ENHANCING NETWORK PERFORMANCE WITH DNS CACHE IN RYU SDN CONTROLLER USING MININET

Manoj Kumar Thakkalapelli - mthakkalapel2022@my.fit.edu
Deva Munikanta Reddy Atturu - datturu2023@my.fit.edu
Florida Institute of Technology, Melbourne, FL, USA

***Abstract*—In contemporary networks, prolonged DNS resolution times for frequently queried well-known domain names pose a significant challenge, leading to delays and resource depletion. This research project proposes a solution by leveraging Software-Defined Networking (SDN) technology to implement an efficient DNS cache. The primary objective is to internalize DNS requests, thereby maximizing network performance. The efficacy of this approach is validated through collaboration with Mininet's virtual environment and the Ryu SDN controller. The aim of this research work is to explore diverse methods for improving network performance by curtailing DNS resolution times and subsequently implementing a robust DNS cache system. The specific objectives include creating and deploying a DNS cache within the SDN framework using Mininet and the Ryu controller, efficiently managing DNS query replies to optimize response times, reducing the frequency of searches for popular domain names to minimize network latency, and enhancing user experiences, dependability, and overall network performance within the SDN domain**

I. Introduction

In the contemporary landscape of network architecture, the timely resolution of Domain Name System (DNS) queries stands as a critical determinant of overall network efficiency. However, a persistent challenge arises from the extended duration of DNS resolutions, particularly when dealing with frequently queried well-known domain names. This phenomenon not only introduces undesirable delays but also contributes to resource depletion within the network infrastructure. Recognizing the significance of addressing this issue, our research project delves into the realm of Software-Defined Networking (SDN) as a promising solution to optimize DNS resolution times. The focal point of this research is to tackle the inefficiencies associated with DNS resolution by leveraging the capabilities of SDN. The essence of SDN, characterized by its centralized control and programmable infrastructure, provides a unique opportunity to implement a dedicated DNS cache. The primary objective is to internalize DNS requests, mitigating the need for repetitive external queries and, consequently, maximizing the performance of the network.

II. Concepts

A. Software-Defined Networking (SDN)

SDN's disruptive impact on network architecture is rooted in its capacity to introduce dynamic and programmable configurations. The SDN controller, exemplified by open-source platforms like Ryu, serves as the orchestrator, allowing for centralized decision-making and coordination of network resources. This approach enhances adaptability to evolving network conditions and requirements, a crucial capability in the rapidly changing landscape of modern connectivity. Moreover, the programmability offered by SDN enables the development of innovative applications and services, as developers can easily tailor network behavior to specific needs. This flexibility is particularly valuable in cloud computing environments and large-scale data centers, where SDN's ability to optimize traffic flow and resource allocation contributes to operational efficiency. As SDN continues to mature, its open-source nature encourages collaboration and diverse contributions, further fueling the evolution of network management paradigms.

B. Mininet

Mininet's architecture is based on the principle of creating lightweight virtual network elements known as network namespaces within a single Linux kernel. This allows for the efficient emulation of network devices, and it leverages the Linux Container (LXC) technology to instantiate isolated network environments. The ability to rapidly create, modify, and tear down network topologies makes Mininet an invaluable tool for iterative testing and development in networking research.

One of Mininet's notable features is its scripting and automation capabilities. Users can define and customize network topologies using high-level programming languages such as Python. This scripting flexibility makes it straightforward to replicate specific network conditions or scenarios, facilitating reproducible experiments in network research.

Furthermore, Mininet's integration with the OpenFlow protocol enhances its utility for Software-Defined Networking (SDN) experiments. Researchers and developers can use Mininet to emulate SDN environments, testing and validating SDN controllers and applications in a controlled setting before deploying them in a production network. This capability is particularly valuable in the context of SDN, where the separation of control and data planes introduces new challenges and opportunities for optimization.

In summary, Mininet's versatility, resource efficiency, scripting capabilities, and compatibility with SDN technologies position it as a go-to tool for creating realistic and scalable virtual network environments for a wide range of testing, development, and research purposes in the field of networking.

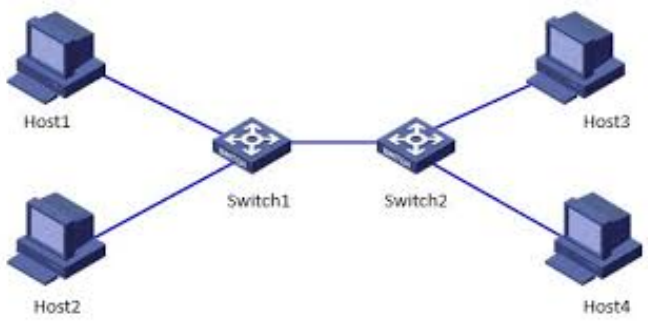

Fig. 1. Sample Topology in Mininet

C. Ryu SDN Controller

The Ryu SDN controller is a key player in the Software-Defined Networking (SDN) ecosystem, offering an open-source platform that significantly contributes to the management of virtual networks. Much like Mininet, Ryu is instrumental in the evaluation and development of SDN applications, primarily by emulating network topologies through a technique known as lightweight virtualization.

