving broadcast messages and implementing the logic for reactive routing.

4.6 Use of Docker and Virtual Network Topology

Docker plays a significant role in simulating the network topology. Each router and endpoint is configured as a Docker container, allowing the creation of a virtual network with a defined topology. This setup enables the testing of the routing mechanism in a controlled environment, closely simulating real-world network scenarios.

In summary, my design is a well-orchestrated combination of reactive routing principles, efficient header design, and the use of Docker and Python to create a realistic and dynamic network environment. This design not only addresses the assignment's requirements but also provides a robust framework for understanding and implementing advanced network protocols in a virtualized environment.

5 Implementation

In this section, I detail the technical aspects of implementing the solution. This implementation involves developing a reactive routing mechanism using Python and Docker, focusing on the main components: routers and endpoints.

5.1 Python and Docker Environment

The entire solution is implemented in Python due to its robust support for network programming and ease of use. Python's socket library is crucial for handling UDP communications, vital for the reactive routing mechanism. For the virtual networking environment, Docker is employed. Docker containers simulate network elements (routers and endpoints), providing an isolated and controlled networking setup.

5.2 Router Implementation

The router's role is to establish and maintain the path for packet flows. Each router is implemented as a Python script running within a Docker container. The script binds a UDP socket to listen for incoming packets and employs a forwarding table to route the packets.

```
# Router.py snippet
import socket
import json
import constants as const

forwarding_table = {}
comms = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
comms.bind(("0.0.0.0", const.ROUTER_PORT))

while True:
    message, address = comms.recvfrom(65535)
    message_contents = json.loads(message.decode())
```

```
    # Routing logic
```

Listing 1: Router listens for incoming packets and uses the forwarding_table to decide the next hop based on the packet's destination.

5.3 Endpoint Implementation

Endpoints generate and send packets to routers. The endpoint script randomly generates a unique address for itself and communicates with the router using UDP sockets.

```
# Endpoint.py snippet
import socket
import random
import constants as const

def generate_address():
    return ':'.join(['%02x' % random.randint(0, 255) for _ in
range(4)])

endpoint_address = generate_address()
comms = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
router_addr = ("router1", const.ROUTER_PORT)

data_packet = {'source': endpoint_address, 'data': 'Hello World'}
comms.sendto(json.dumps(data_packet).encode(), router_addr)
```

Listing 2: Endpoint generates its address, creates a data packet, and sends it to the router.

5.4 Docker and Network Topology

Docker Compose is used to define the network topology, linking routers and endpoints. The compose.yaml file specifies the configuration of each Docker container representing network elements.

```
# Snippet from compose.yaml
services:
  router1:
    image: "pycimage"
```

```
    command: python3 router.py router1
    networks:
      - stub1
      - trans1
  endpoint1:
    image: "pycimage"
    command: python3 endpoint.py router1 1
    networks:
      - stub1
networks:
  stub1:
    ipam:
      config:
        - subnet: 192.168.0.0/24
```

Listing 3: Configuration defines the routers and endpoints as services, along with their network assignments.

5.5 Network Traffic Analysis

For monitoring and analysing the network traffic, tcpdump is used within Docker containers. This tool captures the packets flowing through the network, helping in understanding the behaviour of the implemented routing mechanism.

The implementation showcases the practical application of computer networking concepts, particularly reactive routing in ad hoc networks. Using Python and Docker, I created a realistic and functional network environment, with routers and endpoints efficiently managing the flow of packets. This setup not only meets the assignment's criteria but also serves as a foundational model for understanding dynamic routing mechanisms in computer networks.

