



UNIVERSIDADE D
COIMBRA

Henrique Miguel Simões Silva

**FAIRNESS CENTRED FRAMEWORK FOR
THE ONOS SDN CONTROLLER**

Dissertation in the context of the Master in Informatics Engineering, specialisation in Software Engineering and Communications, Services and Infrastructure, advised by Professor Bruno Sousa, co-advised by Researcher Noé Godinho and presented to the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra

September of 2022



FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE
COIMBRA

DEPARTMENT OF INFORMATICS ENGINEERING

Henrique Miguel Simões Silva

Fairness Centred Framework for the ONOS SDN Controller

Dissertation in the context of the Master in Informatics Engineering, specialisation in Software Engineering and Communications, Services and Infrastructure, advised by Prof. Bruno Sousa, co-advised by Researcher Noé Godinho and presented to the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

September 2022



FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE
COIMBRA

DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

Henrique Miguel Simões Silva

Plataforma Focada em Fairness para o Controlador SDN ONOS

Dissertação no âmbito do Mestrado em Engenharia Informática, especialização em Engenharia de Software e Comunicações, Serviços e Infraestruturas, orientada pelo Professor Doutor Bruno Sousa, co-orientada pelo Investigador Noé Godinho e apresentada ao Departamento de Engenharia Informática da Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

setembro 2022

The work presented in this dissertation was carried out within the Laboratory of Communications and Telematics (LCT) group of the Centre for Informatics and Systems of the University of Coimbra (CISUC) in the context of the following projects:

- MH-SDVANET: Multihomed Software Defined Vehicular Networks. FCT (CMU-PT) PTDC/EEI-COM/5284/2020.

This work has been supervised by Professor Bruno Sousa, Assistant Professor at the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.



Acknowledgements

The presented work wouldn't be possible without the effort and support of many individuals. Thus, I would like to take the time to thank everyone that helped me during the dissertation period and allowed me to conclude this project.

First and foremost, I would like to thank my advisor, Professor Bruno Sousa, for providing me with another opportunity to work with him. For his patience when I encountered unexpected problems or made silly mistakes, for his awareness and comprehension of the inevitability of these situations, and for his guidance and availability to resolve each roadblock. It is truly an understatement that this work wouldn't be possible without professor Bruno's support.

Also, I would like to thank my lab colleague, Noé Godinho, that kept my motivation high and gave me valuable advice, avoiding me from becoming sidetracked.

Furthermore, I would like to thank my bachelor's and master's colleagues, Carlos Santos, Francisco Guerra, João Maruesdos, Paulo Gonçalves, Pedro Almeida, and Miguel Teixeira, for listening to my concerns and for making me realise that they were also struggling with similar problems.

Finally, a special thanks to my family and to Rita. Although they don't fully grasp the subject of this work, have always supported my endeavours.

Abstract

In the last years, internet activity has exploded in use. Migration of services to the cloud and emerging new use cases increased the number of users and brought more network traffic with heavier loads. Management and optimisation activities became imperative to assure the quality of experience of users in such information-clogged networks. The SDN paradigm can facilitate the life of network administrators regarding network management and optimisation by presenting the whole network as a programmable entity.

The evolution of SDN technologies brought more options to build and orchestrate networks but also increased the complexity of solutions, raising the knowledge barrier for new network administrators. Furthermore, management challenges start to build up when we consider recent network trends, e.g. mesh connections, battery-powered devices, used to help increase the reliability or computational efficiency of devices, but that are arduous to manage properly.

To combat this, we reviewed orchestration platforms in the literature and researched common management activities to develop a management and optimisation framework for the ONOS SDN controller with helpful forwarding mechanisms.

Our intent is to close the gap between the complex technological stack and network administrators. By maintaining a collection of forwarding algorithms in the framework, users can simply choose the more adequate to use to steer network behaviour in the desired way. To provide an answer for trendy network features, that are difficult to manage, we also propose a novel service fairness mechanism that considers service delay, energy consumption and loss probability.

We aim for our framework to be compatible with wireless scenarios so it can be deployed in emerging 5G. The results achieved in the evaluation of the proposed framework supporting different algorithms reveal good performance indicators of the framework to tackle complex topologies, close to real networks. This document serves the purpose of compiling our research, as well as the main decisions and implementation steps that lead to the target framework.

Keywords

SDN Management, ONOS Framework, Wireless Networks, Fairness

Resumo

Nos últimos anos, atividades que ocorrem na internet explodiram em uso. A migração de serviços para a nuvem e novos casos de uso emergentes aumentaram o número de utilizadores e trouxeram mais tráfego para as redes com cargas ainda mais pesadas. Atividades de gestão e otimização tornaram-se imprescindíveis para garantir a qualidade da experiência dos utilizadores em redes tão congestionadas. O paradigma de redes definidas por *software* (*Software Defined Networks*, SDN) pode facilitar a vida dos administradores de rede no que diz respeito à sua gestão e otimização, apresentando toda a rede como uma entidade programável.

A evolução das tecnologias SDN trouxe mais opções para construir e orquestrar redes, mas também aumenta constantemente a barreira de conhecimento para os administradores de rede. Além disso, os desafios de gerenciamento começam a acumular-se quando consideramos tendências recentes, por exemplo conexões *mesh* ou dispositivos alimentados por bateria, usados para ajudar a aumentar a confiabilidade ou eficiência computacional dos dispositivos, mas que são difíceis de gerir adequadamente.

Para combater isso, revemos as plataformas de orquestração na literatura e pesquisamos atividades comuns de gerenciamento para desenvolver uma estrutura de gestão e otimização para o controlador SDN ONOS com mecanismos de encaminhamento úteis.

A nossa intenção é facilitar a interação entre a complexa pilha tecnológica e os administradores de rede. Ao manter uma coleção de algoritmos de encaminhamento na nossa estrutura, os utilizadores podem simplesmente escolher o mais adequado para orientar o comportamento da rede segundo a maneira pretendida. Para dar resposta às novas propriedades de redes modernas, que são difíceis de gerenciar, também propomos um novo mecanismo de justiça entre serviços que considera atraso do serviço, consumo de energia e probabilidade de perda.

O nosso objetivo é que a nossa estrutura seja compatível com cenários sem fio para que possa ser implantada em ambientes 5G. A avaliação da framework, com suporte para diversos algoritmos, revela que a mesma consegue suportar de forma satisfatória topologias complexas, baseadas em redes de cenários realistas. Este documento serve o propósito de compilar a nossa pesquisa, bem como as principais decisões e etapas de implementação que levam à estrutura alvo.

Palavras-Chave

Gestão de redes SDN, Plataforma para ONOS, Redes sem fios, Justiça entre Serviços

Contents

1	Introduction	1
1.1	Main Objectives	4
1.2	Contributions	4
1.3	Document Structure	5
2	Background	7
2.1	Software Defined Networking	7
2.1.1	Traditional Networking Paradigm	7
2.1.2	SDN Paradigm	8
2.1.3	SDN controllers	11
2.1.4	Comparison	17
2.2	SDN Related Protocols	19
2.2.1	Openflow	19
2.2.2	Data plane programming and P4	21
2.2.3	Other Protocols and Languages	22
2.3	Network emulation	23
2.3.1	Mininet	23
2.3.2	Mininet-WiFi	24
2.4	Traditional Monitoring Protocols	24
3	Research Projects & Related Work	27
3.1	Ongoing projects	27
3.1.1	SNOB-5G	27
3.1.2	MH-SDVANET	29
3.2	Related SDN European projects	31
3.3	Management solutions for SDN	31
3.3.1	Summary	33
4	Research Objectives & Approach	35
4.1	Objectives	35
4.2	Approach	37
4.2.1	Research and Development Methodology	37
4.2.2	Planning	38
4.2.3	Issues and Readjustments	41
4.2.4	Risks	44
5	Requirements Elicitation	47
5.1	Management Activities	47
5.2	Functional Requirements	49

5.2.1	Use Case Diagrams & User Stories	50
5.3	Non Functional Requirements	61
5.4	Design and Technical Restrictions	61
5.5	Requirement Listing	62
5.5.1	Requirement Fulfilment Analysis	69
6	Project Architecture	71
6.1	C4 Model	71
6.2	Architectural Artefacts	73
7	Framework Development	79
7.1	Permanent Storage	79
7.1.1	Database Schema	79
7.1.2	Metrics collector	81
7.2	Web Server	82
7.3	Forwarding Mechanisms	84
7.3.1	Custom Fairness algorithm	85
7.3.2	K-shortest ONOS algorithm	88
7.4	Potentially Interesting Features	88
8	SDN Experimental Scenario & Algorithm Results	91
8.1	Experimental Environment	91
8.1.1	Problems with Mininet-Wifi	96
8.1.2	Mininet Environment	97
8.1.3	Generate Traffic	98
8.1.4	Energy Formulas	99
8.2	Experiences & Results	99
8.2.1	Heuristic Validation	102
8.2.2	Partial Inter Cluster	102
8.2.3	Full Inter Cluster	106
8.2.4	Intra Cluster	112
8.3	Discussion	115
9	Feature Testing & Validation	117
9.1	Web Server End-points	117
9.2	CLI Commands	122
9.3	Energy Formulas	125
9.4	Algorithms	126
10	Conclusion	129
10.1	Future work	130
Appendix A	Terminology	143
Appendix B	Openflow Pipeline and Control Channel	147
Appendix C	P4 Pipeline and Example	155
Appendix D	Framework Vision in the First Semester	161

Acronyms

ABC Activity-Based Congestion management.

AC-RLNC Adaptive and Causal Random Linear Network Coding.

AP Access Point.

API Application Programming Interface.

ATCLL Aveiro Tech City Living Lab.

BGP Border Gateway Protocol.

BMV2 Behavioral model version 2 (BMV2).

CLI Command-Line interface.

DPI Deep Packet Inspection.

gNMI gRPC Network Management Interface.

gRPC Google's Remote Procedure Calls.

GUI Graphical User Interface.

IDS Intrusion Detection System.

IETF Internet Engineering Task Force.

IoT Internet of Things.

IPFIX Internet Protocol Flow Information Export.

ITU International Telecommunication Union.

JDBC Java Database Connectivity.

MATs match-action-tables.

MD-SAL Model-Driven Service Abstraction Layer.

MEC Multi-access Edge Computing.

mmWave Millimeter Wave.

NETCONF Network configuration.

OBU On-Board Unit.

ODL OpenDaylight.

OF Openflow.

OF-Config Openflow Management and Configuration.

ON.Lab Open Networking Lab.

ONAP Open Network Automation Platform.

ONF Open Networking Foundation.

ONOS Open Network Operating System.

ORM Object-Relational Mapping.

OSGi Open Services Gateway initiative.

OVS Open vSwitch.

P4 Programming Protocol-Independent Packet Processors.

PaaS Platform as a Service.

PCEP Path Computation Element Communication Protocol.

PISA Protocol-Independent Switching Architecture.

QoE Quality of Experience.

QoS Quality of Service.

RAN Radio Access Network.

REST Representational State Transfer.

RESTCONF Representational State Transfer Configuration Protocol.

RPC Remote Procedure Call.

RSSI Received Signal Strength Indication.

RSU Road Side Unit.

RTCM Rede Temática de Comunicações Móveis.

SaaS Software as a Service.

SAL Service Abstraction Layer.

SD-CORE Software Defined Core.

SD-RAN Software Defined Radio Access Networks.

SDN Software Defined Network.

SDN-C Software Defined Network Controller.

SFC Service Function Chaining.

SLA Service Level Agreement.

SNMP Simple Network Management Protocol.

TL1 Transaction Language 1.

TLS Transport Layer Security.

TTL Time to Live.

UC Use Case.

URI Uniform Resource Identifier.

US User Story.

VANET Vehicular ad hoc networks.

VFC Virtual Function Chains.

VNF Virtual Network Function.

WLAN Wireless Local Area Network.

WMN wireless mesh networks.

List of Figures

2.1	Traditional network architecture example	8
2.2	Roles of the control and data planes in traditional switches [Goransson et al., 2016]	8
2.3	SDN network architecture example	9
2.4	ONOS architecture [Bill Snow, 2015]	12
2.5	OpenDaylight architecture - operational view [OpenDaylight Team, 2020]	13
2.6	Lighty.io architecture [PANTHEONtech, 2018]	15
2.7	ONAP architecture [ONAP Team, 2022a]	16
2.8	Ryu architecture [Irawati and Nuruzzamanirridha, 2015]	17
3.1	Traditional wireless communication networks encountered in 5G [Cohen et al., 2020]	28
3.2	Proposed architectures for seamless handover in MH-SDVANET [Silva et al., 2021]	30
4.1	Gantt Chart for the second semester	40
4.2	Activities performed in the second semester	42
5.1	Algorithm management diagram	51
5.2	Asset management diagram	53
5.3	Configuration management diagram	56
5.4	API Utilisation diagram	58
6.1	Architectural diagram - Preliminary draft	72
6.2	C4 architectural model label	73
6.3	C1 architectural diagram - Context	74
6.4	C2 architectural diagram - Container	75
6.5	C3 architectural diagram - Component: ONOS management application	77
7.1	Database conceptual diagram	80
7.2	GET reply example	83
7.3	GET reply example with filtering	83
8.1	Aveiro Open Lab technologies [aveirotechcity, 2021]	92
8.2	Aveiro open lab wifi topology	93
8.3	Virtual topology draft	93
8.4	Cisco C9130 AXI 2.4GHz Antenna patterns [Cisco, 2021]	95
8.5	Mininet-Wifi Virtual Topology Diagram	96

8.6	Mininet Virtual Topology Diagram	98
8.7	Topology Representation in the ONOS GUI	101
8.8	Partial Inter Cluster - Packets Sent Comparison	105
8.9	Full Inter Cluster - Packets Sent Comparison - Sent by swc39	111
8.10	Full Inter Cluster - Packets Sent Comparison - Sent by swc40	111
8.11	Full Inter Cluster - Packets Sent Comparison - Sent by swc41	112
8.12	Intra Cluster - Packets Sent Comparison	114
9.1	Structure of the JGraphT Test	127
9.2	Structure of the JGraphT Modified Test	127
A.1	Network coding in a butterfly network [Bassoli et al., 2013]	145
B.1	Flowchart of packet pipeline in an Openflow switch [Open Net- working Foundation, 2015]	148
C.1	PISA model [Hauser et al., 2021]	156
C.2	V1model pipeline architecture [Hauser et al., 2021]	156
D.1	Vision for the Framework in the First semester	162

List of Tables

2.1	SDN Controller Comparison	18
2.2	Openflow evolution	20
2.3	Openflow - flow table example	20
2.4	P4 specification history	22
3.1	Related work summary	34
4.1	Risk classification caption	45
4.2	Risks identification	46
5.1	Management activities	48
5.2	Metrics necessary for management activities	49
5.3	Management activities requirements list	63
5.4	Algorithm management requirements list	65
5.5	Asset management requirements list	65
5.6	Configuration management requirements list	66
5.7	API utilisation requirements list	67
5.8	Miscellaneous non functional requirements list	68
5.9	Restrictions and constraints requirements list	69
7.1	Device example after forwarding decisions	84
8.1	Services and respective traffic models	98
8.2	Power model settings	100
8.3	Partial Inter Cluster Flows	103
8.4	Partial Inter Cluster Paths	104
8.5	Partial Inter Cluster - iPerf UDP Metrics	106
8.6	Full Inter Cluster Flows	107
8.7	Full Inter Cluster Paths	107
8.8	Full Inter Cluster - iPerf UDP Metrics	112
8.9	Intra Cluster Flows	113
8.10	Intra Cluster Paths	113
8.11	Intra Cluster - iPerf UDP Metrics	115
9.1	BlackBox Tests - Retrieving Information with End-points	118
9.2	BlackBox Tests - Retrieving Filtered Information with End-points	119
9.3	BlackBox Tests - Insert and Update Information with End-points	120
9.4	BlackBox Tests - Retrieving Information with CLI	122
9.5	BlackBox Tests - CLI for Management	123
9.6	BlackBox Tests - Normalised Objective Values	126

9.7	BlackBox Tests - Reserve and Replenish Bandwidth	128
B.1	Instructions of Openflow version 1.5.1	149
B.1	Instructions of Openflow version 1.5.1	150
B.2	Actions of Openflow version 1.5.1	150
B.3	Counters of Openflow version 1.5.1	151
B.3	Counters of Openflow version 1.5.1	152

Chapter 1

Introduction

The recent migration explosion of services to the cloud came to flood networks with a great deal of data traffic. Recent studies estimated that, by 2023, there will be 5.3 billion total internet users, more than 35% of what was registered in 2018 and the average fixed broadband speeds would handle 110.4 Mbps, up from 45.9 Mbps in 2018 [Cisco, 2020].

New emergent technologies and services also trend towards high-performance applications, taking advantage of infrastructures developments and internet proliferation, contributing to the increase in network traffic: Internet of Things (IoT) sensors can enable smart irrigation systems and elderly care monitoring [Fraga-Lamas et al., 2020; Stavropoulos et al., 2020]; smart-cities environments collect data about traffic conditions, air quality, surveillance for fires or gas leakage [Atitallah et al., 2020]; cloud computing allows Platform as a Service (PaaS) and Software as a Service (SaaS) [Sun, 2020]; big data integration in smart environments [Hajjaji et al., 2021]; popularisation of short-video sharing and video on demand [Zhang et al., 2022].

As these applications get more rigorous requirements, they expect more bandwidth and higher speeds, which leads to heavier loads, near to real-time sampling, and denser topologies. Network management becomes an unavoidable activity to guarantee Quality of Service (QoS) and handle applications with such stiff requirements in an efficient and economical way.

Furthermore, the challenges of management increase when we consider that many of these services need special attention due to having wireless connections: managing networks with heterogeneous wireless technologies and spectrum interference (e.g 5G, LoRA, Zigbee, IEEE 802.11); tracking of elements involved in mobility; seamless service handover.

Thankfully, network management has become simplified due to the evolution of the Software Defined Network (SDN) paradigm [Goransson et al., 2016]. This technology separates the device's behaviour algorithms from the network infrastructure that enforces it and moves them to a new entity, the Software Defined Network Controller (SDN-C). The controller acts as a logically centralised agent that network devices report to allowing the SDN-C to have a global view of the

behaviour of the network. Network administrators can then write the desired behaviour into SDN-C applications to configure and optimise multiple network resources as they please. There is no need to implement logic directly into devices anymore since the topology can be seen as a big programmable entity. This technology enables a more flexible network behaviour and management of its assets compared with its classical counterpart.

Throughout the years, technologies to enable SDN architectures appeared and evolved as a response to issues faced by the community. Some well-structured technologies became industry standards. For device and controller communications the standard protocol solution is Openflow [Open Networking Foundation, 2015]. Devices maintain a collection of tables, with rules populated by the controller, to manipulate network packets. However, for wireless environments with mobility and multihoming Openflow is not a perfect fit (e.g. weak support for forwarding with mobility, inflexible to configure statistics).

More recently, a new paradigm materialised to answer some issues of Openflow, the data plane programming model, that allows the customisation of packet pipeline table headers. From the proposed protocols the one that became standard was the Programming Protocol-Independent Packet Processors (P4) language [Hauser et al., 2021].

Also, a plethora of SDN controllers have been developed, each one with different features, architectures, and objectives. To name some of the more popular, we have Open Network Operating System (ONOS)¹, maintained by Open Networking Lab (ON.Lab), OpenDaylight (ODL)², maintained by the Linux Foundation, and some community supported projects like Ryu³.

The deep technological stack and different options to consider means that the entry barrier to network managers can be high. Know-how of different evolving technologies and systems is required to guarantee essential network properties, e.g. user fairness and congestion avoidance. Furthermore, intricate network compositions, with mesh connections, wired and wireless links, traffic of different services and devices with heterogeneous energy consumption and computational power further stress the need to manage resource and node usage more intelligently and with a fairness perspective [Ghaleb et al., 2021]. This is critical to avoid unfair channel distribution and assignment so that all the nodes have the same opportunities to send and receive information, or to prolong network lifetime by not draining the battery of devices.

In this work, we developed a framework for the ONOS SDN-C that can expose end-points for easier network monitoring and management and that provides diverse forwarding solutions, including a novel multi-objective fairness mechanism.

Such end-points allow users and external tools to act on the topology, monitoring devices and configuring the behaviour of forwarding algorithms. Framework

¹<https://opennetworking.org/onos/>

²<https://www.opendaylight.org/>

³<https://ryu-sdn.org/>

users can activate or deactivate available forwarding solutions without having to worry about the underlying technologies.