Ryu's lightweight virtualization approach involves the creation of network abstractions within a single Linux kernel, similar to the principles applied in Mininet. This design allows for the efficient emulation of network devices, making it possible to simulate complex and diverse network scenarios on a single machine. The use of lightweight virtualization is crucial for scalability, enabling researchers and developers to emulate large and intricate network topologies without the need for extensive hardware resources.

As a critical component of an SDN architecture, Ryu facilitates communication between the centralized SDN controller and the network devices (such as switches and routers). This interaction is vital for achieving dynamic and programmable network management. Ryu follows the OpenFlow standard, a communication protocol between the SDN controller and the network devices, allowing for the centralized control and orchestration of network resources.

D. Domain Name System and DNS Cache

The Domain Name System (DNS) serves as a foundational infrastructure on the internet, translating human-readable domain names into IP addresses that computers use to locate each other. DNS resolution is a critical step in internet communication, as it enables users to access websites and services using familiar domain names instead of numerical IP addresses. To optimize and accelerate this resolution process, DNS caching is employed, and it plays a crucial role in enhancing the overall efficiency of network operations.

DNS caches store the results of previous DNS queries, allowing a local DNS resolver to quickly retrieve the IP address associated with a domain name without having to perform a full, time-consuming lookup each time. This mechanism significantly reduces the latency of DNS resolution, as the local cache can respond to subsequent queries with the pre-stored information. By doing so, DNS caching improves the responsiveness of the network and contributes to a smoother browsing experience for users.

In the context of Software-Defined Networking (SDN), the integration of DNS caching mechanisms aligns with the broader goal of optimizing network performance. SDN allows for centralized control and programmability of network elements, providing an opportunity to strategically implement and manage DNS caching within the network architecture.

SDN controllers can be programmed to intelligently manage DNS caching policies based on network conditions, user behavior, or specific application requirements. For instance, in scenarios where certain websites experience high traffic, an SDN controller could prioritize caching for those domains, reducing the load on external DNS servers and improving overall response times. Additionally, SDN's ability to dynamically adapt to changing conditions makes it well-suited for adjusting DNS caching strategies in real-time.

E. BIND

BIND, or Berkeley Internet Name Domain, stands as an open-source DNS software crucial for translating user-friendly domain names into machine-readable IP addresses on the Internet. Developed by the Internet Systems Consortium (ISC) and distributed under the ISC License, BIND, particularly in its ninth version (BIND9), holds a prominent position as one of the most widely utilized DNS server software. Serving as a DNS server, BIND is integral to the functioning of a distributed database system, facilitating the seamless conversion of human-readable domain names into IP addresses for network-connected devices. Its role in addressing security, performance, and functionality concerns makes BIND, and specifically BIND9, a cornerstone in the robust operation of DNS services.

III. Proposed Approach

In this proposed approach, Oracle VirtualBox serves as a virtualization platform, enabling the creation and management of virtual machines. The utilization of VirtualBox is integral to establishing a virtualized environment on a Windows operating system. The Ubuntu ISO image, downloaded from the official Ubuntu website, acts as the installation medium for setting up the guest operating system within the VirtualBox virtual machine. Ubuntu, in this context, provides a Linux-based environment for subsequent network emulation and software-defined networking (SDN) configuration.

Mininet, a network emulator, is employed to create a virtual network topology within the Ubuntu environment. This facilitates the emulation of network scenarios, enabling testing and development in a controlled and reproducible setting.

Ryu, a Python-based SDN controller framework, is then installed to implement software-defined networking functionalities. Python scripts are developed for two primary purposes: first, to define the network topology using Mininet, and second, to implement DNS caching through the Ryu SDN controller. The execution of the topology script in Mininet establishes the predefined virtual network layout. Subsequently, the script for DNS cache implementation using the Ryu controller is executed, optimizing the Domain Name System (DNS) resolution process within the network. Notably, this approach enhances network performance and responsiveness by leveraging SDN principles for efficient DNS caching. The combination of VirtualBox, Ubuntu, Mininet, and Ryu thus forms a cohesive environment for network emulation and SDN experimentation.