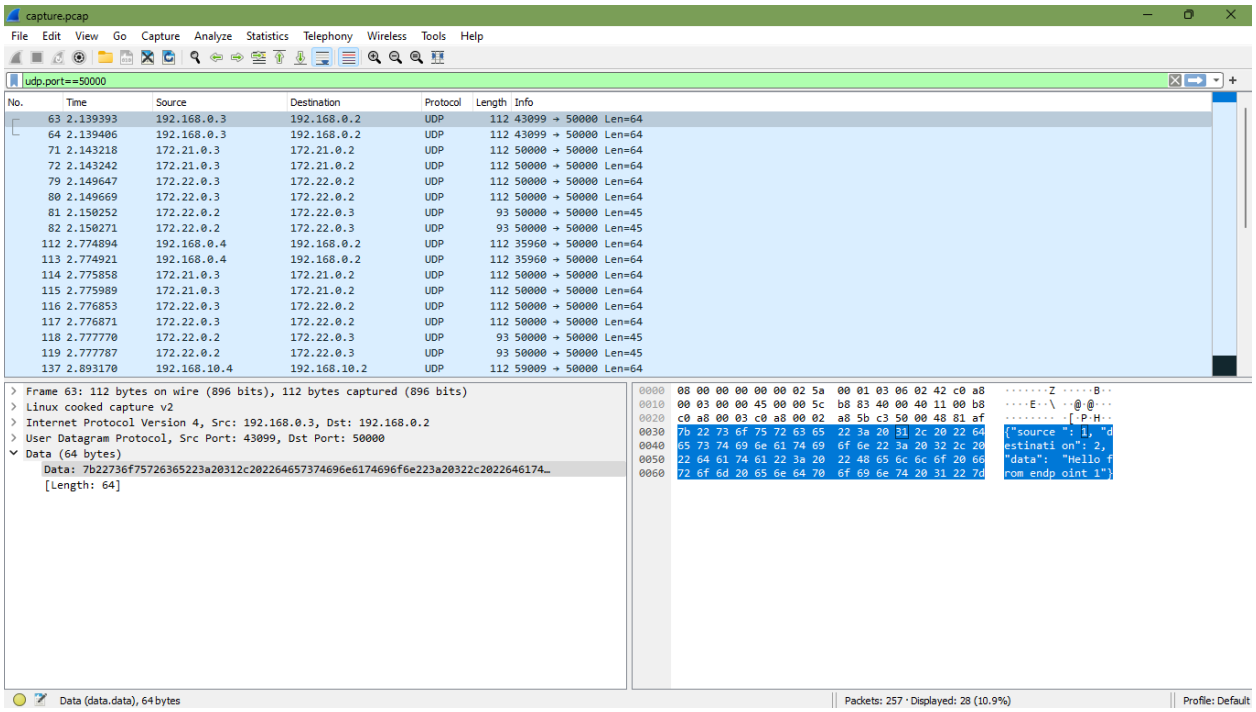

Figure 4: Wireshark Packet Analysis: Detailed Examination of UDP Packet Transmission and Content in the Network Simulation.