These forwarding mechanisms are maintained in the framework as a collection of possible forwarding algorithms to use. This way, users can choose the one more appropriate to use in each individual scenario: e.g. find a path that minimises energy cost, or the path with less delay, or even a route that minimises energy cost while maximising resilience. Currently, the framework provides two distinct forwarding solutions: one using the K-shortest algorithm with energy concerns; and another modelled as a min-cost-flow greedy heuristic with three objectives. Upon detection of a new service flow in the network, the framework handles it by calculating the forwarding path and installing Openflow rules in the affected devices, **making this a completely automated process.**

To evaluate the framework performance, we emulated a scenario of a real smart-city environment with Mininet and generated traffic flows of services typically found in these scenarios: video streaming, voice communications and Web resources. The emulated ambient contains illustrative characteristics of a possible topology where the framework would be deployed: wireless and mesh connection, node grouping in clusters and hardware heterogeneity. The paths obtained by our custom service fairness solution were compared against the out-of-shelf K-shortest mechanism that ONOS supports, to prove that these approaches do not scale properly to be used in complex and constantly changing networks. Furthermore, we also evaluated the paths obtained by our fairness centred mechanism, against the mathematical formulation from where our heuristic was derived: the heuristic is implemented in the framework and gives a close to optimal path, while the formulation always provides the optimal path, but is too slow to use in a real-time setting.

The results obtained showed that, for a scenario of node communications within the same cluster, the heuristic obtained the same value for the objective function as the formulation, which means that **the heuristic retrieved one of the best possible solution paths.** For experiments of communications between different clusters our method was able to distribute the load between redundant nodes, while the approach of ONOS remained incapable of taking advantage of these alternative paths. Additionally, we found that our mechanism was able to recognise the changes in the network resources throughout the traffic simulations, and propose, better suited, alternative paths for subsequent instances of a service.

When comparing the overall quality of the services, between the paths of the fairness heuristic and the K-shortest path of ONOS, we found that the variations in the jitter were almost unnoticeable, and in most cases, the differences in loss percentage were more homogeneous in the services that used the fairness mechanism paths. This shows the potential gains of our approach, over standard solution offered by common SDN controllers.

1.1 Main Objectives

The main objectives of this work are twofold. Firstly, design and evaluate a framework for the ONOS SDN controller, which incorporates management functionalities and forwarding solutions, to facilitate the activities of network administrators. Secondly, deliver our proposal for service fairness through a forwarding mechanism available in the framework and demonstrate its advantages over out-of-shelf solutions offered by SDN controllers.

1.2 Contributions

This section serves the purpose of summarising the main contributions of this dissertation:

1. **Analysis and comparison of SDN related technologies** (see Chapter 2). Throughout this work, we were required to interact and gain knowledge of many SDN related mechanisms: compare popular SDN-C; compare and understand standards for southbound communications; create topologies in network emulators.
2. **Contribution to research projects and academic workshops** (see Chapter 3). Some objectives of this work are aligned with research projects where we are integrated as researchers, namely SNOB-5G and MH-SDVANET. We were able to provide meaningful feedback to our research colleagues, accelerate SDN-C integration, explain the OF and P4 pipeline, etc. We also shared our endeavours in workshops of the department like "LCT Workshop 2022" and "RTCM 2022".
3. **Implementation of a novel fairness model** (see Section 7.3.1). We implemented a heuristic in ONOS that gives us a close-to-optimal path solution for service traffic. This heuristic works as a single-path min-cost-flow algorithm with three objectives (energy consumption, link delay, and observable packet loss) and is constrained by bandwidth and link usage criteria.
4. **Development of a management framework for ONOS** (see Chapter 7). Upon detecting a new flow, the framework runs the preferred forwarding algorithm and installs the respective forwarding rules in the devices of the topology. The framework collects topology information and statistics in a database to expose it through Representational State Transfer (REST) endpoints to users and external tools.
5. **Writing a conference paper** We also compiled our work in preparation for submitting it to the IEEE Consumer Communications & Networking Conference⁴, a B ranked conference. "SALEM: Service Fairness in Wireless Mesh Environments" presents our fairness model and compares it against

⁴<https://ccnc2023.ieee-ccnc.org/workshop/sdwn23>

an out-of-the-shelf algorithm of ONOS in a real-world smart-city emulated setting.

1.3 Document Structure

The remaining document consists of the following chapters.

Chapter 2 describes the key concepts necessary to understand this work. It compares SDN vs traditional network paradigm, popular SDN controllers differences and communication protocols. It also introduces emulation tools and monitoring protocols.

Chapter 3 reviews related SDN management platforms and their activities. Furthermore, this chapter presents research projects that this work contributes towards.

Chapter 4 explains the motivation behind this dissertation and the main goals to achieve. Later in the chapter, the research and development approach is presented, and the main issues faced are documented alongside how they were handled.

Chapter 5 contains requirements artefacts produced and a requirement listing that establishes the scope of our work.

Chapter 6 provides the technological choices made to implement our framework in the form of architectural diagrams.

Chapter 7 displays the implementation steps and decisions made during development. It documents information about the database schema, the web server that exposes end-points, and how we reactively detect new flows and install the necessary flow rules.

Chapter 8 describes the experimental setup and the experiences conducted to evaluate our framework performance and success.

Chapter 9 compiles the procedures taken to validate the correctness of features and routines.

Chapter 10 provides closing remarks and summarises the outcome of this work.

Appendix A contains a description of critical networking concepts required to understand this work.

Appendix B documents the behaviour of the OF packet pipeline and control messages exchanged between devices and controller.

Appendix C explains the P4 packet pipeline and provides an example of a routing solution using this protocol.

Appendix D illustrates the aim of this project in the first semester, so the reader can better understand the reasoning behind the scope adjustments.

Chapter 2

Background

In this chapter, we cover the key concepts necessary to understand our work. In Section 2.1, we present the reasoning behind the need for Software Defined Network (SDN), alongside a comparison of popular SDN controllers. Section 2.2 explains the behaviour of SDN related protocols and their evolution. Popular network emulator tools are documented in Section 2.3 and in Section 2.4 we introduce relevant traditional protocols for network monitoring.

2.1 Software Defined Networking

Despite the impressive accomplishments of traditional networking paradigms, the modern computer network has evolved into a complex system that is challenging to manage. With accelerated innovation, new business models, and emerging services enabled by new technologies, traditional devices can't cover the size and complexity of the requirements vital for today's networking environment [Goranson et al., 2016].

In this section, we present the main SDN architecture ingredients and their contribution to a network. We also explain the benefits that SDN brings in comparison with the more traditional networking paradigm.

2.1.1 Traditional Networking Paradigm

In a classical network architecture, see Figure 2.1, each device would have its own data plane and control plane. The logic programmed into the device is the control plane, while the hardware and structures that enforce it would be considered the data plane.

Data Plane

Making an example out of the traditional switch, see Figure 2.2, the data plane consists of the functions and processes that allow reception of data packets, information modifications, and finally transmission, all based on control plane logic.

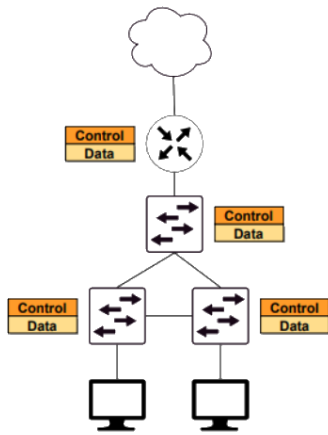


Figure 2.1: Traditional network architecture example

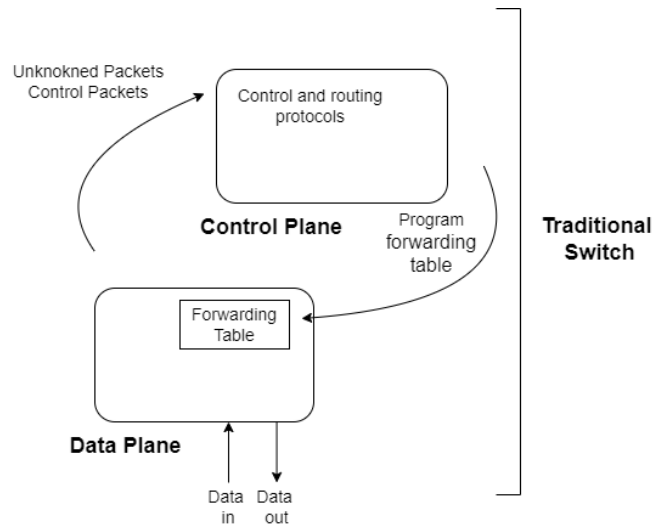


Figure 2.2: Roles of the control and data planes in traditional switches [Goransson et al., 2016]

The data plane assumes responsibility for packet buffering, packet scheduling, header modification, and forwarding.

This plane has no embedded intelligence to take autonomous decisions, so the switch follows the rules of a forwarding table that was populated by the control plane. When an arriving data packet's header information is found in the forwarding table, it will be forward without any intervention of the control plane [Goransson et al., 2016].

Control Plane

The logic and protocols that run on the switch and populate the forwarding table constitute the control plane. The control plane intervention is needed when the incoming packet's information is not currently available in the forwarding table or it belongs to a control protocol. Its primary role is to keep the information in the forwarding table current so that the data plane can independently handle a high percentage of traffic as possible [Goransson et al., 2016].

Management Plane

There is also a third plane, the management plane. It stands above the control plane and allows a network administrator to monitor and configure devices by extracting and modifying network information collected by control plane protocols. Network administrators have access to a Command-Line interface (CLI) or Graphical User Interface (GUI) to perform interactions and can write scripts or algorithms to manage the control plane protocols [Goransson et al., 2016].

2.1.2 SDN Paradigm

The physical separation of the control and data plane is one of the major principles that SDN is known for [Jarschel et al., 2014]. The network devices maintain their data plane but the control plane functions are handled by an external en-

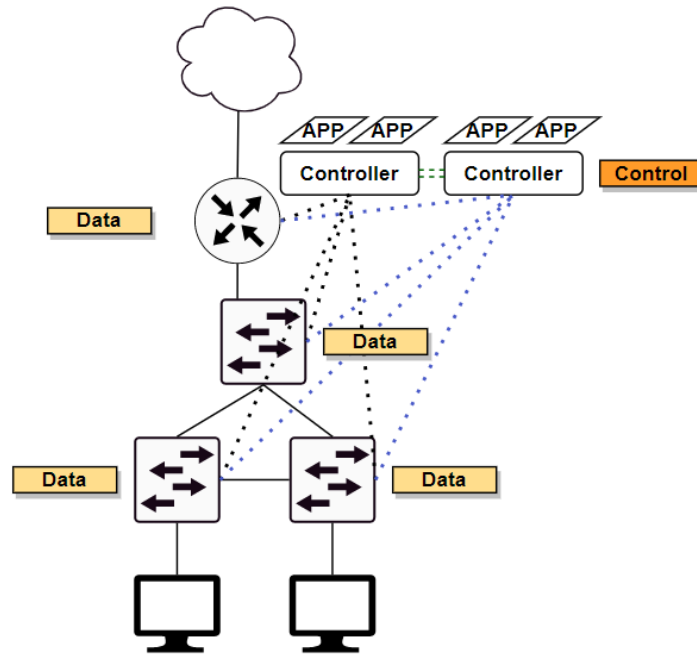


Figure 2.3: SDN network architecture example

tity, often called SDN controller, see Figure 2.3. Network devices report to the Software Defined Network Controller (SDN-C) so the controller has a bird’s-eye view of the topology so it can manage each device’s forwarding tables taking into account the traffic of the whole network.

This architecture style allows the existence of network applications that can be programmatically built and deployed onto the controller’s *northbound interface*, at run-time. These applications steer the way the controller manages the network devices, allowing changes on the fly, and making the network’s behaviour more flexible.

These applications serve as an interface for network administrators to configure the control plane to work as they need, thus they can be seen as a part of the management plane.

To the “south” of the controller, there is the *southbound interface*. It exchanges specific protocol messages between controller and network devices to e.g. configure devices, populate forwarding tables, and retrieve information.

The centralised control logic allows a global view of the network but poses some new challenges in terms of scalability and performance when requests to the controller increase or when the size of the network grows [Benamrane et al., 2017].

To reduce this problem, popular solutions involve having multiple SDN-C or even SDN-C clusters in a topology to enable a logically centralised or logically distributed control plane [Sufiev et al., 2019]. In a logically centralised control plane, SDN-Cs balance network loads between each other and synchronise their decisions. The synchronisation can be challenging, so the logically distributed approach divides the network into smaller domains and assigns each one to a

controller.

Communication between controllers is achieved through another interface, the *east-west interface*. It utilises different Application Programming Interfaces (APIs) and protocols to achieve controller-to-controller communication. There is worth mentioning that, not like the north or south interfaces, there isn't a industry standard protocol for this type of activity [Almadani et al., 2021].

Furthermore, multiple controllers can also be found as a way to provide other characteristics: e.g primary and backup controllers can help increase network resilience.

The traditional static way of managing network devices is time consuming, error prone, and leads to inconsistencies. In comparison, the SDN paradigm brings several beneficial characteristics:

Simpler and cheaper devices In SDN, network devices are controlled by a centralised system. This means that the same thousands of lines of complicated control plane software don't need to be deployed in each device. Each one only requires a set of components to handle data plane traffic directed to it and to communicate with the controller, instead of being loaded with general purpose functionality. With such an approach, the cost of specialised equipment drops, and devices with lower resources in terms of computational power can be part of the topology [Goransson et al., 2016].

Openness Another characteristic of SDN is that its interfaces are open. Having standard, well documented, non-proprietary interfaces opens room for more flexibility and adaptable topologies with new and innovative methods of network operation. A closed or proprietary interface would limit interoperability between components and reduce innovation [Goransson et al., 2016].

Programmability The external logically centralised controller and the open interfaces provide the ability to treat the network as a single programmable entity instead of an accumulation of devices that need to be configured individually. The network is now more than a "sum-of-its-parts", it opens up the customisation of the network according to a specific setup or scenario [Goransson et al., 2016; Lopes et al., 2016].

Elasticity Elasticity is the ability that allows SDN elements to dynamically adapt to the requirements of their environment by scaling up or down the available resources. Due to the software nature of the controller, when necessary, it can scale up or down, by running in more/less powerful machines [Benzekki et al., 2016].

Resilience defines the ability to maintain acceptable service experience in the face of faults and atypical behaviour. In case of a service or node failure, the controller can shift the faulty device responsibilities into another node or alter the traffic path. Even SDN-C failure can be handled, e.g. by having redundant controller instances [Benzekki et al., 2016].

Virtualisation SDN facilitates virtualisation of components and functions by assuring that the data can be routed properly between services or functions according to defined policies. The logical instances of services that form a Virtual Network Function (VNF) can be deployed in distinct network elements, and the SDN controller will provide a flexible way to enforce policies and manage service host resources [Bizanis and Kuipers, 2016].

2.1.3 SDN controllers

As mentioned before, in SDN architectures the control plane is moved off the switching devices and onto a centralised controller. This controller maintains a view of the entire network, implements policy decisions, manages the decision of infrastructure devices, and provides a northbound interface for custom applications.

The controller of an SDN network is a logically centralised entity, that is, it can consist of multiple physical or virtual instances but behaves like a single component.

The global network information that is collected in the SDN-C enables it to adapt its network policy with respect to routing and forwarding much better and faster than a system with traditional routers could.

There are different available implementations of SDN controllers in the market, each one was built with different objectives in mind and now are maintained by different entities. The following subsections document differences between popular SDN-C solutions, with regards to their background, architecture model, provided interfaces, and supported protocols.

ONOS

Open Network Operating System (ONOS) is an open-source SDN controller released on 5 December 2014 by Open Networking Lab (ON.Lab), and it has been actively improved.

The ONOS project gained traction in the community and many of the major service providers and transport vendors, including Alcatel-Lucent, AT&T, China Unicom, Ciena, Cisco, Ericsson, Fujitsu, Huawei, Intel, NEC, NTT Communications, among others, become partners and invest into ONOS development [Bill Snow, 2015]. Also, on October 13, 2015, the Linux Foundation entered a partnership with the ONOS project [THE LINUX FOUNDATION, 2015]. ONOS is written in Java and uses Apache Karaf Open Services Gateway initiative (OSGi) containers. It is designed to be distributed, stable and scalable with a focus on service provider networks [Open Networking Foundation, 2019].

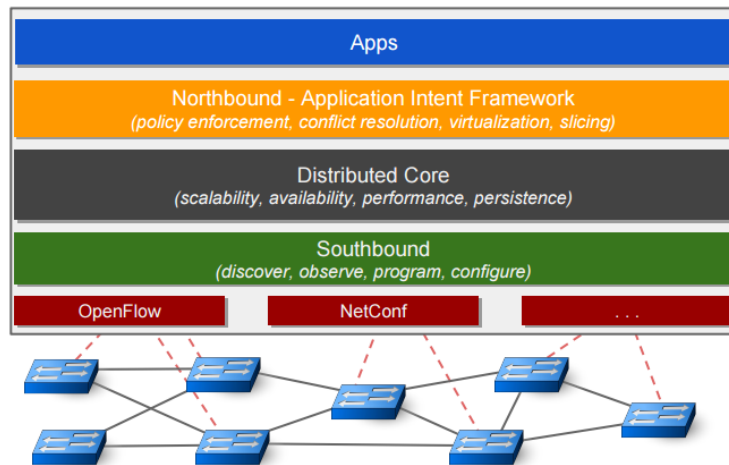


Figure 2.4: ONOS architecture [Bill Snow, 2015]

Architecture The controller’s architecture, as depicted in Figure 2.4, enables its core to expose abstractions, information models, and the network state to northbound programmable applications. This prevents the system from becoming tied to a specific configuration or control protocol.

ONOS is built as a physically distributed system but remains logically centralised. This means that in an SDN network with ONOS, the controller is actually a symmetric cluster of individual instances [Thomas Vachuska, 2015]. Note that network devices need to be connected to each one of these controller instances.

This enables a network that can easily scale up or down the resources at the control plane, that is more resilient to failures, becomes fault-tolerant, and is capable of balancing workloads by distributing the load between individual controllers in the cluster.

Interfaces On the north-facing side, ONOS offers many interfaces for communication between applications and its core, namely, REST API, GUI, Google’s Remote Procedure Calls (gRPC), Representational State Transfer Configuration Protocol (RESTCONF), or CLI. Also, its northbound applications can be connected or disconnected in run-time which adds more flexibility when trying to manage networks dynamically.

On the southbound interface, again the abstraction of the core guarantees that there are no dependencies on the protocols used on the south-facing side of the controller. Still, ONOS supports multiple controller-device communications protocols in its networks. Openflow (OF) is the primary southbound protocol the SDN controller focuses on, but it can work with other SDN related protocols like Programming Protocol-Independent Packet Processors (P4), Network configuration (NETCONF), and with non-SDN protocols too like Transaction Language 1 (TL1), Simple Network Management Protocol (SNMP), Border Gateway Protocol (BGP), and Path Computation Element Communication Protocol (PCEP) [Andrea Campanella, 2016].

ONOS also supports the dynamic creation of flows/rules in devices through an intent-based abstraction. Intents provide an abstraction that give appli-

cation programmers the ability to code what they want to be accomplished in the network rather than how it should be accomplished. For instance, application developers only need to express the need for communication from a source node to a destination node.

OpenDayLight

OpenDaylight (ODL) is currently the major competitor of the ONOS controller. ODL is a collaborative open-source project hosted by The Linux Foundation. It launched on 5 February 2014 in partnership with Brocade, Cisco, Citrix, Ericsson, IBM, Juniper Networks, Microsoft, NEC, Red Hat, and VMware [LightReading Team, 2013].