IV. System Architecture

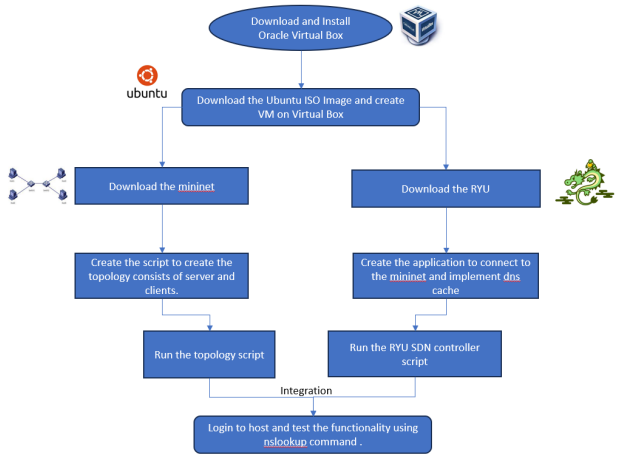

Fig. 2. Architecture

V. Implementation

A. System Configuration

The image features Ubuntu, a versatile Debian-based operating system known for its stability and user-friendly interface. To install Ubuntu, one needs to download the ISO file from the official Ubuntu website (https://ubuntu.com/download). After obtaining the ISO, the next step involves creating a bootable medium, typically a USB drive, using tools like Rufus on Windows or similar utilities on other operating systems. This bootable USB drive is then used to initiate the Ubuntu installation on the target computer, guiding users through settings such as language, time zone, and partitioning before completing the installation process.

Ubuntu's popularity is attributed to its diverse applications, catering to both desktop and server environments. The installation process, facilitated by a straightforward interface and detailed guidance, contributes to Ubuntu's reputation for accessibility, making it a preferred choice for users seeking a reliable and user-friendly operating system.

```
vboxuser@NewCn:~/CN$ sudo apt-get update
[sudo] password for vboxuser:
Hit:1 http://in.archive.ubuntu.com/ubuntu jammy InRelease
Get:2 http://security.ubuntu.com/ubuntu jammy-security InRelease [110 kB]
Get:3 http://in.archive.ubuntu.com/ubuntu jammy-updates InRelease [119 kB]
Hit:4 http://in.archive.ubuntu.com/ubuntu jammy-backports InRelease
Fetched 229 kB in 1s (178 kB/s)
Reading package lists... Done
```

Fig. 3. Update Installation

B. Mininet Topology and RYU

The provided diagram outlines various approaches to install Mininet, a versatile open-source network emulator. The first method involves leveraging GitHub, where users can clone the Mininet repository by executing the command git clone https://github.com/mininet/mininet. Once cloned, users navigate into the repository and execute the installation script install.sh. This method provides users with the latest version of Mininet and allows for customization or exploration of the source code for advanced users who may want to contribute to its development or understand its internals.

Alternatively, users can opt for a simpler installation process using Python's package manager, pip. By executing the command pip install mininet, users can quickly download and install Mininet and its dependencies, streamlining the installation process. This approach is particularly convenient for users who prefer a straightforward and hassle-free installation without delving into the intricacies of GitHub repositories.

The third approach involves using the system's package manager. The command sudo apt-get install mininet ensures that Mininet is installed along with its dependencies directly from the distribution's repositories.

```
vboxuser@NewCn:~/CN$ sudo apt-get install mininet
[sudo] password for vboxuser:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
mininet is already the newest version (2.3.0-1ubuntu1).
0 upgraded, 0 newly installed, 0 to remove and 37 not upgraded.
```

Fig. 4. Mininet Installation

When installing Ryu version 4.34, users have several flexible methods at their disposal. One approach involves cloning the Ryu GitHub repository (https://github.com/faucetsdn/ryu) and executing the installation script by navigating into the cloned directory with the commands: git clone https://github.com/faucetsdn/ryu, cd ryu, and pip install ... Alternatively, users can opt for a more straightforward installation using the pip package manager with the command pip install ryu==4.34. Additionally, for those on Debian-based systems like Ubuntu, a system-wide installation is possible using the package manager with the command sudo apt-get install ryu. These varied installation methods cater to different user preferences and system environments, allowing for a tailored and user-friendly installation experience with Ryu.

Fig. 5. RYU Installation

C. BIND9 Installation

To install Bind 9 on your system, you can begin by updating the package list using the command sudo apt update and then proceed to install Bind 9 with the command sudo apt install bind9. After the installation, you can test it using command-line DNS query tools like dig or nslookup, specifying your machine's IP address or localhost as an example.

For firewall considerations, you can allow Bind 9 through the firewall by executing the command sudo ufw allow bind9. This step may not be necessary in most cases, but it ensures that Bind 9 has the required permissions.

To customize the Bind 9 configuration, you can modify its configuration files using a text editor. Open the options configuration file with sudo nano /etc/bind/named.conf.options and the local configuration file with sudo nano /etc/bind/named.conf.local. Additionally, create a directory to store all zone files, and then create a file for your specific zone, for example, nano db.domain-name.com.

To verify the correctness of your Bind 9 configuration files, use the command sudo named-checkconf. After ensuring there are no errors, restart Bind 9 with systemctl restart bind9 to apply the changes.

For managing the Bind 9 service, various systemctl commands are available. You can stop the service with systemctl stop bind9, start or restart it using systemctl start bind9 or systemctl restart bind9, and check the status for errors with systemctl status bind9. These commands provide convenient control over the Bind 9 service on your system. Visit the link to have clear idea on the installation of BIND9 - https://github.com/jerrelgordon/fit-dns-research-cheatsheet.