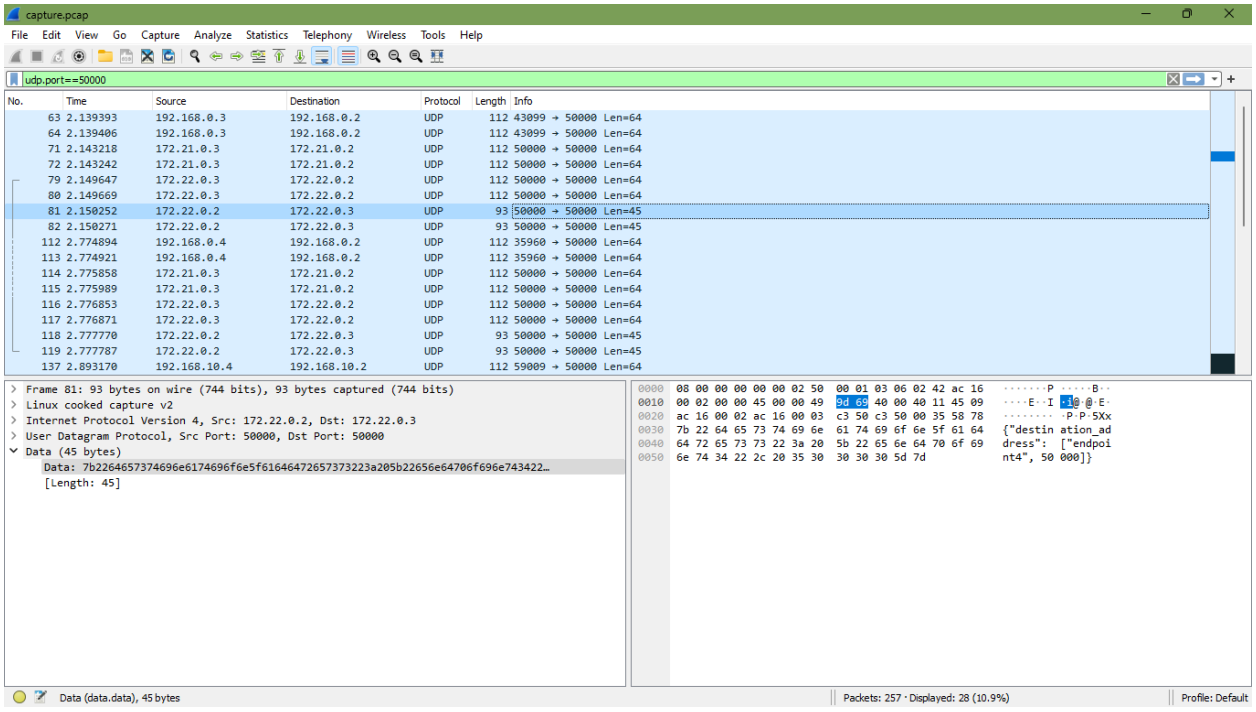

Figure 5: Wireshark Packet Analysis: Detailed Examination of UDP Packet Transmission and Content in the Network Simulation.

6 Discussion

Reflecting on the design and implementation of my reactive routing solution for the CSU33031 Computer Networks Flow Forwarding assignment, I delve into its effectiveness, comparative aspects, and the learnings gleaned from the process.

Effectiveness in Addressing the Problem:
The primary objective of the assignment was to develop a reactive routing approach for a network topology without hierarchical organisation. My solution, built on Python and Docker, effectively addressed this by establishing a dynamic routing mechanism that reacts to broadcast requests for unknown destinations. By integrating routers and endpoints within a Dockerized environment, the solution demonstrated practical application in ad hoc networks, particularly useful in scenarios like natural disasters where traditional infrastructures are absent.

Comparison with Existing Technologies:
Compared to established protocols like Ad hoc On-Demand Distance Vector (AODV) Routing, my approach offers a simplified yet effective mechanism for dynamic path establishment. While AODV provides a comprehensive solution for ad hoc routing with route discovery and maintenance strategies, my implementation focuses on a more basic yet agile setup, suitable for less complex network scenarios.

Advantages:

- **Flexibility and Scalability:** The ability to dynamically adjust to changing network topologies and to cater to different types of endpoints.
- **Minimal Overhead**: The use of lightweight Docker containers and Python scripts ensures a minimalistic yet functional setup.
- **Ease of Deployment:** The solution's modular nature makes it easily deployable in various network scenarios.

Limitations:

- **Lack of Sophistication:** The solution lacks the advanced features of more mature routing protocols, such as route maintenance and optimization.
- **Limited Fault Tolerance:** In its current form, the system does not robustly handle node failures or network partitioning.
- **Security Concerns:** The implementation does not inherently include security features, making it vulnerable in hostile environments.

Learnings from the Implementation Process:

- **Importance of Simplicity:** Keeping the design simple was essential for maintaining focus on the core objectives of the assignment.
- **Balancing Theory and Practice:** The project underscored the need to balance theoretical knowledge with practical implementation skills.
- **Adaptability:** I learned to adapt my approach based on the specific requirements of the network environment, such as using Docker for network element simulation.