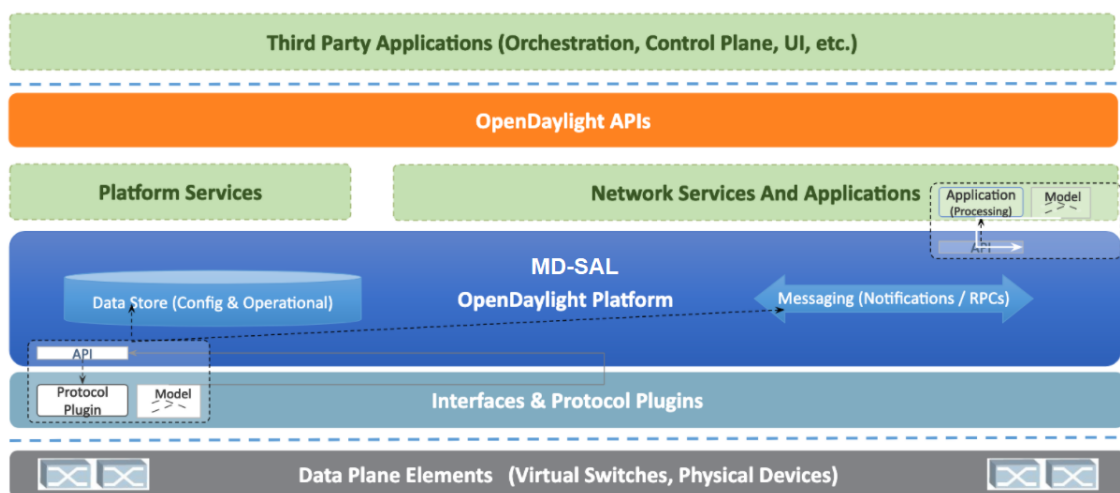


Figure 2.5: OpenDaylight architecture - operational view [OpenDaylight Team, 2020]

Currently, ODL is being used around the globe in many ways, by telcos, enterprises, research, and academic institutions like Alcatel-Lucent, AT&T, China Unicom, Ciena, Cisco, Ericsson, Fujitsu, Huawei, Intel, NEC, and NTT Communications [The Linux Foundation, 2021a]. Just like ONOS, ODL is a Java written controller that includes Karaf OSGi containers.

Architecture The core of the ODL platform is the Model-Driven Service Abstraction Layer (MD-SAL), see Figure 2.5.

The MD-SAL has two major items: the datastore, with is composed of the *config datastore* and the *operational datastore*, and the message bus that provides a way for the various services and protocol drivers to notify and communicate with one another. The *config datastore* maintains a representation of the desired network state and the *operational datastore* represents the actual network state.

Similar to ONOS's core, Service Abstraction Layer (SAL) provides generalised descriptions of network devices or application capabilities without needing to know the specific implementation details or the protocols used

between the controller and the network devices. This is possible because these data exchanges are made using generic modelling protocols (i.e. YANG data models) [RFC7950, 2016]

The SAL exposed models can be of two role types, “producer” or “consumer”. The producer implements end-points that expose information and the consumer utilises that information [The Linux Foundation, 2021b].

Another similarity with ONOS is that ODL is built to have a fault tolerance mechanism. In a scenario of a master node failure, a new leader is selected to take control.

Interfaces The Yang models that the SAL data store exposes are utilised by the north applications and by the south protocols as a way to interface with the controllers core. They access producer or consumer models depending on the job they perform. For example, a southbound protocol plugin could either be a *producer* of information about the network or a consumer of northbound application instructions it receives via the SAL.

ODL offers a large set of northbound interfaces with gRPC, Representational State Transfer (REST) API, RESTCONF, CLI, and GUI. Just like ONOS, this controller takes advantage of OSGi containers for loading bundles at run-time, allowing a very flexible approach to adding functionality. On the southbound interface, ODL supports multiple protocols as plugins, e.g. OF 1.0, 1.3, 1.4, P4, NETCONF, SNMP, PCEP, LISP, and BGP with link state [OpenDaylight Project, 2021].

Lighty.io

Lighty.io is a lightweight ODL run-time library. It is developed by PANTHEON.tech, a privately owned company, based in Slovakia previously involved in leading ODL projects. The main idea of Lighty.io is to concentrate on the network functionalities and remove the dependencies of Apache Karaf that are present both in ONOS and ODL.

Lighty.io can run on a plain Java SE environment [PANTHEONtech, 2021c]. The initial Github code drop was on the 11 of May 2018 [PANTHEONtech, 2021a]. This project has the partnership of major enterprises and vendors like Arista, AT&T, Cisco, Frinx, Huawei Kaloôm, Huahuan, and many others [PANTHEONtech, 2021d]. It is available in a free and in a premium paid version.

Architecture Lighty.io core contains base ODL services and components like the MD-SAL and YANG Tools, see Figure 2.6. Without the dependence on Karaf, this solution offers an easier deployment, reduced cost of testing, integration, and maintenance, and a simpler tool when compared with the complexity of OpenDaylight’s Karaf features. It also maintains the MD-SAL clustering support from ODL [PANTHEONtech, 2021e].

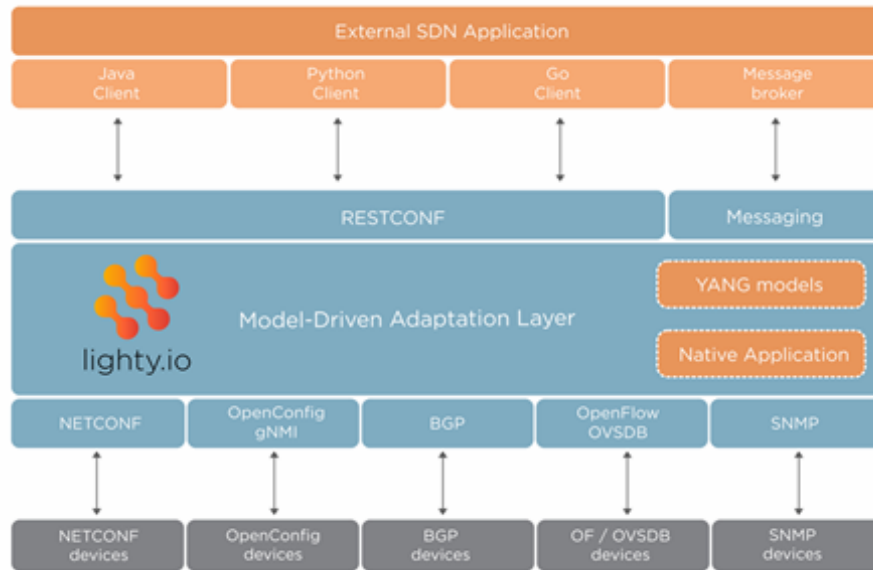


Figure 2.6: Lighty.io architecture [PANTHEONtech, 2018]

Interfaces Lighty.io allows north applications to interface with the platform’s core using RESTCONF, and client libraries in Java, Python, and Golang. This controller also offers enhanced NETCONF device simulator for testing and YANG model data serialisation and deserialisation, both not present in ODL. Users can also access the controller using a GUI [PANTHEONtech, 2021e].

On the south interface, the controller offers support for SNMP, BGP, NETCONF, gRPC Network Management Interface (gNMI), and OF support [PANTHEONtech, 2021b].

ONAP

Open Network Automation Platform (ONAP) is an open-source platform hosted by The Linux Foundation initially released on the 20th of November, 2017. This platform is composed of several components that together offer a complete orchestration and automation platform. The SDN-C element, which is one of the many components, is essentially ODL with some extras [PANTHEONtech, 2022].

ONAP introduces other useful elements [ONAP Team, 2022a]: a network inventory to view and keep a history of network resources, services, and products; a policy framework that supports multiple policy engines; a Portal for differentiating user experience based on account roles. Each of these components is released as a docker container forming a microservices-based system. This allows scalability and flexibility because users can only deploy the components that they need. Besides automation, this platform also focuses on enabling and supporting policy-driven physical, virtual, or cloud network functions, 5G scenarios, and complete lifecycle management (design, setting up, operating). It allows service providers and developers to scale their network in an automated

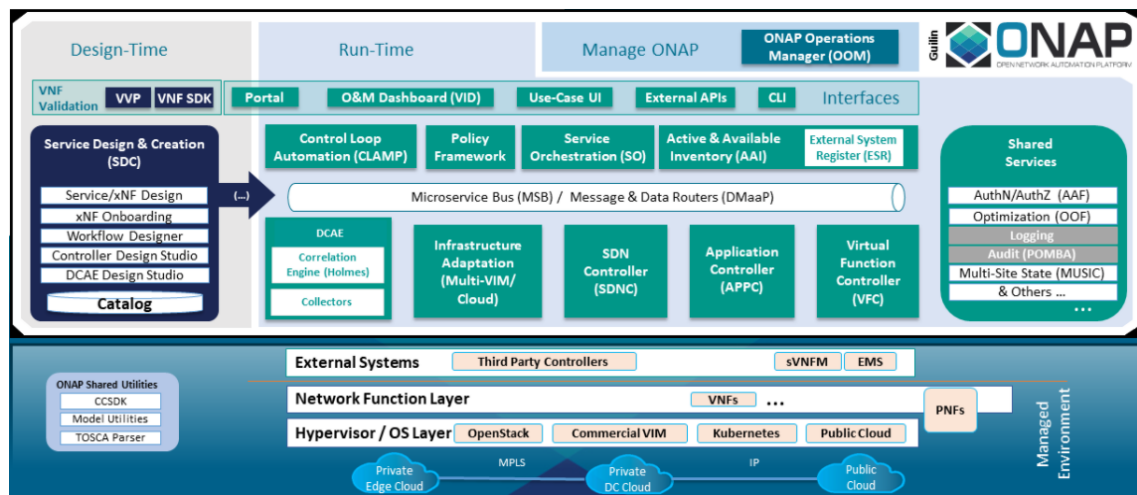


Figure 2.7: ONAP architecture [ONAP Team, 2022a]

manner by handling onboarding, orchestration, control, inventory, policy, and more [ONAP Team, 2022b]. Contributing organisations include AT&T, Samsung, Nokia, Ericsson, Orange, Huawei, Intel, IBM, and more.

Architecture ONAP’s architecture consists of two major frameworks, namely the Design-time framework and the Run-time framework, see Figure 2.7. The former provides service design while the latter handles service deployment and operations [ONAP Team, 2022a]. The service design component allows modelling of the resources and relationships that constitute services, and products and how they are managed. This component also contains policy design and implementation that guide the service behaviour. It enforces a set of rules defining control, orchestration, and management policies respecting some constraints. The service deployment is achieved by a policy-driven orchestration and control framework that provides automated instantiation of services when needed. Service operations are decided based on the service’s specified design and the behaviour monitored during its lifecycle. Collecting and evaluating event data is needed to deal with situations ranging from those that require healing to those that require scaling of the resources.

Interfaces ONAP provides a common set of REST APIs as a northbound interface. It supports the same north and south interfaces of the ODL controller, although with some alterations necessary to bring them to this project.

Ryu

Ryu is a bit different from the previous controller options presented because it is not a full-fledged SDN-C right out of the box, it is more like a tool kit to build an SDN-C. Ryu is Python component-based framework maintained by an active developing community. Their first commit goes back to the 6 of December 2011. With this product, a developer can create a platform complete with network management and control applications that fit specific purposes and network require-

ments. This is beneficial when high flexibility/customisation of the controller behaviour is needed or when certain control mechanisms of off-the-self controllers aren't necessary thus removing overhead [Ryu Team, 2017].

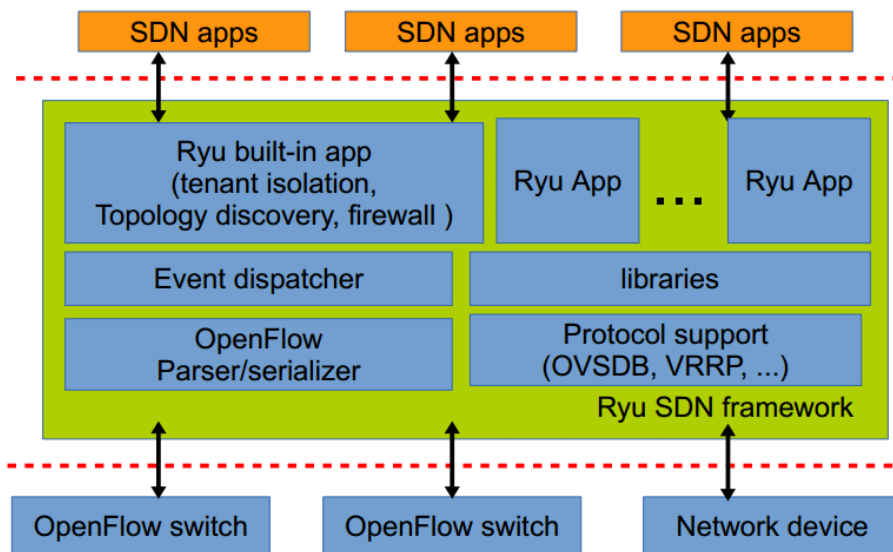


Figure 2.8: Ryu architecture [Irawati and Nuruzzamanirridha, 2015]

Architecture Ryu has a centralised architectural model. At its core, developers can create network management and control applications according to their own needs by using the provided components, e.g. OpenStack quantum, firewall, and topology discovery, see Figure 2.8. Developers can also find some ready to use applications. The SDN applications are used to perform analytics, run algorithms and perform monitoring as a way to orchestrate the network using the controller [Islam et al., 2020]. Ryu does not have an inherent clustering ability like ONOS or ODL [Farzaneh Pakzad from Aptira, 2021]. This controller offers support from integration with Snort, a popular Intrusion Detection System (IDS), and GUI access to users [Faucet Organisation, 2022].

Interfaces Ryu core provides north and south interfaces to communicate with the business and logic applications and to communicate with the physical and virtual devices of the network, respectively. It supports REST APIs, API for Quantum, and user-defined API via REST or RPC as a northbound interface. On the southbound interface, it offers multiply protocols for managing devices, such as OF (1.0 - 1.5), NETCONF, Openflow Management and Configuration (OF-Config), and P4 [Farzaneh Pakzad from Aptira, 2021; Islam et al., 2020].

2.1.4 Comparison

This section aims to compare some characteristics of the SDN-C previously presented. Table 2.1 summarises the comparison between the SDN controllers with

regards to type of architecture, the diverse north and south interfaces, the support for clustering, among others. We focus on technological and commercial characteristics as well as target scenarios of each controller utilisation. Some of the table fields are empty due to a lack of information or misleading statements that made us unsure.

Table 2.1: SDN Controller Comparison

	ONOS	ODL	Light.io	ONAP	Ryu
Programming Language	Java	Java	Java	Java	Python
Platform support	Windows, Linux, Mac OS	Windows, Linux, Mac OS	JVM	Docker containers	Ubuntu 16.04 or higher
Microservice Deployable	Yes	Yes	Yes	Yes	Yes
Architecture	Distributed	Distributed	N/A	Distributed	Centralised
Northbound	REST API, RESTCONF, gRPC	REST API, RESTCONF, gRPC	RESTCONF, Client lib in Java, Python or Go	REST API	REST API, API for Quantum, User API (RPC)
Southbound	OF 1.0-1.5, P4, NETCONF, TL1, SNMP, BGP, PCEP	OF 1.0, 1.3, 1.4, P4, NETCONF, SNMP, LISP, BGP with link , state, PCEP	OF, gNMI, SNMP, BGP, NETCONF	ODL Interfaces	OF 1.0-1.5 with Nieira Extensions, OF-Config, P4, NETCONF
GUI	Web Based	Web Based/DLUX	N/A	Yes	Web Based
CLI	Yes	Yes	Yes	Yes	N/A
Clustering Support	Yes	Yes	Yes	Yes	No
Initial Release	05 December 2014	5 February 2014	11 May 2018	20 November 2017	9 December 2011
License	Apache 2.0	Eclipse Public 1.0	N/A	Apache 2.0	Apache 2.0
Intent Based Networking	Yes	Yes	N/A	Yes	No

From our analysis, ONOS and ODL are major competitors: are maintained by industry respected companies, their partners are well-known network providers, support useful SDN protocols, and are popular choices for management solutions in the literature [Cohen et al., 2020; Gilani et al., 2020; Jang et al., 2017].

Light.io and ONAP were born out of the ODL project: the former focus on Apache Karaf's independence and the latter on an automatic and complete solution for orchestration.

The last controller analysed, Ryu, can be interpreted as a toolbox from which controller functionality can be built. Ryu is the less complex of the five solutions presented but this also reflects its lack of utilisation in real-world scenarios. The missing functionality out of the box, the single point of failure caused by its architecture model, and the missing scalability and reliability support make this a controller not fit for utilisation in scenarios with more exigent requirements (e.g smart-cities, 5G drivers, mission-critical).

Expanding on the comparison of ONOS and ODL, both solutions support distributed deployments to achieve reliability and scalability in emerging networking scenarios, e.g smart-cities. In terms of performance, these solutions don't differ much. For example, ODL is faster in flow processing and ONOS has a higher flow response rate (flows/ms) [Cohen et al., 2020].

Although ONOS and ODL support many equivalent features and advantages

and can work with the same SDN related protocols and technologies, their origin and the initial focus were different. ONOS was designed motivated by carrier networks and ODL was primarily datacenter focused [Salman, Ola and Elhajj, Imad H. and Kayssi, Ayman and Chehab, Ali, 2016]. ONOS has been employed by network operators to enable innovation at 5G networks through the paradigm of Software Defined Radio Access Networks (SD-RAN) [Open Networking Foundation, 2022c] and Software Defined Core (SD-CORE) [Open Networking Foundation, 2022b]. On the other hand, ODL has been employed for managing networks in the cloud with different virtual network functions [Mirantis Blog, 2022].

Another indicator of controller evolution and functionality coverage is that ONOS didn't have support for OpenStack and ODL didn't have intent based programming. They seem to slowly be incorporating popular functionalities of the other SDN-C.

Another aspect to consider is that some authors argue that the ODL has a steeper learning curve than ONOS due to the complexity of MD-SAL [Goransson et al., 2016].

2.2 SDN Related Protocols

As mentioned before, see Section 2.1.2, the separation between data and control planes is one of the most defining characteristics of SDN networks. Because of this, there was a need for a common language between SDN controllers and network devices. This led to the creation of many distinct southbound protocols over the years, some more relevant than others. This section aims to present some key protocols that became standards for southbound communications in SDN controllers.

2.2.1 Openflow

Openflow (OF) is an SDN southbound protocol created and managed by the Open Networking Foundation (ONF). This open-source project is now considered the standard for communications between controllers and devices of an SDN architecture having a lot of major switch and router vendors announced the intent of having OF support on their products [Open Networking Foundation, 2022a].

Since its first release in 2009, version 1.0, OF has had a lot of modifications and improvements. The work of [Latif et al., 2020] sums up nicely the evolution of the protocol. See Table 2.2 to see an enumeration of the main improvements over the previous versions and the reason behind modifications.

Giving a brief explanation of the OF behaviour, compliant devices contain **flow tables** that the controller can populate with table entries, see Table 2.3. By matching incoming packet headers or ingress ports to flow table entries each packet can be handled differently depending on its properties. The data plane devices inform the controller of the state of the network, both reactively (in response to

Table 2.2: Openflow evolution [Latif et al., 2020]

Version	Year	Major Features	Reasons of Extension
OF 1.0	2009	Single Table	-
		Fixed Matching Fields	-
OF 1.1	2011	Multiple Tables	Avoids Flow Entry Explosion
		Group Tables	Action Set to a Group of Tables
		VLAN and MPLS Support	-
OF 1.2	2011	OF extensible Match Using TLV Structure	Increased Matching Flexibility
		IPv6 Support	-
		Controller Role Exchange	Controller Scalability
OF 1.3	2012	Meter Table	Add Quality of Service
		Table-miss Entry	Provides Flexibility
OF 1.4	2013	Synchronised Table	Enhances Table Scalability
		Bundle Supports Group	Modifications Enhances Switch Synchronization
OF 1.5	2015	Egress Table	Allows processing on output ports
		Scheduled Bundle	Extends Switch Synchronisation Further

events) and proactively. This way, the control plane can add, update or delete table entries at run-time.

Table 2.3: Openflow - flow table example

Match Field	Priority	Counters	Instructions	Timeouts	Cookie	Flags
Ingress Port	99	Packets	Goto Table 2	300 sec	{controller issued identifier}	OFPPF_SEND_FLOW_REM
Ingress Port + Packet Headers	4	Bytes	Out Port X	500 sec	{controller issued identifier}	OFPPF_CHECK_OVERLAP
...