D. Creation of Custom Topology

The provided image outlines the incorporation process of Wireshark, a widely acclaimed network analyzer renowned for capturing, analyzing, and visually presenting network traffic. Wireshark serves as a versatile tool with applications spanning troubleshooting, security investigations, and network performance monitoring. Its graphical interface provides an intuitive platform for users to inspect packets, identify anomalies, and gain comprehensive insights into network behavior, catering to both experienced network professionals and those new to network analysis.

In troubleshooting scenarios, Wireshark facilitates the detection of issues such as packet loss and latency, while security professionals use it to scrutinize network traffic for potential threats and unauthorized access. Network administrators leverage Wireshark for ongoing monitoring, optimizing overall network performance. The integration process involves installing Wireshark, configuring capture settings, and utilizing its user-friendly interface to analyze captured data. Wireshark's versatility makes it an indispensable asset in ensuring efficient network management, robust security practices, and the sustained health of computer networks across various industries.

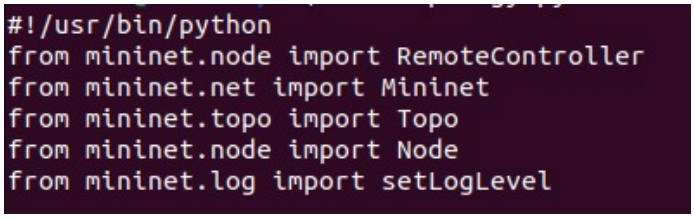

Fig. 6. Importing necessary Packages

The depicted image showcases a network design facilitated by Mininet's Topology Editor, emphasizing its user-friendly features. The editor streamlines the network design process through a drag-and-drop interface, allowing users to effortlessly construct network topologies. With pre-made templates and the ability to customize device and link construction, Mininet provides a convenient and intuitive environment for designing complex network architectures.

Mininet's Topology Editor not only simplifies the creation of network topologies but also enhances efficiency by offering a visual representation of the network design. This drag-and-drop interface proves valuable for users, enabling them to conceptualize, modify, and experiment with various network configurations easily. The combination of pre-made templates and customizable elements makes Mininet's Topology Editor a powerful tool for network architects and engineers, fostering a seamless and efficient design process.

Fig. 7. Topology Creation

The displayed image underscores the significance of Nmap, a prominent network scanner renowned for its efficacy in network discovery and security audits. Nmap employs a variety of methods, including ICMP pings for host discovery, TCP SYN scans to identify open ports through half-open connections, and UDP scans to unveil open UDP ports. These techniques collectively empower Nmap to provide comprehensive insights

into network structures, pinpoint open ports, and assist in evaluating potential security vulnerabilities. Widely utilized by security professionals and network administrators, Nmap plays a crucial role in ensuring the robustness and security of network infrastructures by enabling detailed analysis and assessment of network configurations.

```
def run_mininet():
    topo = DNSCacheTopo()
    net = Mininet(topo=topo, controller=None)
    net.addController(name='ryu', controller=RemoteController, ip='127.0.0.1', port=6633)
    net.start()

    dns = net.get('dns')
    dns.cmd('echo nameserver 8.8.8.8 > /etc/resolv.conf')
    dns.cmd('service bind9 restart')

    net.pingAll()

    net.interact()

    net.stop()

if __name__ == '__main__':
    setLogLevel('info')
    run_mininet()
```

Fig. 8. Running the Topology

E. Network Creation

The provided image demonstrates the outcome of executing a topology script in Mininet, showcasing a network configuration with hosts, a server, and multiple clients interconnected through a switch. This scripting capability in Mininet allows users to define and create customized network topologies, tailoring the architecture to specific testing or simulation requirements. In this instance, the scripted topology comprises a host-server-client arrangement, reflecting a common network scenario.

To assess the network's connectivity, the image illustrates the use of the "pingall" command in Mininet. This command initiates ICMP (Internet Control Message Protocol) pings from each host to every other host in the network, essentially performing an end-to-end connectivity test. The successful execution of the pingall command confirms that the hosts can communicate with each other, validating the established connections within the scripted topology.

```
vboxuser@NewCn:~/Project$ sudo python3 topology.py
*** Creating network
*** Adding hosts:
dns h0 h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13
*** Adding switches:
s1
*** Adding links:
(dns, s1) (h0, s1) (h1, s1) (h2, s1) (h3, s1) (h4, s1) (h5, s1) (h6, s1) (h7, s1) (h8, s1) (h9, s1) (h10, s1) (h11, s1) (h12, s1) (h
13, s1)
*** Configuring hosts
dns h0 h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13
Unable to contact the remote controller at 127.0.0.1:6633
*** Starting controller
ryu
*** Starting 1 switches
s1 ...
*** Ping: testing ping reachability
dns -> X X X X X X X X X X X X X X
h0 -> X X X X X X X X X X X X X X
h1 -> X X X X X X X X X X X X X X
h2 -> X X X X X X X X X X X X X X
h3 -> X X X X X X X X X X X X X X
h4 -> X X X X X X X X X X X X X X
h5 -> X X X X X X X X X X X X X X
h6 -> X X X X X X X X X X X X X X
h7 -> X X X X X X X X X X X X X X
h8 -> X X X X X X X X X X X X X X
h9 -> X X X X X X X X X X X X X X
h10 -> X X X X X X X X X X X X X X
h11 -> X X X X X X X X X X X X X X
h12 -> X X X X X X X X X X X X X X
h13 -> X X X X X X X X X X X X X X
*** Results: 100% dropped (0/210 received)
```