Figure 6: Visualising the Packet Flow and Router Responses in the Network Simulation Environment.

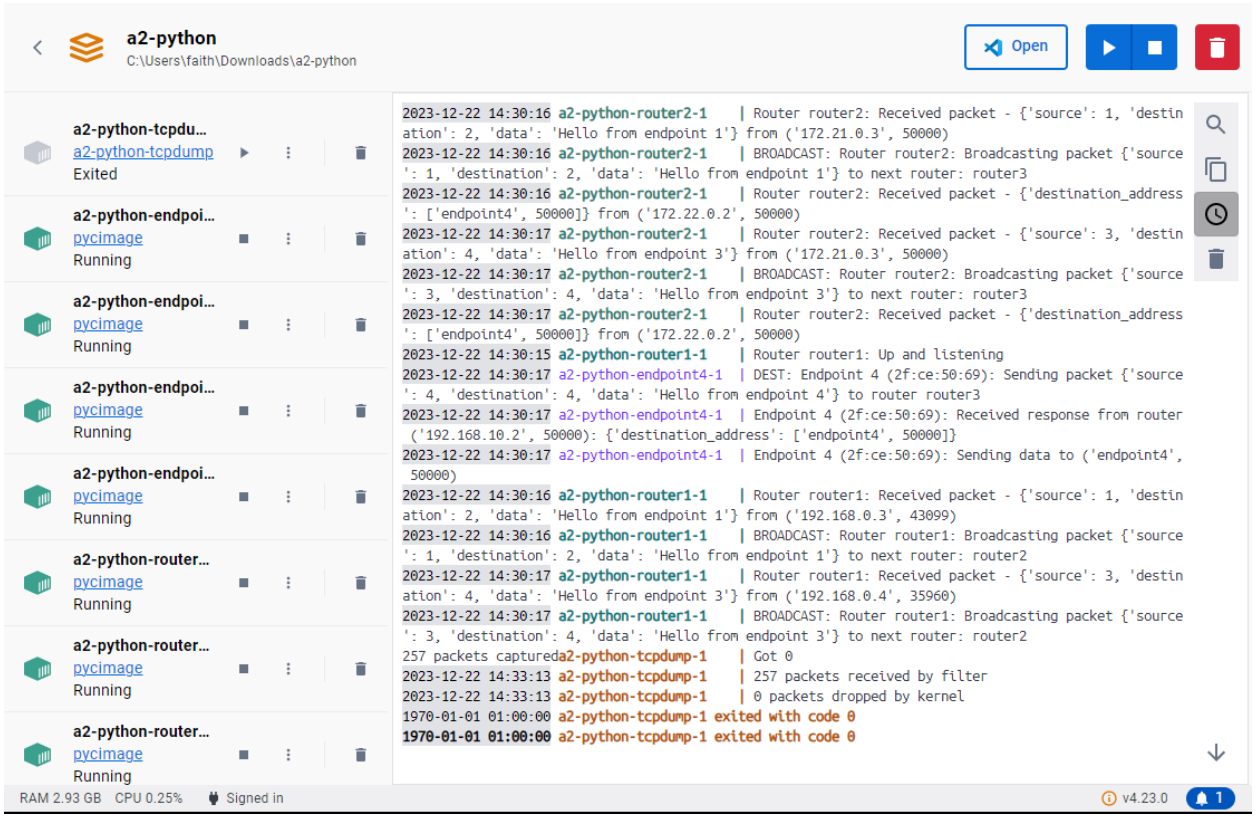

Figure 7: Visualising the Packet Flow and Router Responses in the Network Simulation Environment.

My solution for the assignment represents a balanced approach between theoretical concepts and practical application. While it may not rival the complexity and robustness of established routing protocols, it serves as a functional model for understanding the basics of reactive routing in ad hoc networks. Future enhancements could focus on integrating advanced routing features, improving fault tolerance, and incorporating security measures to elevate its applicability in more complex and diverse network environments.