Flow tables entries contain multiple fields with different purposes: **match fields** are used to match incoming packets; **priority** gives us matching precedence, **counters** are updated when packets are matched to a given entry; **instructions** modify the pipeline processing or action set to be applied in each packet; **timeouts** gives the maximum amount of time before a flow is expired; **cookies** are a data value attributed by the controller to identify a flow entries; **flags** are tags that customise behaviour around the flow entry (e.g OFPPF_CHECK_OVERLAP check for overlapping entries first, OFPPF_SEND_FLOW_REM send flow removed message when flow expires or is deleted);

The evolution of the OF protocol affected the components of compliant devices. In the latest OF release, version 1.5.1 [Open Networking Foundation, 2015], a

switch can have one or more flow tables for pipeline packet matching, a group table for representing additional methods of forwarding, a meter table to implement Quality of Service (QoS) like rate-limiting and OF channels to maintain communication with controllers.

In Appendix B, we explain the role of these components and do a walk-through of the packet pipeline of OF 1.5.1. Furthermore, we explain the behaviour and messages of the interface between OF enabled devices and SDN controller, the OF Control Channel.

Openflow Limitations

Different issues of the OF protocol came to light throughout the years. The evolution of the standard resolved some of them, introduced more quality of life features, and offered more functionality to developers.

However, some concerns could be fixed because they were part of the way the protocols were built to function. By having predefined fixed headers in its flow tables, a preset number of methods to modify packets, and somewhat of a stiff pipeline, OF ends up being a protocol with inflexible implementations.

Another limitation of OF is that with each new version there are alterations to the pipeline and table headers. The introduction of new features leads device manufacturers to refresh their product chips logic to comply with changes. This is expensive for manufacturers, because they need to update to the newest version, and slows down the refresh cycle of the protocol because device vendors might want to remain with older devices for more extended periods [Zanna et al., 2019].

As a way to fix these issues, from the development community efforts emerged a paradigm that allows programmable southbound protocols: data plane programming. We present the notions of this paradigm in the next section.

Another OF limitation is the fact that it is not ready to comply with wireless scenarios. It can't assure additional features required in networks with wireless connections (e.g. wireless channel selection, interference mitigation, mobility management). Researchers end up modifying the OF to end up with a customised version of the protocol that works in their scenarios [Jang et al., 2017].

Hopefully, in the next iteration of the protocol, we can start to see support for the technologies.

2.2.2 Data plane programming and P4

The lack of flexibility offered by OF motivated the emergence of data plane programming. By having predefined fixed headers in its flow tables, OF's pipeline is locked.

Data plane programming has the solution to OF's inflexibility problems. By allowing custom headers, implementations using data plane programming can be

completely tailored to the application’s needs because users can program only the required protocols. The full flexibility of packet processing enabled users to program new protocols and design unique applications faster than having to implement these solutions into silicon [Hauser et al., 2021].

One of the data plane programming languages that increased in popularity and became a standard was Programming Protocol-Independent Packet Processors (P4). Its latest version, P4₁₆, was standardised recently being released in early 2017, see Table 2.4.

Table 2.4: P4 specification history [Hauser et al., 2021; Open Networking Foundation, 2020]

Date	Version	Feature
May 2013	-	initial idea and the name “P4”
September 2014	P4 ₁₄	P4 ₁₄ v1.0 release
May 2017	P4 ₁₄	P4 ₁₄ last release with v1.0.4
April 2016	P4 ₁₆	first commits
May 2017	P4 ₁₆	specification release

P4₁₄ language was discontinued in favour of P4₁₆ because it fixed the limitations of the previous version. The lack of means to describe various P4 targets and architectures models, weak support for program modularity, and mislaid strict typing, expressions, and nested data structures were some of the issues worth mentioning [Hauser et al., 2021].

In Appendix C we provide the materials to fully understand the behaviour of P4 programs and the pipeline that packets take. We even provide a template program of a simple routing solution.

Data Plane Programming Advantages

Data plane programming is a paradigm with potential. It allows programmers to think programmatically instead of thinking protocols. Programming a custom packet pipeline with custom headers and functions assures that the switch is loaded with only the required implementations.

Switch resources are more effectively used. This paradigm also introduces some advantages of programming languages like software reuse, data hiding, library creation, and easy debugging.

2.2.3 Other Protocols and Languages

This section aims to shed light on the purpose of some protocols that are usually found in SDN environments.

A common appearance is the NETCONF protocol which is used to more easily modify the configurations of network devices. While OF is used to manipulate the forwarding tables of devices, NETCONF can install, manipulate, and delete the configuration of network devices. It uses an Remote Procedure Call (RPC) based mechanism to facilitate communication between a “client”, like a script or application typically running as part of a network manager, and a “server”, typically a network device [Internet Engineering Task Force (IETF), 2022a].

Another protocol emerged because of NETCONF, the RESTCONF protocol. It exposes REST-like end-points to allow access to NETCONF functionalities (e.g. it uses JSON instead of XML and uses HTTP methods over XML tags) [Internet Engineering Task Force (IETF), 2022b].

Another prominent sight is the usage of the data modelling language Yang. It is used to model configuration and operational features in the NETCONF and RESTCONF protocols as well in the P4 language [Internet Engineering Task Force (IETF), 2022c].

2.3 Network emulation

Network emulation is a popular way to create a realistic virtual network as a way to test our implementations without having to buy expensive equipment or needing to mount a real testbed.

Programmers can declare the target network specification in an emulation tool to have the program creates an environment whereby real devices, applications, products, and services can be connected to the emulated network. Normally, emulation tools also give the option of connecting virtual elements which makes this approach much more convenient compared with real-world implementations.

In this section, we discuss some popular network emulators that can be used with SDN enabled topologies.

2.3.1 Mininet

Mininet is a python open-source emulation tool that can run on native Ubuntu or in a container environment. It has great compatibility with SDN scenarios as it supports OF and P4 protocols and also works with real SDN-C, like ONOS, ODL, and others [Arahunashi et al., 2019]. Mininet creates a virtual network, running a real kernel with hosts, switches, controllers, and links [Mininet Team, 2022a]. It can offer Open vSwitch (OVS) software in the devices (with support for OF, NetFlow, sFlow, IPFIX, and others), or Behavioral model version 2 (BMV2) (BMV2) that supports the V1Model target architecture for the P4 language;

Users can create virtual topologies using one of the presets that Mininet offers, e.g. linear topologies, tree topologies, or opt to build a custom layout using the python Mininet library.

Is common to see literature work resort to emulators like Mininet as a way to test frameworks and mechanisms [Song et al., 2017]. This is because building scenarios with virtual devices is less cumbersome when compared with real hardware. When a change in the topology layout/configuration is needed it can be done by changing some input parameters or lines of script code. This approach is also cheaper because researchers don't need to buy expensive hardware to build their topologies, that with multiple devices would get expensive very quickly [APS Networks, 2022; Edge-core Networks, 2022].

2.3.2 Mininet-WiFi

Mininet-WiFi is a fork of the Mininet emulation tool with extensions that make it able to emulate wireless scenarios. This version adds support for wireless stations and access points, device mobility, different 802.11 standards, mesh and ad-hoc layouts, different propagation models, and other features to enable diverse wireless scenarios.

Since Mininet-WiFi adds new functionality on top of the Mininet tool, everything that functions in Mininet also works in Mininet-Wifi [Mininet Team, 2022b].

Mininet-WiFi can simulate the wireless medium using two approaches: Traffic Control (TC) or `wmediumd`. TC was the first approach adopted in Mininet-WiFi for simulating the wireless medium. It is the same program used to configure the Linux kernel packet scheduler. TC is used to apply values for bandwidth, loss, latency, and delay in Mininet-WiFi [Mininet Team, 2022b].

The other option, `wmediumd` approach, uses the kernel module `mac80211_hwsim`. Node positions and ranges are simulated by assigning stations to other stations or access points that make up the wireless links. The Mininet-Wifi documentation depicts `wmediumd` as the superior mode to simulate wireless medium because it has more advantages, e.g. it isolates the wireless interfaces from each other and is a requirement for mesh or adhoc networks.

Mininet-Wifi is also a popular choice in the literature to emulate networks/topologies with wireless settings, mobility patterns, among others [Fontes et al., 2015; Gilani et al., 2020].

2.4 Traditional Monitoring Protocols

Network monitoring goes way back as an activity necessary in the first networks. Way before the introduction of the SDN paradigm, network administrators needed to have information on the behaviour of the network.

These traditional monitoring techniques evolved and we see today some of the more mature ones in SDN literature as a way to aid in network monitoring. These tools provide sampling and flow measurements, that are challenging to accomplish using only OF [Suárez-Varela and Barlet-Ros, 2017].

In this section, we present some relevant protocols that, although were not designed to work in SDN environments, are still used to enhance monitoring techniques in this type of scenarios.

sFlow

sFlow¹ is a standard for monitoring traffic in traditional networks. The sFlow collection technique is sampling based. This allows the collecting process to be decoupled from the routers and switches forwarding logic. Thus, this design is ideal when we need to worry about scaling and monitoring at Gigabit speeds.

The sFlow technique requires two different entities: **sFlow Agent** to sample the traffic of a device, this agent can perform two types of sampling; *Flow samples* by randomly sampling packets of switches flows and *Counter samples* by sending the device's interface counters between defined polling intervals; **sFlow Collector** to receive the data from the agents; The gathered sampled information is then sent to the collector as a *sFlow datagrams* to the *sFlow collector* that analyses and reports on network traffic.

sFlow has been adopted into the SDN paradigm solutions to support monitoring activities:

In 2017, researchers [Jang et al., 2017] compared their monitoring framework against the capabilities of native OF and sampling-based monitoring of sFlow as a way to evaluate the implemented framework.

In 2018, researchers [Fawcett et al., 2018] utilised sFlow as the monitoring preference in appropriate scenarios was a way to reduce the volume of monitoring traffic in their multi-level monitoring SDN framework.

NetFlow

NetFlow² is a flow export protocol proprietary of Cisco. It aggregates packets into flows and sends them to a collection point for storage and analysis.

This system is composed of three different components. **Flow exporters** are network devices that support NetFlow and collect traffic statistics to export them to collection points as flow records. **Flow collectors** are responsible for receiving flow records from flow exporters and storage and pre-processing of data. **Analysis Application** evaluates the flow records to avoid flow table entry explosion [Suárez-Varela and Barlet-Ros, 2017].

The interest in NetFlow led Internet Engineering Task Force (IETF) to build an open standard, Internet Protocol Flow Information Export (IPFIX). IPFIX³ is the successor of NetFlow. It is based on NetFlow version 9 and follows IETF standards [Trammell and Boschi, 2011]. With some exceptions (extra fields added), IPFIX has identical formats to the ones of NetFlow.

¹<https://datatracker.ietf.org/doc/html/rfc3176>

²<https://www.cisco.com/c/en/us/tech/quality-of-service-qos/netflow/index.html>

³<https://datatracker.ietf.org/doc/html/rfc7011>

Chapter 3

Research Projects & Related Work

In this chapter, we document external projects that affected the scope and direction of our work. Section 3.1 introduces research projects where we are involved. Section 3.2 provides information on recent European Software Defined Network (SDN) management projects that steered our requirements. Section 3.3 describes relevant related work in terms of SDN network management platforms and monitoring solutions.

3.1 Ongoing projects

This section presents research projects in which we integrated the research team and where this work contributed to their development. We provide the objectives of each project and relevant impact in the state-of-the-art. We also briefly introduce the ways our work added value to each one of them.

3.1.1 SNOB-5G

SNOB-5G ¹ is an international project that aims to research and develop wireless backhaul solutions for scalable 5G heterogeneous networks. Urban 5G services can especially benefit from this project since smart-cities have scenarios with mixing technologies and heterogeneous requirements, see Figure 3.1. This project accommodates the interoperability of these emerging urban services by promoting connectivity with a high-bandwidth capacity and latency requirements.

The SNOB-5G project expects to be impactful in several ways:

Cheaper backhaul can be achieved by enabling wireless technologies in urban infrastructures, thus avoiding the limitations related to the availability and installation costs of wired connections.

Enabling 5G emergent services is also within SNOB-5G scope. Unlocking 5G

¹<https://snob-5g.com/>

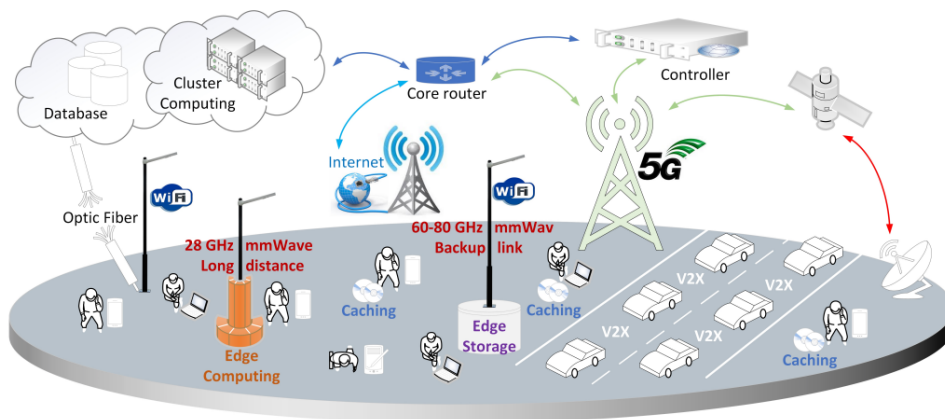


Figure 3.1: Traditional wireless communication networks encountered in 5G [Cohen et al., 2020]

based services (e.g intelligent transportation systems, vehicular-to-vehicular or vehicular-to-infrastructure communications, smart-city sensors, and industrial automation) demands handling its requirements. According to the International Telecommunication Union (ITU) 5G technologies can operate with peak datarates of 20 Gbit/s, low latency of up to 1 ms, and 10^6 more devices per km^2 than fourth generation technologies [International Telecommunication Union, 2015].

SNOB-5G backhaul solution manages these high demand 5G services by developing self-optimised, intelligent, and fault-tolerant networks.

In the ambit of SNOB-5G, [Cohen et al., 2020] brings network coding, through SDN, into a highly-meshed heterogeneous architecture, typical in 5G scenarios, see Figure 3.1. An architecture for this heterogeneous multi-source multi-destination network is proposed and a control model that uses multiple Software Defined Network Controller (SDN-C) is suggested to handle mesh communications and to improve fault tolerance. The authors also suggest the utilisation of Millimeter Wave (mmWave) as a technology capable of handling mesh connections. A decentralised multi-modules network coding model is then implemented to ensure reliable network communications and service degradation robustness. Each of the modules of the Adaptive and Causal Random Linear Network Coding (AC-RLNC) implemented collaborates and interacts with each other and with the SDN-C to perform data transmission This solution proved to offer reliability benefits in communications of highly interconnected meshed networks.

Highly resilient networks is another aim of the project. Having a self adjusting and intelligent autonomous network makes it able to work in mesh topologies, replacing traffic paths in case of node or link failure or discovery of a better suited path. The lack of a point of failure and a higher resilient network ensures the availability of sensitive requirements of some 5G services.

In the ambit of SNOB-5G, Abreu *et. al* [Abreu et al., 2020] proposes a framework to orchestrate Virtual Function Chains (VFC) service requests while maximising service resilience and availability. Firstly, a grammatical model is used to verify the correctness of the defined service chains and virtual function replicas re-

requested. Then two algorithms are presented for scattering the replicas through the network based on a prioritisation of availability: a formal mechanism using Integer Linear Programming model and a heuristic model based on Fluid Communities ideal to utilise in larger and more complex scenarios seeing that it consumes less time and resources than the last model in these environments. Finally, the platform was evaluated in a simulated scenario which concluded that higher values of resilience can be reached while orchestrating the VFC loads.

Security and Privacy is also a concern of the project. Intelligent backhaul nodes might utilise third-party applications to improve Quality of Experience (QoE) of users. These applications are leveraged using Multi-access Edge Computing (MEC) or Application Programming Interfaces (APIs), thus we need to be sure that only authorised application are used. Privacy-preserving techniques are also relevant due to the fact that a lot of services generate data from user inputs and data protection is necessary despite the need for user behaviour for network re-configuration and optimisation.

One of our focuses is to help advance SNOB-5G so that the project reaches its goals. Since both works share similar objectives we can contribute simultaneously: we helped in requirements elicitation and in delivery artifacts, gained know-how of emulation and containerisation approaches, and implemented metric collection in Open Network Operating System (ONOS). Furthermore, we are currently working towards developing an experimental scenario with multi-link and network coding for utility maximisation. The interfaces with the controller (e.g., using gRPC/REST) and the control logic are our responsibility.

3.1.2 MH-SDVANET

MH-SDVANET is a research project that proposes an SDN solution to orchestrate multihoming 5G vehicular networks. In this type of wireless network, also called Vehicular ad hoc networks (VANET), vehicles are equipped with On-Board Unit (OBU) to communicate with other vehicles and to communicate with road infrastructure, Road Side Unit (RSU). This paradigm fosters vehicle-to-vehicle and vehicle-to-infrastructure communications to enable new Use Cases (UCs), e.g. accident notification, navigation services, pedestrian protection, and truck platooning. The project aims to design and implement an SDN architecture for 5G multihoming VANET as a way to optimise the placement and migration of VFC services. Effective management of network resources is necessary to support the heavy requirements of these services: latency-sensitive, high throughput, high mobility, and technological heterogeneity are some common examples. To cope with these requirements, the SDN platform utilises different technologies to manage topology devices and to monitor network behaviour (eg. Openflow, Programming Protocol-Independent Packet Processors (P4), 802.11p, 5G, and mmWave).

The MH-SDVANET project addresses several key topics of VANET environments:

Service Handovers is essential in vehicular mobility environments. Users in movement might need to establish multiple new connections due to the previ-

ous one becoming out of range. Building effective handover algorithms while guaranteeing the quality of the user's experience is a challenging activity due to the small time frames of each connection.

In the ambit of MH-SDVANET, [Silva et al., 2021] research and develop an SDN architectural model to support seamless service handovers. The authors briefly explain the lack of handover optimisation for VANET scenarios and the reasons for its necessity. Their research presents related works of handovers in vehicular networks that adopt the SDN paradigm. Afterwards, two architectures for handover management are proposed and implemented, see Figure 3.2, and later evaluated against a popular SDNless IP-based solution (N-PMIPv6) using real vehicular equipment.

The results show that the less complex SDN solution outperformed the non-SDN solution with regards to handover time, average delays, robustness, packet losses, and network overhead enabling seamless handovers.

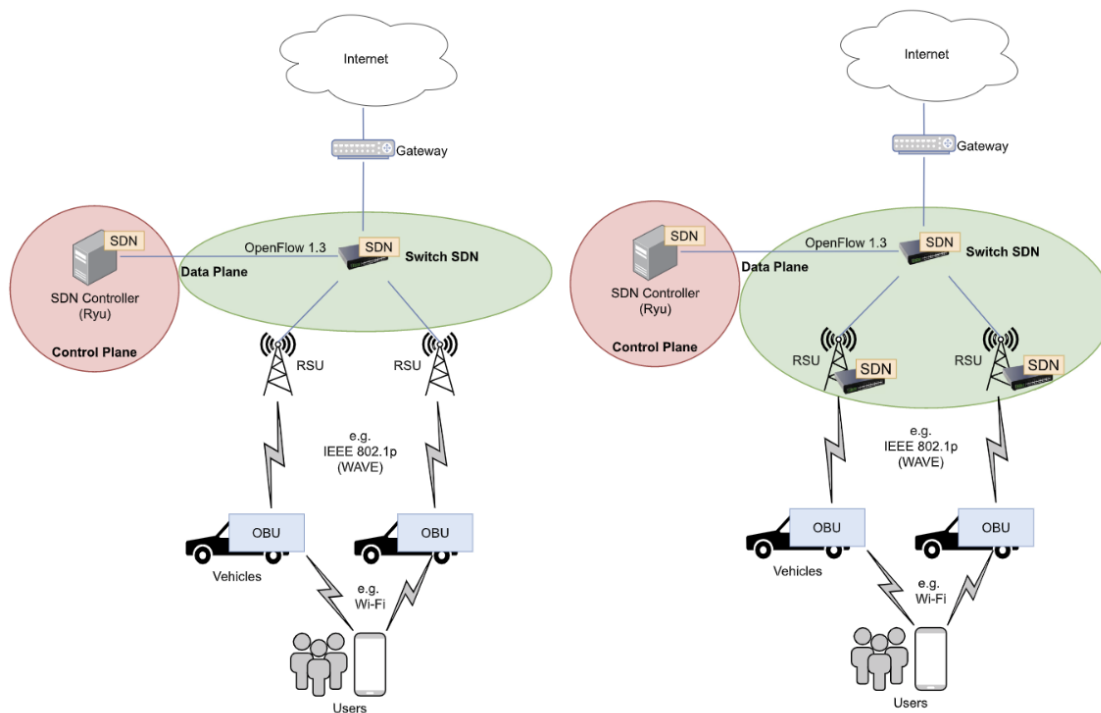


Figure 3.2: Proposed architectures for seamless handover in MH-SDVANET [Silva et al., 2021]

Our work and MH-SDVANET also share similar targets, thus there is work done that contributes toward the common goals of the project and this thesis: research and documentation of possible management solutions, consolidation of knowledge of southbound protocols, more notably P4, and research and development of multi-objective service fairness algorithms in ONOS are some of the more relevant examples.