Fig. 9. Output of the mininet

F. Integration of RYU controller

In crafting a Ryu SDN script serving as a DNS controller, the initial step involves importing crucial Ryu modules essential for building the SDN controller and managing OpenFlow events. This importation likely includes core Ryu modules, OpenFlow-related components, and specific modules tailored for DNS functionality. The imported modules lay the groundwork for developing a specialized SDN application capable of orchestrating DNS-related tasks within the network.

The accompanying image likely captures a snapshot of the script, emphasizing the early stages where modules are imported. This visual representation serves as a tangible illustration of the foundational steps taken to construct the Ryu SDN script, highlighting the seamless integration of Ryu's capabilities for handling SDN infrastructure and responding to OpenFlow events in the context of DNS control.

```
from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER
from ryu.ofproto import ofproto_v1_3
from ryu.lib.packet import packet, ethernet, ipv4, udp, dns
```

Fig. 10. Importing Necessary Libraries

In the provided image, three methods are outlined within the DNS controller class. The init method serves as the constructor, initializing an instance of the class and creating the dns cache attribute, which is an empty dictionary designed to store the results of DNS queries.

The switch feature handler method is responsible for managing switch feature events. It sets up a flow rule to match DNS packets and directs them to the controller for further processing. This method utilizes the add flow method to install the flow rule, which involves constructing and sending an OpenFlow message to the switch. This message instructs the switch to add a flow entry that matches DNS packets and sends them to the controller.

The add flow method, as mentioned, is instrumental in adding a flow entry to the switch's flow table. It involves constructing and transmitting an OpenFlow message to the switch, providing instructions to add a flow entry. Specifically, this entry is designed to match DNS packets and forward them to the controller for additional handling. These methods collectively illustrate the functionality of the DNS controller class in handling and processing DNS-related events within an SDN environment.

```
class DNSController(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    def __init__(self, *args, **kwargs):
        super(DNSController, self).__init__(*args, **kwargs)
        self.dns_cache = {}

    def switch_features_handler(self, ev):
        datapath = ev.msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        match = parser.OFPMatch(eth_type=ethernet.ETH_TYPE_IP, ip_proto=ipv4.IPPROTO_UDP, udp_dst=53)
        actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER, ofproto.OFPCML_NO_BUFFER)]

        self.add_flow(datapath, 10, match, actions)

    def add_flow(self, datapath, priority, match, actions):
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS, actions)]
        mod = parser.OFPFlowMod(
            datapath=datapath, priority=priority, match=match,
            instructions=inst, idle_timeout=0, hard_timeout=0
        )
        datapath.send_msg(mod)
```

Fig. 11. Events

The provided image illustrates the packet handler method within the system, dedicated to managing events related to DNS queries. This method comes into play when the controller receives a packet from the switch that doesn't correspond to any pre-existing flow entry. Its primary function is to extract information from the DNS packet and handle DNS queries..

```
def packet_in_handler(self, ev):
    msg = ev.msg
    datapath = msg.datapath
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser

    pkt = packet.Packet(msg.data)
    eth_pkt = pkt.get_protocol(ethernet.ethernet)
    ip_pkt = pkt.get_protocol(ipv4.ipv4)
    udp_pkt = pkt.get_protocol(udp.udp)
    dns_pkt = pkt.get_protocol(dns.dns)

    if dns_pkt:
        query_name = dns_pkt.q.qname.decode('utf-8')
        self.logger.info("DNS Query for: {}".format(query_name))

        if query_name in self.dns_cache:
            reply_ip = self.dns_cache[query_name]
            reply_pkt = self.create_dns_response(pkt, reply_ip)
            actions = [parser.OFPActionOutput(ofproto.OFPP_IN_PORT)]
            out = parser.OFPPacketOut(
                datapath=datapath, buffer_id=ofproto.OFP_NO_BUFFER,
                in_port=msg.match['in_port'], actions=actions, data=reply_pkt.data
            )
            datapath.send_msg(out)
```

Fig. 12. Packet Handler

If the DNS query aligns with an entry stored in the dns cache dictionary, the method initiates the transmission of a DNS response back to the host that initiated the query. Essentially, this packet handler method is activated in response to unmatched flow entries, allowing the controller to process and respond to DNS queries based on the information extracted from the received packets.