7 Summary

Reflecting on the journey of this project for the CSU33031 Computer Networks Flow Forwarding assignment, it's clear that a significant milestone has been achieved in understanding and implementing a reactive routing mechanism in a simulated network environment.

Accomplishments:

- **Successful Implementation:** The key achievement was the successful design and implementation of a reactive routing protocol using Python and Docker, which was able to dynamically establish routing information in a non-hierarchical network topology.
- **Adaptable Routing Mechanism:** The solution effectively handled the broadcasting of requests for unknown destinations and the subsequent learning of routes by routers. This dynamic nature of the routing process was particularly apt for the simulated ad-hoc network scenarios.

- **Efficient Use of Technology:** Utilising Docker for network simulation provided a flexible and controlled environment to test the protocol. The choice of Python for scripting brought simplicity and efficiency to the implementation process.

Meeting the Objectives:
The core objectives of developing a reactive routing approach and minimising the communication needed for establishing forwarding information were met. The use of broadcast mechanisms for discovering routes and the integration of a forwarding table in routers showcased a practical approach to routing in ad-hoc networks.

Performance and Reliability:
While the implemented solution efficiently addressed the basic requirements of the assignment, there are areas of potential enhancement:

- **Scalability:** The solution works well in a controlled environment with a limited number of network elements. However, scalability to larger, more complex networks might require additional optimizations.
- **Fault Tolerance:** The current implementation has limited mechanisms to handle node failures or network partitions.
- **Security:** Absence of security features makes the protocol vulnerable in potentially hostile network environments.

Reflections:
This project has been an invaluable learning experience, emphasising the importance of practical application in understanding networking concepts. It has also highlighted the significance of simplicity in design and the need for a balance between robustness and efficiency in protocol development.

In summary, this project not only fulfilled its intended objectives but also provided insights into the complexities and challenges inherent in network protocol design and implementation, paving the way for further exploration and enhancement in this fascinating field of computer networks.

8 Reflection

Embarking on the assignment, I was presented with a unique blend of challenges and learning experiences that profoundly enhanced my understanding of computer networks and protocol design.

Personal Experience and Challenges:
- **Complexity and Intricacy:** One of the most striking aspects of this assignment was its complexity, particularly in implementing a reactive routing approach within a simulated network environment. It required a deep understanding of network protocols, routing mechanisms, and data packet management.
- **Learning Curve:** Grappling with Python for network programming, especially managing UDP sockets, was initially challenging. It demanded a steep learning curve, but it was gratifying to see my code successfully simulate the routing and forwarding of packets in the network.

Time Management and Effort Estimation:

Initially, I underestimated the time needed for this project. The complexities of designing and testing the routing mechanism were more time-consuming than anticipated. I spent over 40 hours in total, which included researching, coding, debugging, and optimising the code. This was a valuable lesson in effort estimation and time management for future assignments.

Areas of Improvement and Learning Outcomes:

- **Enhanced Network Understanding:** Through this assignment, I gained a much deeper understanding of network topologies, routing protocols, and the challenges in designing efficient, fault-tolerant systems.
- **Code Optimization:** I learned the importance of writing efficient, scalable code. Going forward, I would focus more on optimising my code for better performance, especially when handling larger network simulations.
- **Improved Debugging Skills:** Debugging network code was challenging, particularly when dealing with packet losses and route discovery in the network. This has significantly improved my debugging skills and patience.
- **Collaboration and Feedback:** Reflecting on the process, I realise the importance of peer collaboration and seeking feedback. For future projects, I plan to engage more with classmates for brainstorming and constructive feedback, which I believe will lead to more robust and innovative solutions.

In summary, this assignment was not only a test of technical skills but also an exercise in critical thinking, problem-solving, and perseverance. It underscored the importance of a methodical approach to complex problems and the need for continuous learning and adaptation in the ever-evolving field of computer networks.