3.2 Related SDN European projects

In this section, the focus relies on European research projects related to our work. Each project proposes management solutions for SDN networks but with different focuses depending on their objectives. This type of project is more extensive and might lead to more developed products and research works when compared with the average conference paper or journal. By learning the UC of each European project we gathered knowledge of the most pertinent functionalities to have in more extensive and complex management solutions.

The following list compiles information on the scope of the projects as well as their objectives:

- ENDEAVOUR² is focuses on SDN. It takes the perspective of the network operator and the users that connect to it. The objective was to research, develop, and evaluate an SDN architecture in the environment of internet service providers. The final product aimed to have monitoring platforms, tools for internal management, and a high speed data plane so it could scale to operate in the environment of internet service providers.
- Teraflow³ proposes a cloud native SDN-C focuses on security. It aims at providing flow aggregation, topology management, and forensic evidence in a micro-services architecture. This project also plans to develop a machine learning intrusion detection system to incorporate into the SDN-C.
- SDNmicro Sense⁴ is directed to critical scenarios, e.g. electrical power and energy systems. This project intends to provide an SDN architecture fit for local independent energy systems as a way to equip these environments with global system visibility, resilient cyber-defence systems, and to protect against data breaches.
- MAD-SDN⁵ is a young project that aims to use the programmable aspect of the SDN paradigm to handle the management of tons of data efficiently. By applying big data analysis techniques to SDN environments it is possible to troubleshoot network problems, detect anomalies, classify complex network traffic and optimise network performance.

3.3 Management solutions for SDN

In this section, we presented the relevant work about SDN management solutions we found in the literature. Our focus is on gathering knowledge of the more predominant techniques and activities to build a management solution.

²<https://www.h2020-endeavour.eu/>

³<https://www.teraflo-h2020.eu/>

⁴<https://www.sdnmicrosense.eu/>

⁵<https://cordis.europa.eu/project/id/893146>

In 2017, [Jang et al., 2017] proposed an SDN solution to monitor and manage Wireless Local Area Network (WLAN) scenarios called RFlow⁺. To enhance scalability, they implemented two different levels of network monitoring: short-term monitoring to perform more immediate actions (e.g. flow regulation) and long-term monitoring to perform eventual actions (e.g. limiting regular heavy loads, blocking slow scanning attackers). The short-term monitoring is performed by a local agent, close to network switches, that communicates with a global agent that resides in a higher layer close to the SDN-C. The framework is evaluated against native Openflow (OF) to compare accuracy and network overhead values. Experiments reported 5% standard error for short-term and less than 1% for long-term while consuming less network overhead compared with OF or sFlow.

In 2018, [Fawcett et al., 2018] presented a distributed SDN security framework that focuses on scalable detection and remediation of attacks called TENNISON. It provides differentiated monitoring capabilities: can perform lightweight monitoring across a large number of flows or Deep Packet Inspection (DPI) in smaller groups of flows. The framework provides options to work in conjunction with standard security tools (e.g. Snort, Bro), flow control using intents, monitoring enhanced by sFlow and IPFIX, and a API interface to register controller applications to regulate network behaviour. Fawcett *et. al* also built a security packet pipeline that prefixes new flow tables on the normal packet pipeline. However, several alterations were performed on the SDN-C instances of ONOS as a way to integrate it with TENNISON.

The team of [Joshi et al., 2018] developed into P4 enabled switches a system to detect and identify the cause of data microburst as a way to reduce latency and jitter. The small timescale didn't allow typical monitoring tools (e.g. sFlow, Netflow) to detect this burst so the authors built BurstRadar into the egress pipeline of each switch. Packets involved in microbursts are marked, cloned, and redirected to allow telemetry.

In 2019, [Menth et al., 2019] aimed to avoid Quality of Service (QoS) degradation in 5G mobile networks. To accomplish their goal, P4 was utilised to build a Activity-Based Congestion management (ABC) mechanism as a way to guarantee bandwidth-sharing congestion management: light users are protected against users that generate heavier data loads. The followup experimental evaluation proved that of the ABC mechanism was able to reduce the bandwidth utilisation of heavy hitters to enhance the QoS of lighter users.

[Song et al., 2017] identify SDN reliability challenges in the connection of data and control planes, the control path, and propose a management framework to address the observed reliability issues. Their work includes a new mechanism to elect a switch interface for controller connection (*Interface Selector*), an entity to store and expose SDN-C cluster configuration (*Cluster Information Manager*), reliability algorithms to detect and recover from failures (*Failure Manager*), and new control message classification model focus on scalability (*Control Message Orchestration*).

In 2020, *et.al* [Gilani et al., 2020] address routing difficulty in wireless meshed networks. The authors reviewed works in this area and compiled the major fea-

tures and characteristics. To respond to the common difficulties of routing traffic in these environments an architecture is presented: in the first routing phase the shortest path is calculated and is later optimised, with regards to delay and congestion, in a second phase; They also propose new protocol messages for Openflow for device-controller communication and utilise them in the proposed routing solution. OpenDaylight, Open vSwitch (OVS), and Mininet-Wifi are used to evaluate the suggested mechanism, with regards to throughput, packet drop ratio, and delay, against popular wireless routing protocols (e.g. OLSR, BATMAN). The experiments simulated representative problems of wireless networks (e.g. link and controller failure, node mobility, gateway saturation). Their experiments proved a success with their implementation outperforming the protocols used in the comparison.

In 2020, [Ndiaye et al., 2020] propose a modular and generic SDN management platform aimed at Internet of Things (IoT) scenarios. Their platform is built on top of IT-SDN⁶ which is an open-source platform that provides an SDN-C and neighbour discovery suitable for this wireless sensor networks. Their solution provides an application plane that exposes different management tasks to users (e.g. Monitoring of network state, task configuration, policy issuance) that are focused on IoT scenarios (e.g. sensor energy efficiency, adaptability of sensor tasks, efficient network expansion). These generic and modular components of the platform mean that it can be deployed in various IoT applications with heterogeneous environments. The authors evaluated their solution against comparable frameworks and found improvements in packet delivery, delay, control traffic, and energy consumption.

3.3.1 Summary

This section aims to compile the main takeaway of related works discussed throughout this chapter. It contains a list of each research work, Table 3.1, detailing their contributions in a summarised manner as a way to provide an overall view of the works analysed.

⁶<http://www.larc.usp.br/users/cbmargi/www/it-sdn/>

Table 3.1: Related work summary

Description	Keywords	Year
RFlow⁺: An SDN-based WLAN Monitoring and Management Framework [Jang et al., 2017] Scalable monitoring and management framework; Performs immediate actions (short) and eventual actions (long);	Framework, ODL, Wireless, Openflow	2017
Control Path Management Framework for Enhancing Software-Defined Network (SDN) Reliability [Song et al., 2017] SDN Framework with four modules:Interface Selector, Cluster Information Manager, Failure Manager, and Control Message Orchestration;	Framework, Floodlight, Reliability, Scalability, Openflow*	2017
TENNISON: A Distributed SDN Framework for Scalable Network Security [Fawcett et al., 2018] SDN topology coordinator with focus on security; Works with modified ONOS;	Framework, Security, Intents, ONOS*	2018
BurstRadar: Practical Real-time Microburst Monitoring for Datacenter Networks [Joshi et al., 2018] Takes a snapshot of all packets involved in a microburst	P4, Telemetry	2018
Implementation and Evaluation of Activity-Based Congestion Management Using P4 (P4-ABC) [Menth et al., 2019] ABC mechanism in P4 to enforce fairness	P4, Fairness, 5G	2018
Bringing Network Coding into SDN: A Case-study for Highly Meshed Heterogeneous Communications [Cohen et al., 2020] Network coding through SDN in 5G scenarios; Implantation of AC-RLNC;	SNOB-5G, Network Coding, Openflow, P4, NETCONF	2020
Resilient Service Chains through Smart Replication [Abreu et al., 2020] Framework to orchestrate VFC; Algorithms to scatter service replicas (Integer Linear Programming model and Fluid Communities Heuristic);	SNOB-5G, Framework,VFC, Reliability	2020
SDNMesh: An SDN Based Routing Architecture for Wireless Mesh Networks [Gilani et al., 2020] Two stage routing solution to combat difficulty in wireless meshed networks	Wireless, OpenDaylight (ODL), Routing, Openflow*	2020
SDNMM—A Generic SDN-Based Modular Management System for Wireless Sensor Networks [Ndiaye et al., 2020] Modular and Generic Framework focuses on IoT applications; built on IT-SDN platform;	Wireless, IoT, Modular Framework	2020
Exploring software defined networks for seamless handovers in vehicular networks [Silva et al., 2021] SDN architecture for seamless service handovers in VANET;	MH-SDVANET, VANET, Horizontal Handovers	2021

* modified version

Chapter 4

Research Objectives & Approach

This chapter documents the major goals of this work and the steps taken to reach them. Section 4.1 presents the main objectives of the dissertation, while, in Section 4.2 we discuss the plans to complete each individual goal, as well as measures taken to handle obstacles encountered.

4.1 Objectives

Our main goal is to develop a management and optimisation framework for the Open Network Operating System (ONOS) controller with novice forwarding mechanisms capable of operating in wireless mesh networks (WMN).

This framework keeps a collection of forwarding algorithms/approaches from where users can choose the preferred one to use. We provide our own approach to forwarding as a fairness focused heuristic: a formulation was developed by research colleagues and modelled as a multi-commodity-flow with 3 objectives; from this formulation was obtained a heuristic that works as a single-path min-cost-flow algorithm that is available in our framework.

Our approach introduces a fairness model taking into account three objectives: fairness of delay, fairness of reliability and fairness of energy, as a way to address the needs of WMN and 5G services. This is meaningful because now is very common to see mesh connections to improve reliability, devices with different energy requirements/costs, and heavier loads in networks. For example, fairness mechanisms can aid by maximising the utilisation of the available topology resources, allocating resources across different services or users, or handling concurrent services restrictions [Shi et al., 2014].

In the developed framework, users can register services to allow service flows to circulate the topology, so that when a packet of a registered service reaches the Software Defined Network Controller (SDN-C) it will find the best suited path, using the preferred forwarding mechanism at the time, and add the necessary flow rules into the devices.

Furthermore, the framework allows users to monitor the topology, e.g. devices, packets, flow tables, so they can see if their chosen forwarding configuration is having the desired effect.

Additionally, all this functionality is exposed through Representational State Transfer (REST) end-points. These end-points serve as an interface to external tools allowing them to act on the topology without having to worry about the underlying technologies and logic.

To reach our main objective, we defined the following goals:

- 1. Research standard features and activities of typical management frameworks**

Chapter 3 documents relevant research works done in this area that helped us build knowledge of possible implementation approaches for our target framework.

- 2. Implement the fairness centred framework in ONOS**

To have our custom fairness mechanism in ONOS we need to first program the heuristic logic as an application in the SDN controller. It is necessary to keep track of the topology layout, each device's energy consumption, the available services in the network, and the loss and delay of each link. With these values, we can now compute each of the link weights that are taken into consideration when the single-path min-cost-flow algorithm is running. The full logic of the heuristic and how we implemented it is documented in Chapter 7.

- 3. Implement REST end-points to allow integration with external tools**

To enhance the utility of our framework integration with external management tools is proposed. Thus, we provide a REST Application Programming Interface (API) with end-points so that users can configure the framework behaviour, e.g. preferred forwarding algorithm, register allowed services to circulate in the topology, and also monitor the information of network assets. In Section 7.2 we document the available end-points and the implementation steps of this API.

- 4. Detect service flows and manipulate flow entries**

Having a collection of forwarding approaches in ONOS, one of them being our fairness heuristic, we now need to be able to trigger the forwarding algorithm and program the flow tables of devices accordingly with the outputted path. We need to devise a custom packet processor to detect each of the packets that reach the controller and filter them to see if any of them are of registered services. After the preferred algorithm finds a path it is necessary to populate the Openflow (OF) flow tables of the affected devices with the appropriate flow entries. Both these endeavours are documented and explained in Section 7.3.

- 5. Evaluate the framework in a real scenario**

As a way to evaluate the frameworks and fairness algorithm performance, we emulate a real-world topology to orchestrate with the developed platform. Some adaptations to topology elements and services were required but the emulated network still resembles a scenario of possible deployment of our tool. This is relevant to measure the success of the project. In Chapter 8 we present the steps toward evaluation.

4.2 Approach

This section documents the methodology used and the decisions taken, according to the development approach and issues that appeared.

4.2.1 Research and Development Methodology

This section aims to provide the vision and methods to research and develop the management framework as well as the fairness mechanism.

Firstly, research on the state-of-the-art of network management solutions was conducted as a means to gain knowledge on what type of feature we could/should include in our framework, what were the more prominent functionalities and how they were implemented. Using search engines like Web of Science, Research Gate, and IEEEExplore, we searched for conference papers and journal publications related with our proposed platform (e.g., “network management solution”, “wireless Software Defined Network (SDN) management”) and with management activities of our interest (e.g. fairness, reliability, load balancing).

We filtered out the less relevant and poorly-constructed works using online tools like Scimago Journal & Country Rank¹ and Core²: the majority of reviewed papers were accepted in conferences rated as CORE B or higher, and Q2 or higher in case of journal publications.

Also, we were able to coordinate our efforts with the practical assignments of “Emergent Systems of Internet” class and researched how to implement a Programming Protocol-Independent Packet Processors (P4) solution to perform load balancing during the first semester.

After documenting the state-of-the-art we proposed a ranked list of requirements needed in our framework. The ranked list followed a MoSCoW technique to help us prioritise the framework functionalities and divide the work into manageable pieces. The requirements list was constructed by taking information from previously built artefacts, e.g., Use Case (UC) diagrams, European project review, and related work.

We opted to follow a **waterfall style** development methodology. More agile methodologies were considered but abandoned due to the necessity of having

¹<https://www.scimagojr.com/>

²<http://portal.core.edu.au/conf-ranks/>

related work compiled and clear requirement documents early on. Investigation of network management projects was essential to grasp all the possible features to include and their priorities in terms of implementation.

An **iterative waterfall model** is the most indicated style to use because the efforts of this work contribute to other projects and research activities that have shorter deadlines and need our contributions to be able to branch toward other areas.

As a way to validate the correctness of the framework in a non-exhaustive manner but without the risk of missing some major issue, we conducted black box tests, see Chapter 9. In this type of test, a routine is called with a set of input parameters. For each call, we observe the output and assess if it was according to what was expected, upon non-compliance we need to readjust the routine logic.

4.2.2 Planning

This section serves the purpose of presenting the evolution of the scope of this work and how we managed it taking into consideration the issues that we faced.

The initial objective for this work was to build a management and optimisation framework agnostic of SDN-C and protocols, with the ability to pug-and-play custom forwarding algorithms. See Appendix D to find detailed information on the initial goal of this dissertation.

From the feedback of the intermediary defence we needed to reduce the initial scope of this project: it was ambitious, had too many non functional requirements (that needed validation) and some of the activities from the first semester were leaking to the second semester, e.g. incomplete information in sections of Chapter 2.

So the first thing done in the second semester was reducing the scope of the project by removing features. Such reduction considered the requirements listing and classification using the MoSCoW methodology done in the first semester:

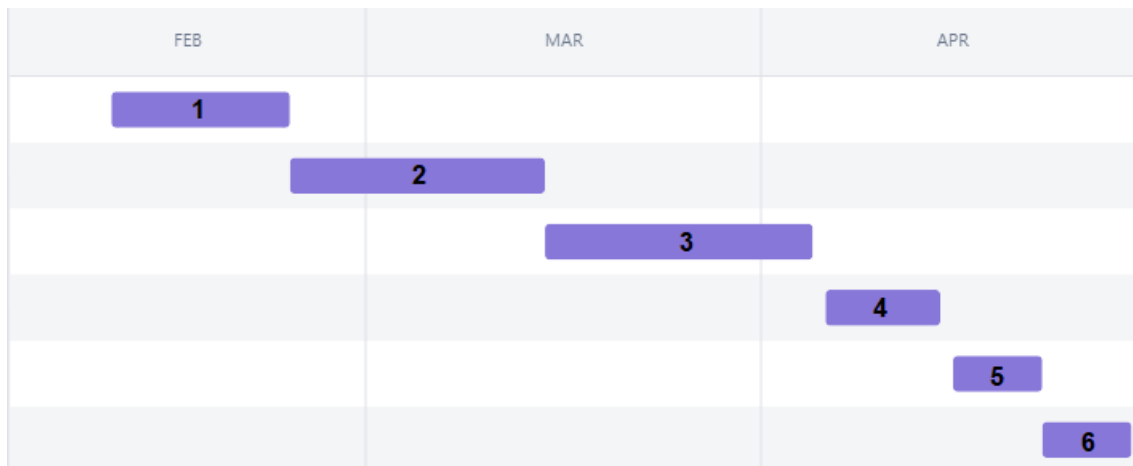
- **Focus on key management activities** We decided to focus only on key management activities related with **load balancing**, **fairness**, **Service Function Chaining (SFC)**, **device mobility** and **redundant connections**. From our research, see Section 5.1, these were common and handy activities to have in smart-city style networks. Furthermore, these features were the ones that fitted best in the interests of the research projects that we still integrate, see Section 3.1.
- **Reducing Non Functional Requirements** To save time we cut out or reduced some of the original Non Functional Requirements: **usability**, **platform support** and **extensibility** were removed completely; **agnosticism** was reduced to building applications for ONOS and for OpenDaylight (ODL) that performed metric collection (thus the framework would be controller agnostic). For **scalability** we only considered a topology with settings from a realistic network.

- **Removed User Management** The framework still holds value without the features of authentication, authorisation and user roles. The framework passes to be operated by a single user without requiring a login or account.
- **Reduced Algorithm Management** since we were planning to develop our custom fairness algorithm and test it against the provided k-shortest path algorithms of ONOS, we decided to keep both of them in the framework and reduced the options for managing algorithms. Now the framework presents the available options for forwarding and users select the preferred one to use, removing the options for creating/importing algorithms.
- **Reduced Asset Management** The ability to authorise and reject the participation of a device in the network is removed. We just kept the monitoring functionalities.
- **Removed Security related requirements** Requirements related to access control, authentication, or alerts of suspicious traffic were also dropped.

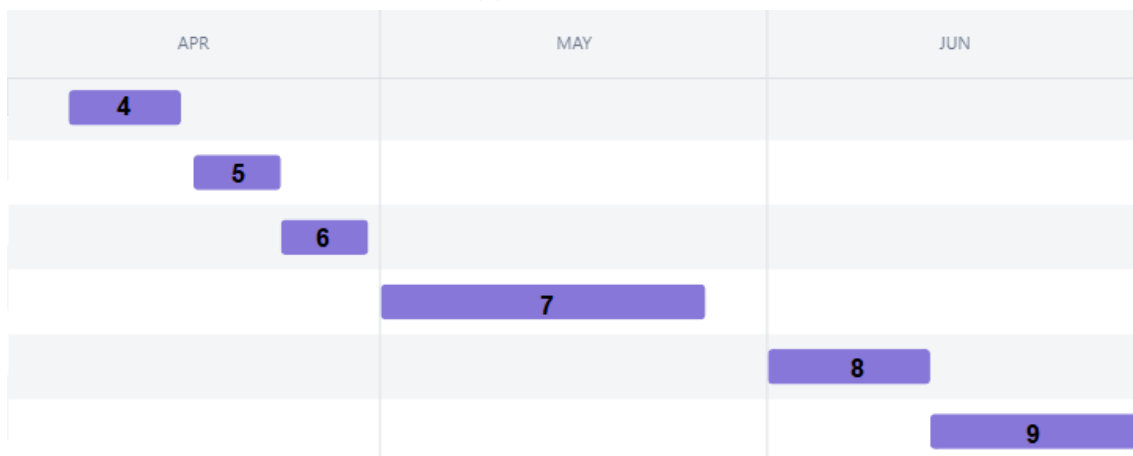
With this new scope, we devised the following plan to successfully complete the project by late July, see Figure 4.1.