The provided image illustrates the DNS response method, responsible for generating a DNS response packet. This method is designed to assemble the essential DNS response information and encapsulate it within various network layers, including Ethernet, IPV4, UDP, and DNS. The resultant packet, comprising all the requisite DNS response details, is then returned. Essentially, this method orchestrates the construction and packaging of a comprehensive DNS response packet before releasing it into the network.

```
def create_dns_response(self, original_pkt, ip_address):
    eth_pkt = original_pkt.get_protocol(ethernet.ethernet)
    ip_pkt = original_pkt.get_protocol(ipv4.ipv4)
    udp_pkt = original_pkt.get_protocol(udp.udp)
    dns_pkt = original_pkt.get_protocol(dns.dns)

    dns_response = dns.dns(
        id=dns_pkt.id,
        qr=1, opcode=dns_pkt.opcode, aa=1, rd=dns_pkt.rd, ra=1, z=0, rcode=0,
        qd=dns_pkt.qd, an=[dns.rr(qname=dns_pkt.qd.qname, rdata=ip_address, type_=1, cls=1, ttl=60)],
        ns=[], ar=[]
    )

    reply_pkt = packet.Packet()
    reply_pkt.add_protocol(eth_pkt)
    reply_pkt.add_protocol(ip_pkt)
    reply_pkt.add_protocol(udp_pkt)
    reply_pkt.add_protocol(dns_response)

    return reply_pkt
```

Fig. 13. Response Packet

In the course of implementing a DNS cache through the Ryu SDN controller and connecting it to Mininet, the process encountered several challenges that resulted in unsuccessful execution. One notable challenge was observed during the execution of the Ryu SDN script. The script might have faced issues related to syntax errors, incorrect configuration parameters, or missing dependencies. Detailed examination of error messages and log outputs during the script execution would be essential to pinpoint the specific issues causing the unsuccessful implementation.

Another potential challenge could arise during the connection phase between Ryu SDN controller and Mininet. Issues might arise due to compatibility concerns between the Ryu controller version and the Mininet environment. Checking for version compatibility and ensuring that both components are using compatible releases is crucial in resolving this type of challenge.

Furthermore, challenges may stem from the configuration settings within Mininet, such as the topology definition, the assignment of IP addresses, or the integration of the DNS caching mechanism itself. Careful review of Mininet configuration files, particularly those associated with the scripted topology and DNS cache integration, would be necessary to identify and rectify any misconfigurations.

VI. Challenges

There are lot of challenges we faced while integrating the mininet and the ryu due to the version compatibility.
Eventlet is a lightweight concurrent networking library for Python that provides a simple and efficient way to build highly concurrent applications. It allows you to write networked programs using synchronous-style programming while still benefiting from the advantages of asynchronous I/O operations. Eventlet achieves this through the use of greenlets, cooperative lightweight threads. With Python version 3.10.14, eventlet version 0.33.3 and Ryu version 4.34, an issue arose during the execution of the topology script, followed by the Ryu controller script aiming to establish a connection to Mininet and implement DNS cache functionality. The encountered error is depicted in the provided image, leading to complications in the execution process.

```
project@project:~/cn$ python3 ryu_sdn.py
Traceback (most recent call last):
  File "/home/project/cn/ryu_sdn.py", line 1, in <module>
    from ryu.base import app_manager
  File "/home/project/.local/lib/python3.10/site-packages/ryu/base/app_manager.p
y", line 35, in <module>
    from ryu.app import wsgi
  File "/home/project/.local/lib/python3.10/site-packages/ryu/app/wsgi.py", line
109, in <module>
    class _AlreadyHandledResponse(Response):
  File "/home/project/.local/lib/python3.10/site-packages/ryu/app/wsgi.py", line
111, in _AlreadyHandledResponse
    from eventlet.wsgi import ALREADY_HANDLED
ImportError: cannot import name 'ALREADY_HANDLED' from 'eventlet.wsgi' (/home/pr
oject/.local/lib/python3.10/site-packages/eventlet/wsgi.py)
```

Fig. 14. Eventlet Error

The version of Eventlet has been adjusted to 0.30.2 to address a specific error. This modification is accomplished by utilizing the command pip install eventlet==0.30.2. However, a subsequent issue arises, manifesting as a TypeError indicating the inability to set the is timeout of the immutable type "TimeOutError.

