Disclaimer

A report submitted to Dublin City University, School of Computing for module CA687I – Big Data Cloud, 2023.

We understand that the University regards breaches of academic integrity and plagiarism as grave and serious.

We have read and understood the DCU Academic Integrity and Plagiarism Policy. We accept the penalties that may be imposed should we engage in practice or practices that breach this policy

We have identified and included the source of all facts, ideas, opinions, viewpoints of others in the assignment references. Direct quotations, paraphrasing, discussion of ideas from books, journal articles, internet sources, module text, or any other source whatsoever are acknowledged, and the sources cited are identified in the assignment references.

We declare that this material, which we now submit for assessment, is entirely our own work and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

By signing this form or by submitting this material online we confirm that this assignment, or any part of it, has not been previously submitted by us or any other person for assessment on this or any other course of study.

By signing this form or by submitting material for assessment online we confirm that we have read and understood DCU Academic Integrity and Plagiarism Policy (available at: http://www.dcu.ie/registry/examinations/index.shtml)

Name(s): Vinit Saini, Marcio Vieira, Krystian Fikert
Date: 14/04/2023

Access Control Management App:
An SDN Based Implementation

Introduction and motivation

In modern computer networks, efficient access control mechanisms are crucial to ensure secure and controlled network traffic. Traditional access control methods often lack the flexibility, scalability, and central management required to meet the dynamic needs of today's networks. This is where Software-Defined Networking (SDN) comes into play. SDN has revolutionized the way computer networks are designed, managed, and operated. By separating the control plane from the data plane and centralizing network control, SDN enables dynamic and programmable network management, resulting in improved network agility, flexibility, and scalability. Our SDN application aims to demonstrate the power of SDN by providing an advanced access control mechanism that allows or disallows network traffic based on custom rules defined by the network administrator.

The motivation behind our SDN access control app stems from the growing complexity of network environments, particularly in multi-tenant data centres, where different tenants may have varying security requirements and network policies. By utilizing SDN's programmability and automation capabilities, our application aims to demonstrate dynamic and fine-grained access control, improving network security, enhancing network performance, and reducing the risk of unauthorized access or data breaches.

In conclusion, our SDN access control app aims to showcase a powerful and flexible solution for network administrators to define, manage, and enforce access control policies in modern networks. At the same time it also demonstrate the capabilities of SDN such as scalability, programmability, and efficiency in the area of access control management.

The created SDN network topology and details of the core functions used in the SDN controller design

Following is the diagram of the topology used in our demonstration

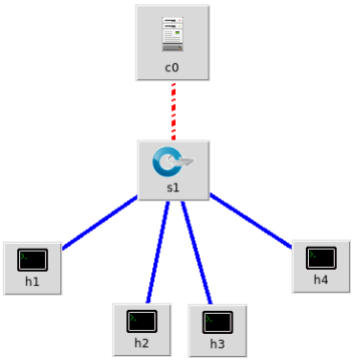

Figure 1 Mininet Topology View (generated by miniedit.py)

Main components and tools used

⇒ Mininet to create a topology
⇒ RYU SDN controller (remote controller)
⇒ Wireshark for monitoring
⇒ Virtual box
⇒ Python

Core functions & features of SDN used

- **Network Topology Discovery:** Relied on RYU's network discovery mechanism to discover the switches and hosts deployed through Mininet emulation software.

- **Access Control and Security:** We have developed a mechanism in our controller to implement the access control function of SDN. Our application reads well-defined rules from a file and utilize these to make decisions if a particular

packet is allowed to be passed through the network. This rule file can be used to Add, Edit or Delete any rule which can be feeded into the network later.

- **Programmability and Automation:** We have also utilized the programmability feature of SDN technology which allows the controller to be easily extended with custom applications or services. We extended the RYU APIs to make our access control application. This involves using RYU APIs and interfaces that enable us to create custom functions, control flows, modifying and applying new flows on OVS Switch. Using RYU APIs we could able to dynamically deploy and execute our code on the SDN controller.