The list of tasks includes:

1. **Document Corrections (13 days)** From the feedback of the intermediary defence, we had to readjust some sections of the document and finish others in Chapter 2 and 5. Some of these activities required further research so we planned accordingly.
2. **Experimental Topology (20 days)** When devising this plan, from the research and experiments carried out in the first semester, we knew that Mininet-Wifi had classes and routines that could help us build our desired scenario for the experimental evaluation. However, we still needed to devote more time to fully understand Mininet-Wifi: have stations connected to multiple Access Point (AP), introduce mobility, choose between *mesh* or *adhoc* connections, choose between *TC* or *wmedium* to simulate wireless properties, among other aspects.
3. **Fairness Heuristic Implementation (20 days)** With the mathematical calculations of the formulation and heuristic taken care of, which was not our responsibility, it is then our obligation to implement the heuristic logic into the ONOS controller. It involved gathering information about the three objectives and calculate the current link weight to use in the heuristic. Also, we wanted to see if there was any Java library that could help us accelerate the implementation of the single-path min-cost-flow algorithm and how we may integrate it with ONOS.
4. **Custom ONOS Implementation (8 days)** Implementation of custom weights into the ONOS forwarding algorithm was needed so that we could compare our fairness centred forwarding solution with the algorithms supported by ONOS.



(a) First months



(b) Following months

Figure 4.1: Gantt Chart for the second semester

5. **Flow Detection (6 days)** It is required to trigger the forwarding mechanism that allows us to obtain the solution path. For that, we needed to figure out how to reactivity detect the packets in the controller so that we can run the default forwarding algorithm.
6. **Inserting Table Entries (6 days)** With the solution path that came out of the forwarding algorithm, it is necessary to parse it and then install the necessary OF rules in the affected devices.
7. **Experiments & Paper (26 days)** The team also wanted to submit a paper showcasing our fairness mechanism theory, how we would implement it, and how we planned to evaluate it to show its advantages. However, due to health complications of the main researcher responsible for the formulation, the mechanism wouldn't be ready in time. So, we adjusted and planned to submit a paper to ICNP22³ at the end of May, showcasing experiments comparing our fairness mechanism with the algorithms that ONOS provides, and results reporting the advantages of our approach.

³<https://icnp22.cs.ucr.edu/>

8. **REST API (13 days)** To allow integration with external tools we needed to add a way for users to configure and also monitor the topology behaviour. To complete this task we decided to build REST end-points to expose these functionalities. We needed to find a suitable Object-Relational Mapping (ORM), an adequate web server, and implement the logic for the required REST verbs.
9. **Engine for Plug-and-Play Algorithms (16 days)** To satisfy the non functional requirements of *modularity* and *agnosticism* in the final version of the framework the algorithms needed to run independently of the SDN-C. This was to allow compatibility of the framework with any underlying SDN-C technologies. This meant that between the paper delivery and the final dissertation deadline we needed to migrate the algorithms to another entity, the “Algorithm Engine”, and figure out how to communicate between the controller and this new entity.

4.2.3 Issues and Readjustments

Following the adjustments done after the intermediary defence, we proceeded with the planned activities. However, during the following months, we faced several issues that compromised the deadlines of the estimated activities and that made us adjust the scope of the project several times, see Figure 4.2.

We also colour coded each activity to facilitate analyses: **green** activities didn't overspend the estimated time and were finished on time; **light green** activities didn't overspend the estimated time but were performed earlier than estimated; **blue** activities didn't overspend the estimated time but had their milestones shifted; **yellow** activities took more time to complete than initial estimated; **red** activities were abandoned, and thus remain uncompleted; grey activities we didn't estimate but they appeared and we had to finish them.

At the beginning of March, we were having issues with Mininet-Wifi, more specifically, trying to connect a single station to multiple AP, which was necessary to conduct the planned experimental activities. Unable to find a solution, we readjusted the experimental setup to use only one connection between station and AP.

We proceeded with the task “Topology Setup”, and at the end of March detected another issue while trying to test how the topology would react to the addition of custom flow rules: after removing the ONOS provided application “Reactive Forwarding”, that automatically install rules in the devices, and populating the devices with custom flow rules that followed similar syntax, the packets weren't being forwarded between switches. After some research, we found that the “Reactive Forwarding” application also manages ARP requests and replies and that we needed to develop an application with similar logic, the “Custom Arp Handler”.

Because of these setbacks we decided that it would be best to deliver the dissertation in September, to compensate for the lost time resolving these issues, and just

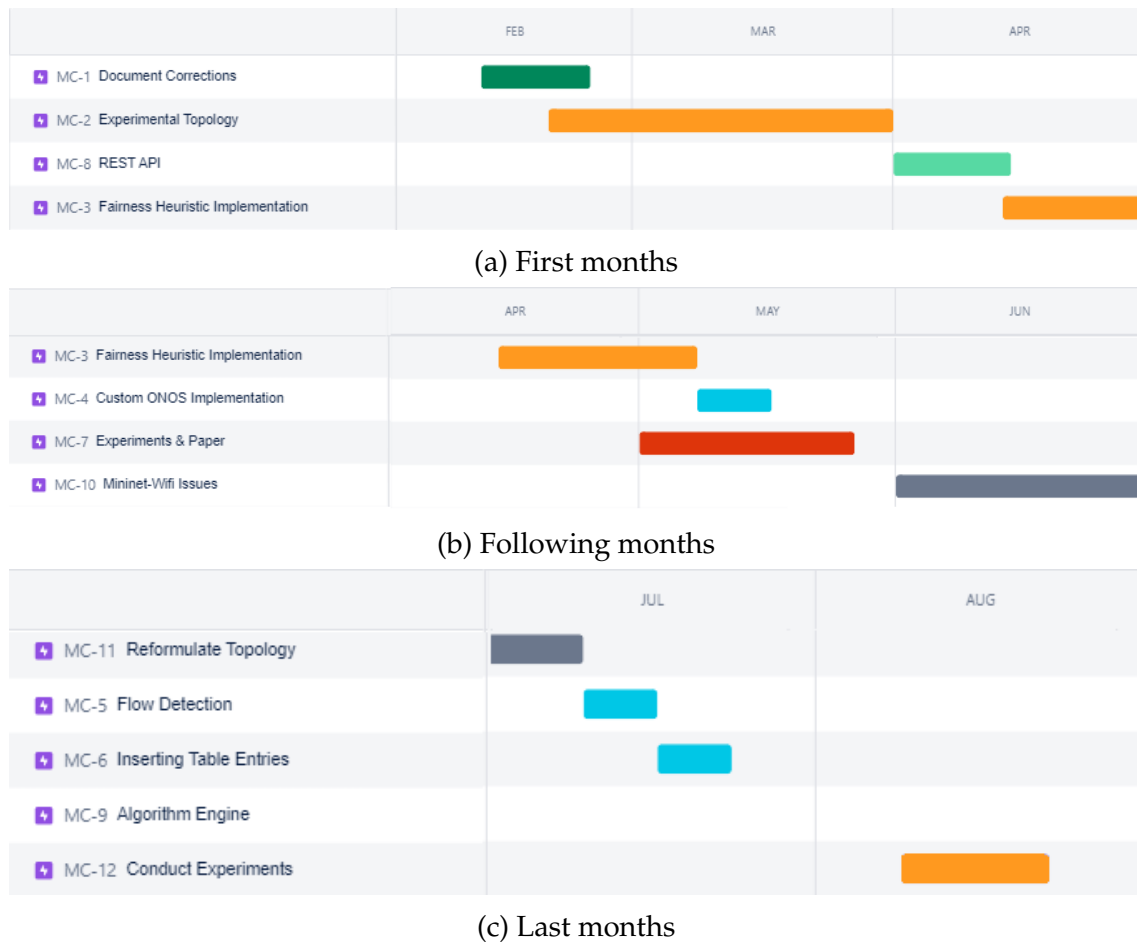


Figure 4.2: Activities performed in the second semester

shift the remaining activities.

At the beginning of April, before starting the “Fairness Heuristic Implementation” activity, we were tasked to provide the formulation author, researcher Noé Godinho, with information of the topology so that he could validate the formulation/heuristic: devices, links, delay, energy consumption, observable loss, bandwidth, service flows and output path of the heuristic. This data was necessary so that Noé could run a solver to know-how efficient the solution of the heuristic was (that gives a close to optimal path in near to real-time), in comparison to the solution of the formulation (that gives the optimal path but is not useful in a live setting).

Due to the amount of data that was necessary to parse and deliver, we made the decision of anticipating the “REST API” task as a way to provide this data through conventional REST end-points and postpone the implementation of the heuristic in ONOS.

In the middle of April we initiated the task “Fairness Heuristic Implementation”. We found the Java library Jgrapht⁴ that had the class *CapacityScalingMinimum-CostFlow()* which helped us to accelerate the heuristic implementation. While validating the Jgrapht behaviour, we found that it was dividing flows between mul-

⁴<https://jgrapht.org/>

multiple paths as a way to use multiple cheaper paths. With the desired behaviour, the algorithm, would use the cheaper path that could handle the whole flow at once. To avoid this we needed to perform some mathematical adjustments, see Section 7.3.1;

Moreover, due to the lack of clear instructions in the ONOS documentation we struggled to figure out how to import Java libraries into controller applications, and ended up performing some blind tests until we figured it out.

After concluding the “Fairness Heuristic Implementation”, and thus having the fairness algorithm implemented in ONOS, the next step was related with the “Custom ONOS Algorithm”. After discovering how to overwrite the default classes, we built a custom link weight for the k-shortest path algorithm of ONOS.

While still struggling with the implementation of the heuristic, at the beginning of May, we started to write the conference paper. We documented the objectives and motivations of the work and described the formulation/heuristic, as well as the experimental setup. The only thing missing was to conduct the experiments and discuss the results.

During this time, we also found a GitHub project, `mineevents`⁵, that provided a tool to define iPerf events in mininet networks. This is useful to generate the traffic of the services that exist in the network. We modified this tool and added some features: introduced our custom topology, support for Mininet-Wifi, pause feature, and ability to work with multiple event files. With these adjustments, we were ready to perform experiments in a flexible fashion.

However, due to the issues faced earlier we only finished implementing the forwarding algorithm in the middle of May, which was close to the paper deadline. Furthermore, we were not in the position of furnishing the “Flow Detection” and “Inserting Table Entries” tasks before the submission date. Nevertheless, we decided to try to experiment with a simple scenario and run the algorithm and install rules in the devices manually. If the results were promising we could still have time to submit the paper.

While trying to conduct the experiments we detected some problems with the emulated topology, e.g., undesired losses in traffic simulations between stations and AP, or mesh connections not reporting any packet loss.

Ultimately, the team decided to not rush the experiments and submit a paper that would probably be rejected. Instead, I was tasked to resolve the issues faced and resume the experiments afterwards.

Following the issues faced in May, at the beginning of June, we explore ways to overcome them, see Section 8.1.1 for a clean explanation of the problems and the ways we approach them. We tried a plethora of procedures to solve these concerns in Mininet-Wifi but in the end we were unsuccessful.

Near the end of June, we realised that the better option was to steer away from Mininet-Wifi to avoid further problems and to resume the experiments with the

⁵<https://github.com/cgiraldo/minievents>

baseline emulator Mininet. Although this emulator does not support wireless capabilities, it provides the characteristics to create an adequate and stable environment to test the performance of the fairness mechanism: OF enabled devices and hosts, adjustable link loss rates, bandwidth and delay (thanks to the Traffic Control classes), and flexible workload of traffic for different services.

Following this decision, came more scope adjustments. We steered away from the idea of a framework agnostic of SDN-C to a framework built on ONOS that provides a custom fairness mechanism:

- **Reduce management activities** Due to the issues with Mininet-Wifi, we removed the features related with **device mobility** and **redundant connections**. The framework supports management activities related with **fairness, load balancing, and SFC**.
- **Removing the Engine from Algorithms** This functionality would require us to implement the algorithm's logic in a new entity and to research effective ways to communicate between this new server and the infrastructure already deployed.
- **Reducing Non Functional Requirements** Extinguishing the "Engine for algorithms" support lead to the suppression of **agnosticism** and **modularity** characteristics of our framework.

Looking back, this project involved multiple technologies and communication interfaces. With each one, we faced their learning curves and "novice traps". The unfortunately common lack of documentation of this type of tools made us resort to perform blind tests as a way to figure out what configurations were causing the problematic and undesired behaviours.

4.2.4 Risks

In the first semester, we documented and classified risks that could arm the success of the project. For the ones that were deemed more problematic, we identified mitigation approaches. In this section we go through the risks that occurred and whether the mitigation plans were really effective.

Table 4.2 describes each entry and grades them by project's impact, occurrence probability and the time frame where they influence our advancements. Table 4.1 presents the description of the options considered for risk classification.

From the risks identified, **#1**, **#2** and **#3** occurred but we only had mitigation plans for the **#1**.

Mitigation for **#1** consists of dropping requirements that ranked low on the MoSCoW classification and/or reducing the quality of more important features. This approach was very effective: we were able to identify what functionalities were essential and the ones that could be dropped. Besides that, opting to deliver the

Table 4.1: Risk classification caption

Attributes	Options	Description
Impact	Low	Adjustments are made inside the budget, like the use of spare hours, or some optional low-priority requirements need to be dropped, without affecting the final product
	Medium	Adjustments are made inside the budget, like the use of spare hours, a meeting has to be scheduled and some optional requirements might have their quality compromised. Although we are still able to achieve the main functionalities
	High	Requirements need to be addressed. Meeting has to be scheduled and some of the functionality of the framework might be compromised
Probability	Low	<40% of chance of happening at least one time during the project
	Medium	<70% & >40% of chance of happening at least one time during the project
	High	>70% of chance of happening at least one time during the project
	Certain	Will happen at least one time during the project
Time Frame	Short	<2 week
	Medium	2-7 weeks
	Long	>7 weeks

dissertation in September gave us more time for an effective implementation. The weekly meetings also contributed to identifying problems, as soon as possible.

For risk #2, due to unexpected issues that slowed down progress and made us fail deadlines, we felt less motivated than usual: the fear of encountering more issues and having to blindly perform tests to figure out what was causing them made us more anxious. When we missed the paper deadline for ICNP22 we took some days off to relax and unwind because previously we were trying to rush experiments. On another occasion, reverting back to Mininet and abandoning Mininet-Wifi was a drastic change that made us uncomfortable. To gather feedback we participated in the seminar Rede Temática de Comunicações Móveis (RTCM) where we presented why we shifted to Mininet. The advice of the participants help us feel more confident in this change and provided useful feedback to better configure the emulated topology. Additionally, just talking and sharing my concerns with colleagues and research partners help us vent and understand that these issues are part of the process.

Risk #3 occurred in February, when Noé Godinho was unavailable due to health concerns and the formulation/heuristic wasn't ready. A publication regarding our proposed method for service fairness was to be published in February, but to mitigate this we simply postponed the paper submission to May, were we would have the mechanism implemented and results to discuss.

Table 4.2: Risks identification

ID	Description	Impact	Probability	Time frame
#1	Development is taking more than expected, which may affect the quality or content of the final delivery	High	Medium	Long
#2	Elements of the team become demotivated or burnout, so they become less productive	Medium	Low	Medium
#3	Elements of the team become sick so development might slow down or stop	Medium	Low	Short
#4	The main author isn't a good front end developer so it can be troublesome to do an elegant framework Graphical User Interface (GUI)	Low	Medium	Short
#5	The main author doesn't have a background in networking so some concepts might be misunderstood	Medium	Medium	Medium
#6	Poor architectural choices were made so framework development may not be compliant with requisites	High	Medium	Long
#7	Problems caused by bad design of the engine for algorithms might lead to progress stagnation	High	Medium	Medium
#8	The engine for algorithm interface is too complex so users might have a hard time using it	High	Medium	Long
#9	A new version of a protocol or SDN-C changes internal objects representation which led to our work being outdated	High	Low	Long

Chapter 5

Requirements Elicitation

The following chapter contains the requirements elicitation steps and resulting specifications. We extracted the management activities from relevant projects, recall Chapter 3, and then documented the functional and non functional requirements and also the implementation restrictions that affect our project. In the last section, we list and rank each individual requirement extracted from the artefacts produced.

5.1 Management Activities

As a way to make our management framework relevant and not just a niche case, it needs to be able to perform common and relevant network management activities. As a way to gather knowledge of the most pertinent functionalities to incorporate, we analysed the requirements of the Use Case (UC) in recent European projects and objectives of literature work, as per Chapter 3.

This analysis made us aware of the spectrum of management activities that could be integrated into our framework. In Table 5.1 we compiled the features of these projects as a way to extract requirements to our management framework.

To help visualise the areas of focus of these activities, we decided to categorise them into four classes:

- **A-Security**: This class groups activities that try to enforce data confidentiality, service availability, and data integrity. Protects the network against unwanted behaviour and undesired threats.
- **B-Service Degradation Avoidance**: The category represents activities of optimisation. Actions taken to assure Quality of Service (QoS) and Service Level Agreement (SLA).
- **C-Telemetry**: Reflects the process of recording and collecting the readings of an asset. Represents the overall monitoring actions.

- **D-Access Control:** Symbolises the regulation of elements that can participate in the network: authenticated users, authorised assets and valid traffic.

Some activities we classified with multiple classes because we found that their efforts were multi-disciplinary. Others we didn't attribute a class at all as a result of being an endeavour for which we previously didn't define a class for. Appendix A documents the terminology that was considered to classify the diverse activities and also to assess their relevance.

Table 5.1: Management activities

load balancing ^B	service fairness ^B	broadcast prevention ^{A,D}	inbound/outbound traffic engineering ^B
centralised management of network assets ^{B,C}	traffic steering ^B	support for service chaining	support for virtual networks
support for management algorithms ^B	support for self-healing methods ^B	high data volumes ^B	support isolation of resources
advanced blackholing ^A	traffic anomaly detection ^A	monitoring traffic flows ^C	monitoring control plane traffic ^C
support for centralised routing	support for redundant connections	traffic classification ^{A,B}	device flow manipulation
user access control ^D	support for mobility	risk assessment ^A	support for IPv4 and IPv6
support for network slicing ^{A,D}	assets access control ^D	—	—

From the plethora of management activities identified we add to prioritise and select the ones that would be integrated into the framework. This selection was weighted by the usefulness of each activity and prevalence in the state-of-the-art, but also by the interests of research projects that we integrate, see Section 3.1.

With this criteria, the focus was on tasks related with fairness and custom traffic flows: **service fairness** and **load balancing**, support for **management algorithms**, **traffic steering** and **flow manipulation**. Initially, other activities were also considered, but due to issues that lead to scope reduction, they were removed.

These activities are reflected in the final features of our framework: providing forwarding algorithms, with fairness concerns, that install custom Openflow (OF) flow rules in the forwarding devices of a topology, see Section 7.3

To support these forwarding mechanisms in our framework we needed to research and understand how their behaviour can be accomplished and what kind of network metrics are necessary to collect to enable them.

For this purpose, we compiled Table 5.2 with related approaches regarding load balancing and service fairness, where we enumerate the type of statistics we need to collect for orchestration operations.