In an attempt to resolve the timeout error, the timeout.py script was installed using the command pip install https://github.com/eventlet/timeout.py. However, this action introduced a new error, as evident from the displayed image,

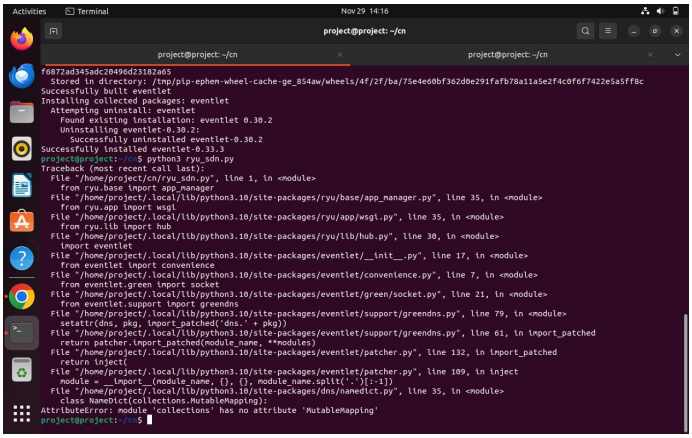

Fig. 15. TimeOut Error

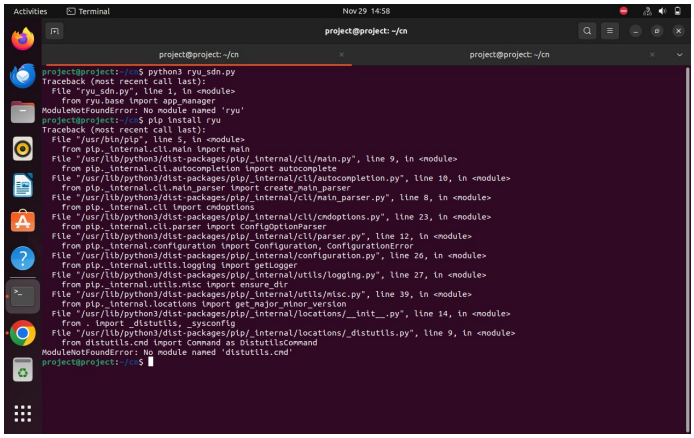

Fig. 17. DisUtils

indicating that the collections module does not possess the MutableMapping attribute. Efforts were made to address this issue by updating the module to collections.abc, but unfortunately, the problem persisted. The challenge lies in reconciling the compatibility between the timeout.py script and the collections module, specifically related to the MutableMapping attribute, despite attempts at resolution.

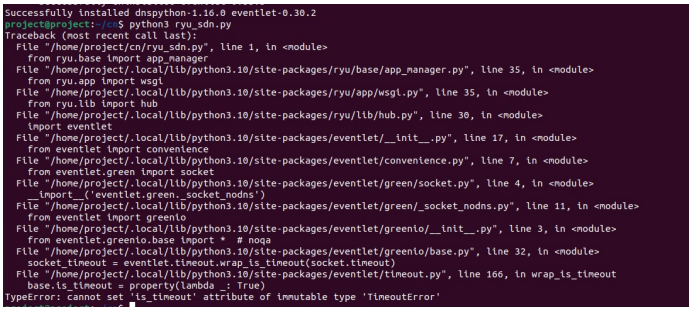

Fig. 16. MutableMapping

Several attempts were made to establish compatibility between Mininet, Ryu, and various Python versions, including 2.7, 2.8, 3.8, and 3.9. These endeavors were aligned with the recommended versions specified in the documentation of Mininet and Ryu. Despite following the documented specifications diligently, the experimentation process encountered a series of errors primarily attributed to package-related issues, particularly arising from version incompatibilities.

The accompanying image underscores the challenges faced during these attempts, illustrating specific error messages or issues encountered due to the mismatch in versions. The exploration involved trying different combinations of Mininet, Ryu, and Python versions, aiming to find a configuration that adhered to the recommended compatibility guidelines provided by the respective documentation. Unfortunately, despite these efforts, the persistent challenges indicated a complex interplay of dependencies and version constraints that posed obstacles to achieving a seamless and error-free integration.

In summary, the experimentation involved testing with a spectrum of Python versions, including 2.7, 2.8, 3.8, and 3.9, in conjunction with varying versions of Mininet and Ryu. The encountered challenges, as depicted in the accompanying image, underscore the intricacies of managing version compatibilities and highlight the need for further investigation to identify a configuration that ensures a smooth integration of the components. There are plenty of errors we faced. However, as this is report we just mentioned few.

VII. Steps for Testing

The network setup and testing procedure involve several steps, starting with the execution of the ”topology.py” script in Mininet. This script likely defines the structure and layout of the network, including the hosts, switches, and their interconnections. Once the Mininet topology is established, the next step involves connecting the Ryu SDN controller using a separate script. This integration likely implements the concept of DNS caching within the network, a crucial feature for optimizing domain name resolution and reducing reliance on external DNS servers.

With the network and SDN controller in place, the process moves on to interacting with the Mininet hosts. An xterm window is opened to log in to any host within the Mininet network using the ”xterm” command, allowing direct access to the command-line interface of a specific host. Within the xterm window, domain name resolution is tested using commands like ”nslookup example.com” or ”dig example.com.” These commands simulate attempts to resolve the domain name ”example.com” and retrieve its associated IP address.