Evaluation design and test results

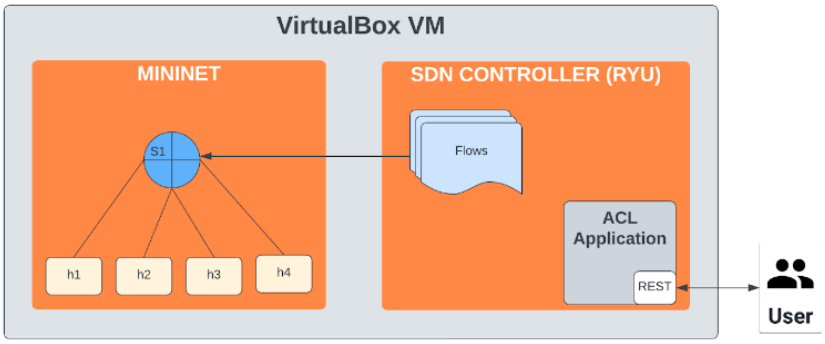

Figure 2Application design on high level (HLD)

The Access Control Management system we have designed is based on Software-Defined Networking (SDN) technology using the Mininet topology and Ryu SDN controller hosted on VirtualBox setup. We have leveraged the Ryu library to build a custom SDN controller that implements the OpenFlow protocol specifications. Our controller also includes a REST interface that allows for the creation and retrieval of rules files from the Open vSwitch (OVS) switch. This REST interface enables our SDN application to apply rules dynamically on-the-fly, which can be understood by the OVS switch, allowing it to make appropriate access management decisions for the topology.

Our Mininet topology consists of one OVS switch (s1), and four hosts (h1, h2, h3, h4) connected through the switch s1. Following figure shows the four hosts having IP address 10.10.0.1 to 10.10.0.4 connected to a vSwitch having DPId as 00:00:00:00:00:00:00:01

Figure 3 vSwitch connections with hosts (generated by floodlight UI)

This is a simple topology designed to demonstrate the use case of Access Control Lists (ACLs). We start by launching the Ryu SDN controller followed by Mininet, which creates our virtual network with a control plane and a data plane. Using HTTP POST requests to our application's REST interface, we upload ACL rules into the application. The application logic then extracts the match criteria and actions from these rules to form new data flows that are eventually pushed down to the OVS switch to take appropriate actions when it receives matching packets on the network.

One of the key features of our application is its ability to take dynamic actions in real-time based on the rules set and pushed by the administrator. This allows the administrator to quickly and efficiently manage the security and throughput of the network. By providing a REST-based interface for rule management and leveraging the flexibility and programmability of SDN, our application offers benefits such as improved network management, rapid response to changing network requirements, and enhanced security by dynamically applying access control rules as needed. Overall, our Access Control Management system built on SDN technology using Mininet topology and Ryu SDN controller provides a flexible and powerful solution for managing network access in a dynamic and efficient manner.

Tests

Following table shows the outcome of a few basic test cases with their inputs and expected output.

Test-Id	Test	Protocol	Input	Expected Output	Result
T1	Allow all requests	Any	h1 wget h2 h1 ping h2	Application should not block any packet	Passed
T2	Block all requests from a specific IP address	Any	h1 ping h2 h1 wget h2	Application should drop such packet	Passed
T3	Block all ping requests only	ICMP	h1 ping h2	Application should not allow this request	Passed
T4	Block all HTTP traffic	TCP	h1 wget h2	Application should not allow this request	Passed
T5	Block all requests except UDP	UDP	h1 hping3 -c 1 --udp h2	Application should allow UDP packets only. All other types are dropped.	Passed
T6	Only specific port is allowed to be requested on a specific host. E.g. http://h2:8080	Any	h1 wget h2	Application will only allow packets destined for port 8080, packets for destination port not 8080 will be dropped	Not-Tested
T7	Specific types of requests are blocked on a specific host. E.g. pings are not allowed for h3	Any	h1 ping h3	Application will drop the ICMP type of packet for h3 host	Passed
T8	Allow access only during specific time periods. Say during a maintenance window	Any	h1 wget h2	Application should block all traffic to a specific host during a specific time window	Not-Implemented
T9	Prevent access from unauthorized devices, say from a different subnet	Any	192.168.1.2 wget h2 Suppose only 10.10.0.0/16 is allowed	Application should block request having source ip other than 10.10.0.0/16 IP	Not-Implemented
T10	Audit and log access control events	Any	Any valid request	Application should log all the events fou audit.	Passed
T11	Modifying ACL rules on the fly		HTTP POST valid rules.json	Application should take effect of changing ACL rules in real time and perform accordingly	Passed
T12	Retreive the existing ACL rules from the switch		HTTP GET request	Application should return existing ACL rules	Passed