Table 5.2: Metrics necessary for management activities

Description	Metrics	Activity
Fair Task Offloading among Fog Nodes in Fog Computing Networks [Zhang et al., 2018] Fog task offloading while minimising task delay with battery and circuit powered devices; Evaluates mechanism success with regards to fairness, delay and energy consumption;	Energy consumption, Battery life, Node tasks, Task delay, Computational power	Fairness
Fairness in Wireless Networks: Issues, Measures and Challenges [Shi et al., 2014] Defines and compares fairness models; Documents major fairness domains in wireless networks;	Energy consumption, Maximum energy consumption, Battery life, Node Tasks, Task delay, Computational power, Packets Dropped, Flow Rules, Bandwidth, Network throughput	Fairness
Computation Offloading and Resource Allocation in Mixed Fog/Cloud Computing Systems with Min-Max Fairness Guarantee [Du et al., 2018] Creates a low complexity algorithm that aims to minimise the maximum cost of delay and energy consumption; Evaluates the convergence performance of the proposed algorithm and the gains in terms of delay, energy consumption and number of energy improved nodes;	Energy consumption, Node Tasks, Task delay, Maximum task delay, Task computational complexity, Bandwidth, Computational power	Fairness
Max-Min Fairness and Its Applications to Routing and Load-Balancing in Communication Networks: A Tutorial [Nace and Pioro, 2008] Theoretical description with example algorithms to solve the problem plus discussion of possible applications;	Node tasks, Bandwidth	Fairness
Cloudlet Load Balancing in Wireless Metropolitan Area Networks [Jia et al., 2016] System to offload tasks while trying to minimise the maximum response time;	Node tasks, Task size, Task delay, Task migration time, Computational power	Load Balancing
Load-balancing algorithms in cloud computing: A survey [Jafarnejad Ghomi et al., 2017] Literature classification of load balancing algorithms;	Energy consumption, Battery life, Throughput, Task delay, Task migration time, Carbon emissions	Load Balancing
Energy-Efficient Load Balancing Ant Based Routing Algorithm for Wireless Sensor Networks [Li et al., 2019] Propose a routing discovery algorithm for wireless sensor networks that combines three novel components: an improved pheromone trail update, a heuristic for low energy route discovery and a new broadcast scheme for control packets;	Energy consumption, Battery life, Received Signal Strength Indication (RSSI), Link quality, Computational power	Load Balancing

5.2 Functional Requirements

This section address the functionalities that fall under the scope of our framework. We produce requirements artefacts to address the functional requirements of the target platform.

5.2.1 Use Case Diagrams & User Stories

We created UC diagrams, and describe them, to represent the flow of the main activities that different framework users can perform. User stories complement the diagrams as a way to give more detail on each of the diagram's activities.

A User Story¹ is a lightweight technique used to represent the features of a product. It is structured in a way as if a system user would create it. This is a common technique in agile development methodologies, where the entity that requested the product documents the desired features in an effort to facilitate the planning and discussion later with the development team.

We opted to use User Stories to document our framework features because in early meetings the main author of the framework and the advisor had difficulties expressing the features that were in the project scope and what functionalities they should have. The lightweight and less formal properties of User Stories help us to promote discussions and converge on the properties to implement in the framework.

The User Stories are structured as follows:

- **Description:** As a type of user I want goal so that result;
- **Acceptance Criteria:** Given that user situation when action performed then expected outcome.

The description helps define the features and the acceptance criteria characterises a successful scenario.

In the current version of the framework, there is no mechanism to perform authentication or access control, so any individual with access to the framework interface can perform all the available operations. Thus, there is only one UC actor:

- *Generic user* Performs all actions. User that can monitor and act on the network behaviour and configuration.

Algorithm Management

Users trying to orchestrate their network(s) have multiple forwarding algorithms available. By selecting one of them, the incoming traffic is forwarded following the rules of the selected algorithm. Figure 5.1 depicts the flow of the functionality related to algorithm management.

The *Generic User* can see the information about the algorithms in the system and select the one responsible for forwarding.

The following list of US helps to detail the functionalities presented in Figure 5.1.

¹<https://www.mountaingoatsoftware.com/agile/user-stories>

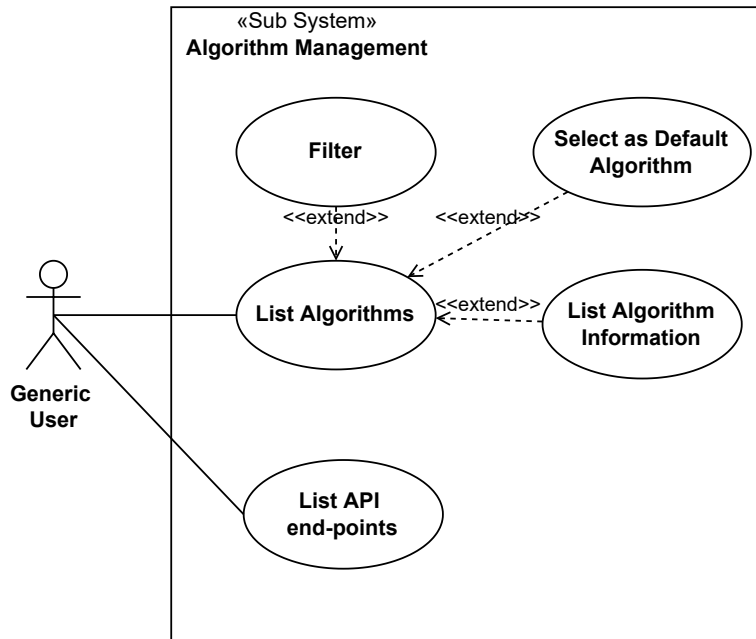


Figure 5.1: Algorithm management diagram

- **US-1: List end-points related with algorithm management**

- **Description:** As a generic user I want to list all the REST end-points that allow me to perform framework algorithm management activities so that I know the end-points available and their structure
- **Acceptance Criteria:** Given that I have access to the framework when I request the unfiltered algorithm management end-point list then the system returns me all the algorithm management end-points.
- End-point information in the list:
 - * **Description**
 - * **REST Verb**
 - * **Uniform Resource Identifier (URI)**
 - * **Query parameters**
 - * **Response samples**
 - * **Response codes**

- **US-2: List Algorithms**

- **Description:** As a generic user I want to list all the algorithms in the system so that I can see the available algorithms.
- **Acceptance Criteria:** Given that I have access to the framework when I request the unfiltered algorithm list then the system returns me all the algorithms.
- Algorithm information in the list:
 - * **Algorithm Identifier**

- * **Name**

- **US-3: Filter Algorithms**

- **Description:** As a generic user I want to apply filters in my algorithm search so that I can find the algorithm(s) I am looking for faster.
- **Acceptance Criteria:** Given that I type the filter values when I submit my request then the system returns the algorithms that match the intersection of the filter values (INNER JOIN).
- Filter by:
 - * **Algorithm Identifier**
 - * **Sub-string of Algorithm Name**

- **US-4: Access Algorithm Information**

- **Description:** As a generic user I want to see all the information associated with a algorithm so that I know its characteristics.
- **Acceptance Criteria:** Given that I have access to the framework when I request all the algorithm data then the system returns all the information.
- Algorithm information:
 - * **Algorithm Identifier**
 - * **Name**
 - * **Description**

- **US-5: Select as Default Algorithm**

- **Description:** As a generic user I want to select an algorithm so that the algorithm performs the forwarding decisions for incoming traffic.
- **Acceptance Criteria:** Given that I have access to the framework and selected an algorithm when I submit the request then the system changes the default forwarding algorithm to the one selected.

Asset Management

Assets represent topology elements, e.g., devices and connections. To manage the network behaviour we need the information of asset and asset statistics, like packet statistics. Thus, this framework subsystem focus on collecting information and providing it to users. Figure 5.2 depicts the flow of the functionality related to asset management.

The *Generic User* can see the information of the assets and their statistics in the system. The framework keeps and history of how this information changed until that point in time.

The following list of US helps to detail the functionalities presented in Figure 5.2.

- **US-6: List end-points related with asset management**

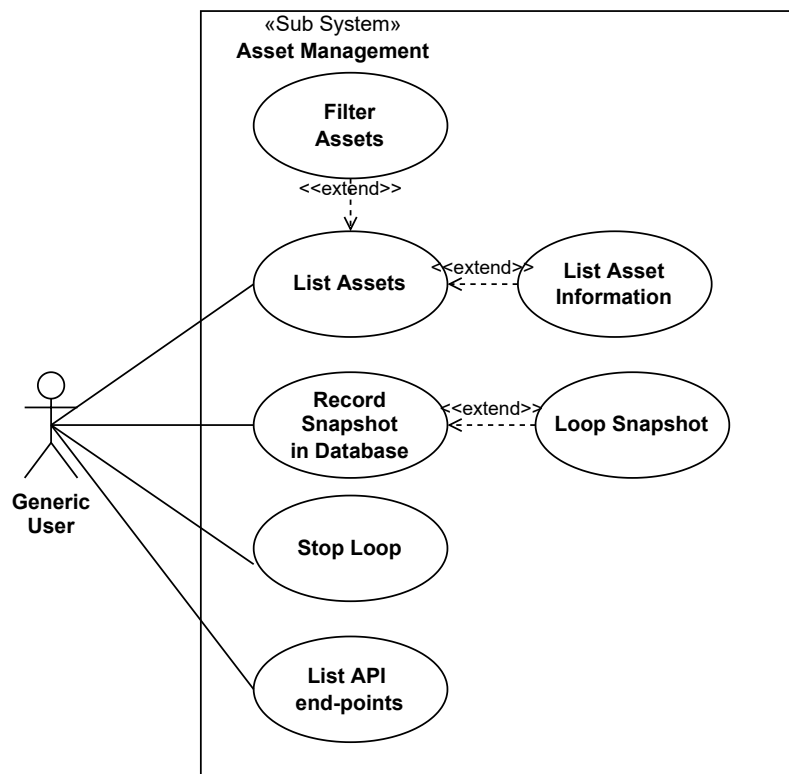


Figure 5.2: Asset management diagram

- **Description:** As a generic user I want to list all the REST end-points that allow me to perform framework asset management activities so that I know the end-points available and their structure.
- **Acceptance Criteria:** Given that I have access to the framework when I request the unfiltered asset management end-point list then the system returns me all the asset management end-point.
- End-point information in the list:
 - * **Description**
 - * **Rest Verb**
 - * **URI**
 - * **Query parameters**
 - * **Response samples**
 - * **Response codes**
- **US-7: List Assets**
 - **Description:** As a generic user I want to list all the assets in the system so that I can see the available assets.
 - **Acceptance Criteria:** Given that I have access to the framework when I request the unfiltered asset list then the system returns me all the assets.

- Assets information in the list:
 - * **Asset Table Identifier**
 - * **Name**
 - * **Asset Type** {*Device, Host, or Link*}
- **US-8: Filter Assets**
 - **Description:** As a generic user **I want** to apply filters in my asset search **so that** I can find the asset(s) I am looking for faster.
 - **Acceptance Criteria:** **Given that** I type the filter values **when** I submit my request **then** the system returns the assets that match the intersection of the filter values (INNER JOIN).
 - Filter by:
 - * **Asset Table Identifier**
 - * Sub-string of **Asset Name**
- **US-9: Access Asset Information**
 - **Description:** As a generic user **I want** to see all the information associated with an asset **so that** I know its characteristics.
 - **Acceptance Criteria:** **Given that** I have access to the framework **when** I request all the asset data **then** the system returns all the information.
 - Asset information:
 - * **Asset Table Identifier**
 - * **Name**
 - * **Asset Type** {*Device, Host, or Link*}
 - * Static Information {identifiers, protocol and drivers versions, interface information, port information}
 - * Statistical Information {flow tables, bytes and packet statistics, loss packets}
- **US-10: Record Snapshot**
 - **Description:** As a generic user **I want** to save the current state of the topology assets in the Database **so that** the information is recorded in permanent storage.
 - **Acceptance Criteria:** **Given that** I have access to the framework and the Database is running **when** I request to save a snapshot of the topology **then** the system places information in the tables of the Database.
 - Tables affected:
 - * Device
 - * Host
 - * Link
 - * Location
 - * Port

- * Port Statistics
- * Flow Rules
- **US-11: Loop Snapshot**
 - **Description:** As a generic user I want save snapshots of the topology periodically so that I can monitor possible information changes in the Database.
 - **Acceptance Criteria:** Given that I have access to the framework and the Database is running when I request to start a looping action and provide the frequency for recording snapshots then the system takes a snapshot of the topology after respecting the frequency indicated.
- **US-12: Stop Loop**
 - **Description:** As a generic user I want stop the looping action so that the system stops recording topology information in the Database.
 - **Acceptance Criteria:** Given that I have access to the framework and the system is taking snapshots periodically when I request to stop the looping action then the system stop taking snapshots.

Configuration Management

There are several configuration parameters that users can adjust to manipulate the way the forwarding algorithms affect the topology flows: energy consumption, service flows, link speeds, device models, and others. Figure 5.3 depicts the flow of the functionality related to configuration management.

The *Generic User* can see the information about the configurations and change its contents. The following list of US helps to detail the functionalities presented in Figure 5.3.

- **US-13: List end-points related with configuration management**
 - **Description:** As a generic user I want to list all the REST end-points that allow me to perform framework configuration management activities so that I know the end-points available and their structure
 - **Acceptance Criteria:** Given that I have access to the framework when I request the unfiltered configuration management end-point list then the system returns me all the configuration management end-point.
 - End-point information in the list:
 - * **Description**
 - * **Rest Verb**
 - * **URI**
 - * **Query parameters**
 - * **Response samples**
 - * **Response codes**

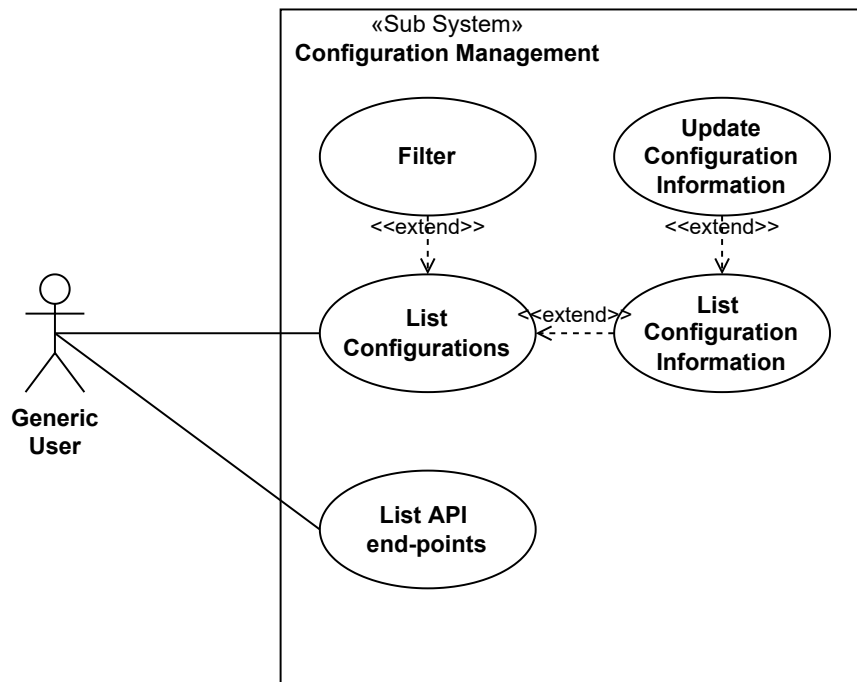


Figure 5.3: Configuration management diagram

- **US-14: List Configurations**

- **Description:** As a generic user I want to list all the configuration parameters in the system so that I can see the available parameters.
- **Acceptance Criteria:** Given that I have access to the framework when I request the unfiltered configuration list then the system returns me all the configuration parameters.
- Configuration information in the list:
 - * Configuration Parameter **Table Identifier**
 - * Configuration Parameter **Name**
 - * Description

- **US-15: Filter Configurations**

- **Description:** As a generic user I want to apply filters in my configuration search so that I can find the configuration parameter(s) I am looking.
- **Acceptance Criteria:** Given that I type the filter values when I submit my request then the system returns the assets that match the intersection of the filter values (INNER JOIN).
- Filter by:
 - * Configuration Parameter **Table Identifier**
 - * Sub-string of Configuration Parameter **Name**

- **US-16: Access Configuration Information**
 - **Description:** As a generic user I want to see all the values associated with a configuration parameter so that I know the values it.
 - **Acceptance Criteria:** Given that I have access to the framework when I request to see the parameter values then the system returns all the information.
 - Information affected:
 - * Parameter Value
- **US-17: Update Configuration Information**
 - **Description:** As a generic user I want to edit configuration parameter values so that the information associated with configuration parameters is up to date.
 - **Acceptance Criteria:** Given that I want to update the information of a parameter when I request to edit it then the system changes the value stored to the value the user requested.
 - Editable Information:
 - * Parameter value
- **US-18: Initialise Configuration Information**
 - **Description:** As a generic user I want to initialise the configuration parameter values so that the information associated with configuration parameters is not null.
 - **Acceptance Criteria:** Given that the tables are empty and I want to initialise a parameter when I request to initialise it then the system changes the stored value to the one that the user requested.
 - Editable Information:
 - * Parameter value

API Utilisation

There are several Representational State Transfer (REST) end-points that allow integration of our framework with external tools. They allow external users to monitor the topology behaviour, but also manipulate it. Figure 5.4 depicts the flow of the functionality related to Application Programming Interface (API) utilisation.

The *Generic User* can access end-points to see or change the topology information. Monitoring activities are provided through GET verbs and modifications are possible using PUT and POST verbs.

The following list of US helps to detail the functionalities presented in Figure 5.4.

- **US-19: List Assets End-Point**

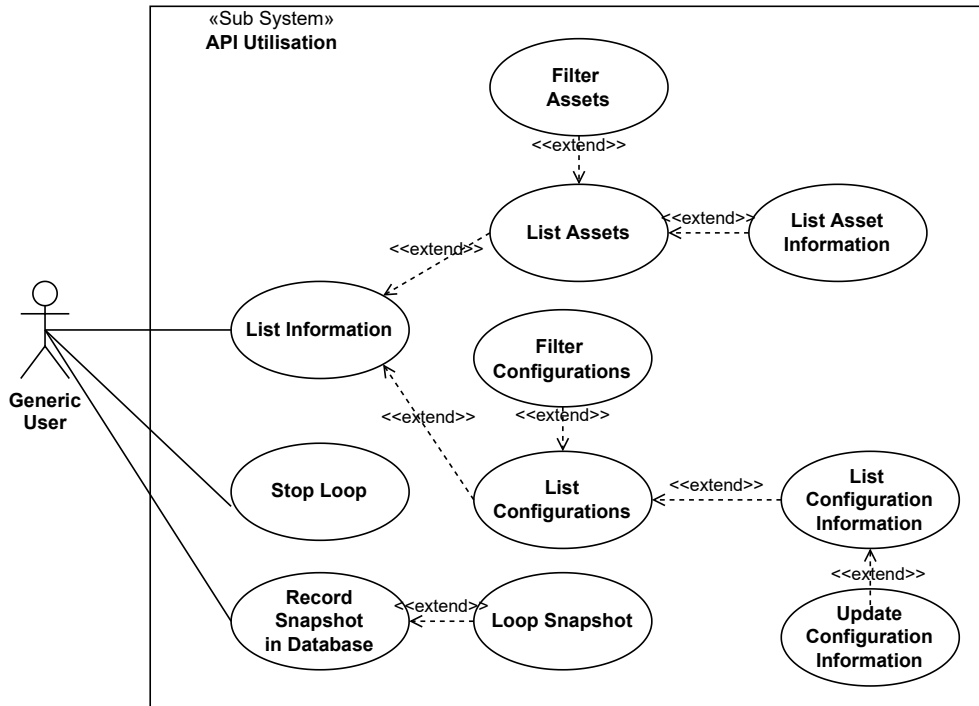


Figure 5.4: API Utilisation diagram

- **Description:** As a generic user I want to list all the assets in the system so that I can see the available assets.
- **Acceptance Criteria:** Given that the web server is running when I make the API call for the unfiltered asset list then the system returns me all the assets (code 200).
- Asset information in the list:
 - * Asset Table Identifier
 - * Name
 - * Asset Type {Device, Host or Link}
- **US-20: Filter Assets End-Point**
 - **Description:** As a generic user I want to apply filters in my asset URI so that I can find the asset(s) I am looking for faster.
 - **Acceptance Criteria:** Given that the web server is running and I type the filter values in the URI when I make the API call then the system returns the assets that match the intersection of the filter values (code 200).
 - Filter by:
 - * Asset Table Identifier
 - * Sub-string of Asset Name
- **US-21: Access Asset Information End-Point**

-
- **Description:** As a generic user I want to see all the information associated with an asset so that I know its characteristics.
 - **Acceptance Criteria:** Given that the web server is running when I make the API call to see all of the asset data then the system returns me all the information (code 200).
 - Asset information:
 - * Asset **Table Identifier**
 - * **Name**
 - * Asset **Type** {*Device, Host* or *Link*}
 - * Static Information {identifiers, protocol and drivers versions, interface information, port information}
 - * Statistical Information {flow tables, bytes and packet statistics, loss packets}
 - **US-22: List Configurations End-Point**
 - **Description:** As a generic user I want to list all the configuration parameters in the system so that I can see the available configuration parameters.
 - **Acceptance Criteria:** Given that the web server is running when I make the API call for the unfiltered configuration parameters list then the system returns me all the configuration parameters (code 200).
 - Configuration information in the list:
 - * Configuration Parameter **Table Identifier**
 - * Configuration Parameter **Name**
 - * Description
 - **US-23: Filter Configurations End-Point**
 - **Description:** As a generic user I want to apply filters in my configuration parameters URI so that I can find the configuration parameter(s) I am looking for faster.
 - **Acceptance Criteria:** Given that the web server is running and I type the filter values correctly in the URI when I make the API call then the system returns the configuration parameters that match the intersection of the filter values (code 200).
 - Filter by:
 - * Configuration Parameter **Table Identifier**
 - **US-24: Access Configuration Information End-Point**
 - **Description:** As a generic user I want to see all the information associated with an configuration parameter so that I know the information that characterises it.
 - **Acceptance Criteria:** Given that the web server is running when I make the API call to see all of the configuration parameters data then the system returns me all the information (code 200).