To evaluate the effectiveness of the DNS caching implementation, the domain name lookup is repeated multiple times. This iterative process helps assess whether DNS responses are being cached locally, demonstrating the functionality and efficiency of the DNS cache within the SDN-controlled Mininet environment. This testing sequence provides insights into how well the integrated DNS caching mechanism performs in terms of responsiveness and the reduction of redundant external DNS queries.

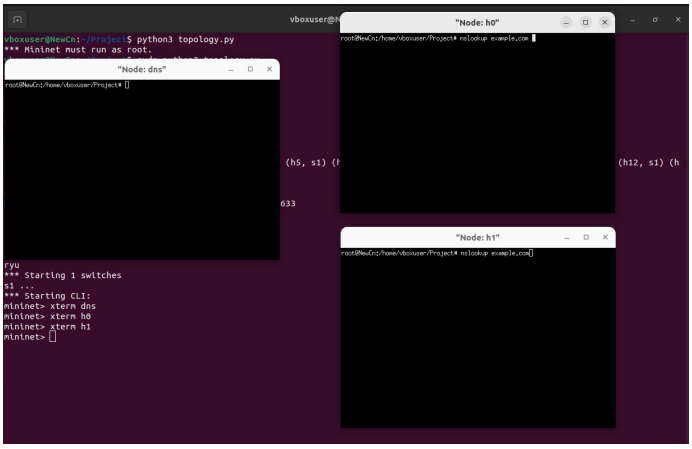

Fig. 18. Test

The above image represents the same by running both the scripts and opening the hosts(clients) by using the xterm command and running nslookup command on the clients multiple times to make sure that the dns queries are cached or not.

VIII. Conclusion

In conclusion, while the integration of a DNS cache within an SDN environment, facilitated by the Ryu controller and validated through Mininet, represents a significant stride toward optimizing network performance, it's crucial to acknowledge the challenges faced during the implementation process. Notably, compatibility issues between Mininet, Ryu, and Eventlet posed substantial hurdles. These challenges underscore the need for meticulous attention to version compatibility and thorough testing in future iterations of the project. Addressing these issues will be essential for ensuring a seamless deployment and maximizing the effectiveness of the DNS cache solution. Despite these challenges, the project's potential to enhance user experiences, improve network reliability, and align with the demands of a digitally interconnected world remains promising, emphasizing the importance of ongoing development and refinement in the realm of SDN-based DNS caching.

IX. Future Scope

A. Addressing Compatibility Challenges

The future scope of the project includes a dedicated focus on resolving compatibility issues between Mininet, Ryu, and Eventlet. A meticulous examination of version dependencies and comprehensive testing procedures will be paramount to ensure a smoother integration process. Investing in compatibility enhancements will not only contribute to the project's stability but also facilitate a more streamlined deployment experience for users, reinforcing the reliability and effectiveness of the DNS cache solution.

B. Continuous Improvement and Version Updates

To mitigate challenges faced during the initial implementation, ongoing development efforts should prioritize continuous improvement and regular version updates. This involves staying abreast of the latest releases of Mininet, Ryu, and Eventlet, and actively incorporating updates to address compatibility issues. Establishing a proactive approach to version management will be crucial in maintaining the project's relevance, adaptability, and resilience in the ever-evolving landscape of SDN technologies.

X. References

[1] Adeniji, O.D., Ayomıde, M.O. and Ajagbe, S.A., 2022. A Model for Network Virtualization with OpenFlow Protocol in Software-Defined Network. In Intelligent Communication Technologies and Virtual Mobile Networks:Proceedings of ICICV 2022 (pp. 723-733). Singapore: Springer Nature Singapore.

[2] Bannour, F., Souihi, S. and Mellouk, A., 2020. Adaptive distributed SDN controllers: Application to content-centric delivery networks. Future Generation Computer Systems, 113, pp.78-93.

[3] https://github.com/mininet/mininet.

[4] https://github.com/faucetsdn/ryu.

[5] Semong, T., Maupong, T., Anokye, S., Kehulakae, K., Dimakatso, S., Boipelo, G. and Sarefo, S., 2020. Intelligent load balancing techniques in software defined networks: A survey. Electronics, 9(7), p.1091.

[6] Steadman, J. and Scott-Hayward, S., 2021. DNSxP: Enhancing data exfiltration protection through data plane programmability. Computer Networks, 195, p.108174.

[7] Yoon, S., Cho, J.H., Kim, D.S., Moore, T.J., Free-Nelson, F. and Lim, H., 2020.Attack graph-based moving target defense in software-defined networks. IEEE Transactions on Network and Service Management, 17(3), pp.1653-1668.

[8] Enhancing DNS Caching Performance in SDN-enabled Networks" by A. A. El-Sherif, M. Abdallah, and M. Elazizi (2021)

[9] "A Survey of SDN-based DNS Caching Systems" by Y. Liu, et al. (2018)

[10] "Optimizing DNS Caching Performance in SDN-based Networks" by X. Tang, et al. (2017)

[11] http://mininet.org/api/annotated.html