Challenges and lessons learned

Well, every day posed a new challenge for us initially, but later we discovered that we could adapt to this virtual environment. There were many hiccups, such as setting up Virtual boxes, Mininet and controllers, as this was a relatively unknown area for all of us. However, once we began to understand how the different components interacted with each other, it became more manageable. Just to highlight a few of the major challenges -

Setting up all-in-one virtual boxes having Mininet and controller packaged: Despite initial expectations, this task proved to be more complicated than anticipated. The ISO images we used were outdated, and we faced various issues during installation and execution, particularly with Floodlight. It seems those were not frequently being updated which caused lot of mismatch between what is running inside the box and what information is available online. Especially, in the case of Floodlight. It was continuously crashing on my system so I had to build Floodlight from binary and then I could able to run it on my local machine and then connecting it to the Mininet VM.

Mismatch of APIs and poor documentation: It is important to highlight that the documentation is not very extensive and adequate at least for Floodlight and RYU APIs which caused quite a lot of confusion, lot more time in troubleshooting and rework.

Monitoring of traffic flows and debugging: Monitoring network packets and understanding internal structure of these packets proved to be time-consuming and labour-intensive. We relied on tools such as Wireshark and TCP command utilities to aid in this process.

Unreliability of Mininet network: Additionally, the unreliability of the Mininet network was a source of frustration. The behaviour of the vSwitch, in particular, was unpredictable at times, and we were unsure if it was due to issues with the Mininet emulated environment or other networking problems.

Despite these challenges, we learned valuable lessons along the way. As we worked with open source projects and different tools in the SDN ecosystem, we gained insights into the inner workings of the controller code. This involved understanding the architecture, design patterns, and algorithms used in popular controllers such as Floodlight and RYU. We also learned how to troubleshoot and debug controller code, including analyzing logs, tracing code execution, and diagnosing errors. This enhanced our skills in identifying and resolving issues related to controller configuration, rules, and policies.

Furthermore, we developed expertise in working with various open source tools that are commonly used in SDN environments. This included Wireshark for packet analysis, Mininet for emulating SDN networks, and TCP command utilities for monitoring and debugging network traffic. These tools were invaluable in diagnosing and resolving issues related to packet flows, network connectivity, and controller-switch communication.

We realized the importance of thorough research and testing before relying on pre-packaged virtual boxes. We also recognized the need for comprehensive and up-to-date documentation for the APIs and tools we used.

Overall, our experience in working with SDN technology and open source projects provided us with valuable insights and expertise. This knowledge and experience will undoubtedly benefit us in future projects and enable us to tackle complex SDN challenges with confidence.

Bibliography

Faucet SDN, (2022) Available at: https://github.com/faucetsdn/ryu (Accessed: April 14, 2023).

Floodlight API. Available at: http://floodlight.github.io/floodlight/javadoc/floodlight/index.html (Accessed: April 14, 2023).

Floodlight Docs, *Atlassian*. Available at: https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/1343622/Javadoc+Entry (Accessed: April 14, 2023).

Kuzin, A., (2019) *OpenFlow application – a web server load balancer*. Available at: https://github.com/annakz1/ryu/blob/master/load_balance.py (Accessed: April 14, 2023).

Mininet, *OpenFlow-tutorial*. (2015) Available at: https://github.com/mininet/openflow-tutorial/wiki (Accessed: April 14, 2023).

Mininet Python API Reference Manual, Available at: http://mininet.org/api/index.html (Accessed: April 14, 2023).

Project Floodlight, *projectfloodlight.org*. Available at: https://github.com/floodlight (Accessed: April 14, 2023).

REST Linkage, *Ryubook 1.0 documentation*, Available at: https://osrg.github.io/ryu-book/en/html/rest_api.html (Accessed: April 14, 2023).

Ryu, *Ryu Documentation*. Available at: https://ryu.readthedocs.io/en/latest/ofproto_v1_3_ref.html (Accessed: April 14, 2023).

Appendix

Link to the video presentation and screencast

Access control management app: https://youtu.be/E_kpBrvIPlI

Link to the code repositories

Access control management app: (Ryu, Python)
https://github.com/mvieiradcu/CA687I-CA2-SDN/tree/main/SDN

Secondary attempt on Java + Floodlight
CDN controller: (Floodlight, Java) [not-completed]
https://github.com/vsdcu/cdn-controller-floodlight.git