- Configuration information:
 - * Parameter Value
- **US-25: Update Configuration Information End-Point**
 - **Description:** As a generic user I want to edit configuration parameter values **so that** the information associated with configuration parameters is up to date.
 - **Acceptance Criteria:** Given that the web server is running and I want to update the information of a parameter **when** I make the API call, using PUT, to edit it **then** the system changes the value stored to the value the user requested (code 201).
 - Editable Information:
 - * Parameter value
- **US-26: Initialise Configuration Information End-Point**
 - **Description:** As a generic user I want to initialise the configuration parameter values **so that** the information associated with configuration parameters is not null.
 - **Acceptance Criteria:** Given that the web server is running, the tables are empty and I want to initialise the information of a parameter **when** I make the API call, using POST, to initialise it **then** the system changes the value stored to the value the user requested (code 200).
 - Editable Information:
 - * Parameter value
- **US-27: Record Snapshot End-Point**
 - **Description:** As a generic user I want to save the current state of the topology assets in the Database **so that** the information is recorded in permanent storage.
 - **Acceptance Criteria:** Given that the web server is running and the Database is running **when** I make the API call to save a snapshot of the topology **then** the system inserts information in the Database in the respective tables.
 - Tables affected:
 - * Device
 - * Host
 - * Link
 - * Location
 - * Port
 - * Port Statistics
 - * Flow Rules
- **US-28: Loop Snapshot End-Point**

- **Description:** As a generic user **I want** save snapshots of the topology periodically **so that** I can monitor information changes in the Database.
 - **Acceptance Criteria:** **Given that** the web server is running and the Database is running **when** I make the API call to start a looping action and provide the frequency for recording snapshots in the URI **then** the system takes a snapshot of the topology after respecting the frequency indicated (code 200).
- **US-29: Stop Loop End-Point**
 - **Description:** As a generic user **I want** stop the looping action **so that** the system stops recording topology information in the Database.
 - **Acceptance Criteria:** **Given that** the web server is running and the system is taking snapshots periodically **when** I make the API call to stop the looping action **then** the system stop taking snapshots.

5.3 Non Functional Requirements

This section discloses the scope of the quality attributes of our target framework.

Pure SDN networks There can be networks that contain legacy switches alongside Software Defined Network (SDN) enabled devices, this type of network, which mixes paradigms, makes part of the hybrid paradigm of SDN. These hybrid SDN enabled networks are out of the scope of our framework.

The framework is built to work in pure SDN environments.

Scalability Our framework research and development process is closely integrated with SNOB-5G and MH-SDVANET projects, recall Section 3.1 These projects deal with emerging services in demanding use cases, like 5G drivers, smart-cities, and other more.

The framework performs its normal functions in a smart-city environment, just like the one of Aveiro Tech City Living Lab (ATCLL)² project. ATCLL testbed can be used to validate the framework.

Interoperability Frameworks must be able to **expose functionalities** of their monitoring and management modules with external tools through REST endpoints.

5.4 Design and Technical Restrictions

This section contains restrictions that affected the decision making process, namely the requirements elicitation process and implementation choices.

²<https://www.aveirotechcity.pt/pt/atividades/aveiro-tech-city-living-lab>

SNOB-5G Our framework research and development process is closely integrated with the SNOB-5G project, see Section 3.1.1. SNOB-5G activities use diverse technologies and tools, like the Open Network Operating System (ONOS) controller and the OF protocol. Thus, implementation for the ONOS controller was preferable. In the scope of the SNOB-5G project is also the research of traffic forwarding solutions, with fairness concerns, in multi path wireless environments. Again, this also weighed our decision to provide a fairness forwarding mechanism in the framework.

MH-SDVANET The research and development efforts to reach our target framework are also closely integrated with the MH-SDVANET project, see Section 3.1.2. This project focuses on the management of Vehicular ad hoc networks (VANETs) and uses the ATCLL as its testbed. These smart-city environments contain devices with distinct energy consumptions and throughput, wireless and mesh connections and diverse service flows in the network. Thus, our target framework was developed with the intent of being able to work in these types of scenarios.

5.5 Requirement Listing

This section aims to compile a complete list of requirements, that were extracted from the previous sections, and rank them by priority.

To perform the prioritisation we utilised the **MoSCoW**³ method, where each requirement is given a tag to arrange them hierarchically with regards to the importance to the project success.

- **Must Have** The requirement is non negotiable. It must be present in the final product to not compromise the success of the product.
- **Should Have** The requirement is non vital but dramatically increases the product value.
- **Could Have** Would be “nice-to-have” the requirement but they are not necessary to the core functionalities of the product. Leaving it out only has a small impact on the product’s success.
- **Won’t Have** Indicates requirements that are out of the scope of the project. If included, this type of requirements contributes to the complexity by increasing the project scope.

This ranking step is important because it helps to know the critical aspects of the product and can give some margin for manoeuvre in case of scope management or risk mitigation. To reach each rank, the main framework author and the advisor professor discussed each entry until a consensus was made.

³<https://www.productplan.com/glossary/moscow-prioritization/>

At the end of the project, we revised this section, to document if a specific requirement was completed (Y) or not (N). This is done to help evaluate the success of the project.

Table 5.3 contains all functional requirements related to management activities that were extracted from the documents of Section 5.1.

Table 5.3: Management activities requirements list

ID	Name	Description	MoSCoW Scale	
MF1	Load Balancing	Support load balancing between different links from a source to a destination	Must have	Y
MF2	Broadcast Prevention	Prevent broadcast message propagation in networks, or network segments	Must have	Y
MF3	Support for Traffic Engineering	Allow definition of traffic flow paths to enforce TE in a effort to achieve QoS	Must have	Y
MF4	Congestion control	Manage the traffic load in certain ports by queuing or dropping packets	Must have	Y
MF5	Monitoring traffic flows at data plane	Obtain metrics regarding traffic flows (OF flow table statistics and counters)	Must have	Y
MF6	Monitoring traffic at control plane	Obtain statistics of OF control messages	Must have	Y
MF7	Set monitoring polling interval	Changes the rate at which information of the network is stored	Must have	Y
MF8	Reacts on events	Software Defined Network Controller (SDN-C) detects a ne flow and acts on it	Should have	Y
MF9	Identify network assets	Identify topologies and respective inventory of assets resources	Must have	Y
MF10	Support Traffic Steering	Support policies for traffic steering by detecting the functions in each route point (e.g. Deep Packet Inspection)	Could have	Y

Continuation of Table 5.3				
ID	Name	Description	MoSCoW Scale	
MF11	Service Function Chaining	Framework guarantees that a selected traffic path passes through the necessary service functions	Should have	N
MF12	Support isolation of SDN-C resources	If deployed in an environment with multiple tenants and/or network slices, instead of having an SDN controller per tenant/slice there is one 'shared' but with resource isolation	Could have	N
MF13	Support virtual networks	If deployed in an environment with virtual components the framework must be able to operate them just like it would a physical element	Must have	Y
MF14	Enable support for resource usage fairness	Support Fairness regarding the usage of links or other devices	Must have	Y
MF15	Support diverse NBI protocols	Support REST	Should have	Y
MF16	Support diverse NBI protocols	Support gRPC	Could have	N
MF17	Support JSON data models	Support the exchange of information between framework and external tools through diverse data models (JSON)	Should have	Y
MF18	Support YANG data models	Support the exchange of information between framework and external tools through diverse data models (YANG)	Could have	N
MF19	Support multi-homed connections	Devices can have multiple connections, through different technologies	Should have	Y
MF20	Support IPv6	Interfaces can be single or dual stack	Could have	N

Continuation of Table 5.3				
ID	Name	Description	MoSCoW Scale	
MF21	Support flow identification	Recognises flows of different services	Should have	Y
MF22	Support traffic classification	Allow the injection of custom header fields to tag traffic (e.g., for anomalous traffic)	Could have	N

Table 5.4 contains the requirements list extracted from the algorithm management artefacts of Section 5.2

Table 5.4: Algorithm management requirements list

ID	Name	Description	MoSCoW Scale	
FAR1	List Algorithms	List algorithms in the system	Should have	Y
FAR2	Filter Listed Algorithms	Filter the list of algorithms before presenting it to the user	Could have	N
FAR3	Access Algorithms Information	See all the information stored related with a algorithm	Should have	Y
FAR4	Select as Default	An algorithm can be selected as the default forwarding option as a way to steer network behaviour	Must have	Y
FAR5	List REST end-points	Present all the end-points responsible for algorithm management activities and their syntax	Could have	N

Table 5.5 contains the requirements list extracted from the asset management artefacts of Section 5.2

Table 5.5: Asset management requirements list

ID	Name	Description	MoSCoW Scale	
FER1	List all the Assets	List assets in the system	Should have	Y
FER2	Filter Listed Assets	Filter the list of assets before presenting it to the user	Could have	N

Continuation of Table 5.5				
ID	Name	Description	MoSCoW Scale	
FER3	Get Asset Information	Get a list of all the collected information about an asset. The information collected and stored in each sampling rate is listed	Should have	Y
FER4	Take Snapshot	Save the current topology information in Database	Must have	Y
FER5	Take Snapshot Periodically	Takes a snapshot of the topology with X seconds of interval	Must have	Y
FER6	Stop Periodic Snapshot	Stops the looping action of taking snapshots	Must have	Y
FER7	List REST end-points	Present all the end-points responsible for asset management activities and their syntax	Could have	N

Table 5.6 contains the requirements list extracted from the configuration management artefacts of Section 5.2

Table 5.6: Configuration management requirements list

ID	Name	Description	MoSCoW Scale	
FCR1	List Configurations	List configurations in the system	Should have	Y
FCR2	Filter Listed Configurations	Filter the list of configurations before presenting it to the user	Could have	N
FCR3	Get Configurations Information	Get a list of all the values of that configuration parameter	Should have	Y
FCR4	Update Configuration Information	Change the value(s) of configuration parameter	Must have	Y
FCR5	Add Configuration Information	Add configuration values in the configuration parameters that support multiple entries	Must have	Y
FCR6	List REST end-points	Present all the end-points responsible for configuration management activities and their syntax	Could have	N

Table 5.7 contains the requirements list extracted from the API utilisation artefacts of Section 5.2

Table 5.7: API utilisation requirements list

ID	Name	Description	MoSCoW Scale	
FIR1	Expose End-point	Have end-points that external tools and users can call to perform framework activities, without access to the framework interface	Should have	Y
FIR2	List Configurations End-point	List configurations in the system using an API call (GET verb)	Should have	N
FIR3	Filter Listed Configurations End-point	Filter the list of configurations before presenting it to the user using an API call (GET verb)	Could have	N
FIR4	Get Configurations Info End-point	Get a list of all the values of that configuration parameter using an API call (GET verb)	Should have	Y
FIR5	Update Configuration Info End-point	Change the value(s) of configuration parameters using an API call (PUT verb)	Must have	Y
FIR6	Add Configuration Info End-point	Add configuration values in the configuration parameters that support multiple entries using an API call (POST verb)	Must have	Y
FIR7	List Assets End-point	List assets in the system using an API call (GET verb)	Should have	Y
FIR8	Filter Listed Assets End-point	Filter the list of assets before presenting it to the user using an API call (GET verb)	Could have	N
FIR9	Get Asset Information End-point	Get a list of all the collected information about an asset using an API call (GET verb)	Should have	Y
FIR10	Take Snapshot End-point	Save the current topology information in Database using an API call (GET verb)	Should have	N

Continuation of Table 5.7				
ID	Name	Description	MoSCoW Scale	
FIR11	Take Snapshot Periodically End-point	Takes a snapshot of the topology with X seconds of interval using an API call (GET verb)	Should have	N
FIR12	Stop Periodic Snapshot End-point	Stops the looping action of taking snapshots using an API call (GET verb)	Should have	N

Table 5.8 groups quality attributes of several miscellaneous categories. The requirements were extracted from the documents of Section 5.3.

Table 5.8: Miscellaneous non functional requirements list

ID	Name	Description	MoSCoW Scale	
NF1	User Scalability	The SDN management platform works in scenarios with smart-cities characteristics, like ATCLL, namely number of users	Must have	Y
NF2	Device Scalability	The SDN management platform works in scenarios with smart-cities characteristics, like ATCLL namely number of devices and connections	Must have	Y
NF3	Service Scalability	The SDN management platform works in scenarios with smart-cities characteristics, like ATCLL namely types of services	Must have	Y
NF4	Share Information	The SDN management platform must be able to share the collected information of network behaviour/state with other external tools	Should have	Y
NF5	Share Commands	The SDN management platform must be able to share framework commands with external tools so that they can act on the network.	Should have	Y

Continuation of Table 5.8				
ID	Name	Description	MoSCoW	
NF6	Pure SDN	The platform is only required to work in pure SDN paradigms	Must have	Y
NF7	Original OF	The framework doesn't modify the OF protocol	Must have	Y
NF8	Original ONOS	The framework doesn't modify the ONOS controller internal logic	Must have	Y

In Table 5.9 we present the ranked list of requirements related to our project restrictions extracted from Section 5.4.

Table 5.9: Restrictions and constraints requirements list

ID	Name	Description	MoSCoW Scale	
RC1	SNOB-5G Priority-SDN-C	Development modules that interface with ONOS controller	Must have	Y
RC2	SNOB-5G Priority-Protocol	Development modules that interface with OF protocol	Must have	Y
RC3	SNOB-5G Optional-SDN-C	Development modules that interface with OpenDaylight (ODL) controller	Could have	N
RC4	SNOB-5G Optional-Protocol	Development modules that interface with Programming Protocol-Independent Packet Processors (P4) protocol	Could have	N
RC5	Fairness Algorithm Priority	Develop service fairness management algorithm in the framework	Must have	Y
RC6	MH-SDVANET Priority-Interfaces	The SDN management platform works with devices that have wireless interfaces	Must have	Y
RC7	MH-SDVANET Priority-Mesh	The SDN management platform works in scenarios that have mesh connections	Must have	Y

5.5.1 Requirement Fulfilment Analysis

This section aims to clarify the reasoning behind the ranking attributed to requirements of the previous tables and explain why some of them weren't completed.

In the current version of the framework, there is no user authentication and user management. The Use Cases only identified one actor, *Generic User*, which is authorised to use all the features of the framework and collected data. This means that the *Generic User*, when more advanced data analyses are required, could query the information in the database to retrieve the data formatted as intended. This was reflected in the previous tables, and section because requirements related to retrieve filtered information, or features to enumerate commands or tables had lower priority.

Additionally, due to the issues and delays documented in Section 4.2.3, there wasn't enough time budget to complete these requirements.

On the other hand, there are some requirements that were also marked as not present in the framework but are only incomplete. The requirements FIR2 FIR3 and FIR8 are related to filtering using the end-points URI. Using the conventional URI syntax we were able to filter, e.g. the byte and packet statistics of a given port's device.

The requirements that need more attention are FIR10-FIR12. They are related to the functionality of starting and stopping snapshots loops using end-points. They were ranked with high priority since they represent key features of the framework and are the only way for tools that access the framework using the end-points to record more information into the database.

Because the web server only reads and writes to the database, for an end-point request to trigger a command directly in the application that runs on the ONOS controller further research of a feasible way to achieve this behaviour is necessary. We speculated possible solutions, e.g. using Remote Procedure Call (RPC), but the actual implementation would require time investment to research, develop and test a proper solution.

Since these features aren't available to external tools, the non functional requirement of "interoperability" is not fully met. The framework is complete for Command-Line interface (CLI) users and provides a range of activities using the end-points, but full interoperability isn't present in the current version of the framework.

Chapter 6

Project Architecture

In this Chapter, we present the schema for the framework architecture. We explain the decisions taken, describe the architectural model utilised, the C4 model, and document the artefacts constructed.

These diagrams helped us gain a perception of how the framework could function and aid us in identifying possible missing components or interfaces that we hadn't previously considered. These steps also made us identify available technologies that could fit our needs and consequently choose the more adequately to use in each activity.

Before the documentation of the architectural artefacts, a preliminary draft of the layout of the framework is available, in Figure 6.1, as a way to introduce readers to the interactions and functions that need to be achieved.

The framework is deployed on top of the Open Network Operating System (ONOS) controller and maintains a communication interface to monitor and manage the topology behaviour. A collection of forwarding algorithms is kept in the framework so that when a service flow is detected a path is calculated and enforced in the topology. To access framework functionalities users can utilise a Command-Line interface (CLI) or opt for the option of calling Representational State Transfer (REST) end-points, which are also available to provide an interface to external entries that want to use the framework.

6.1 C4 Model

To build architectural diagrams we opted to use the C4 Model¹. This model helps to communicate the framework architecture at different detail levels. Software development teams, when communicating with stakeholders with different technological know-how levels, can utilise the more adequate model for each situation. These different levels of detail also help to progressively document the architecture of a product, instead of trying to do it all at once.

¹<https://c4model.com/>

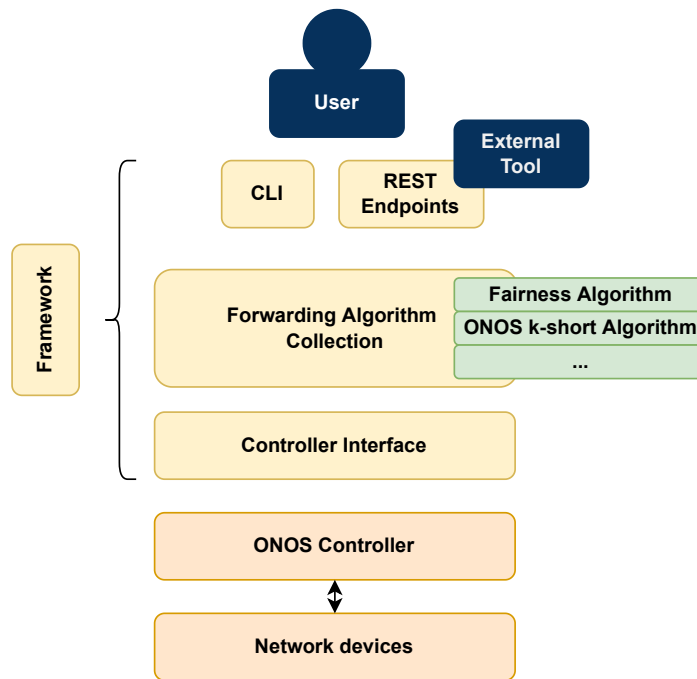


Figure 6.1: Architectural diagram - Preliminary draft

The different levels of detail that the model offers and the previous experience with it, that the authors had, made us choose this model.

There are four diagrams, C1-C4 that increase the detail while zooming in on software components:

- **C1-System Context:** This diagram provides a birds-eye-view of the components. It shows the human and software elements that interact with our product.
- **C2-Container** The diagram zooms into the software system to show the high-level technical building blocks that compose it. It provides also insights into the technologies utilised to interface with different elements of the diagram.
- **C3-Component** This diagram zooms into an individual container that was presented in the C2 view and shows the components inside it.
- **C4-Code** In this level of detail we find a possible implementation for the components that were presented in the C3 view. Normally a UML class is used to achieve this.

There are several elements that make up each diagram, as summarised in Figure 6.2.

Person represents human users that interact with the software system (e.g., actors, roles, personas, etc).

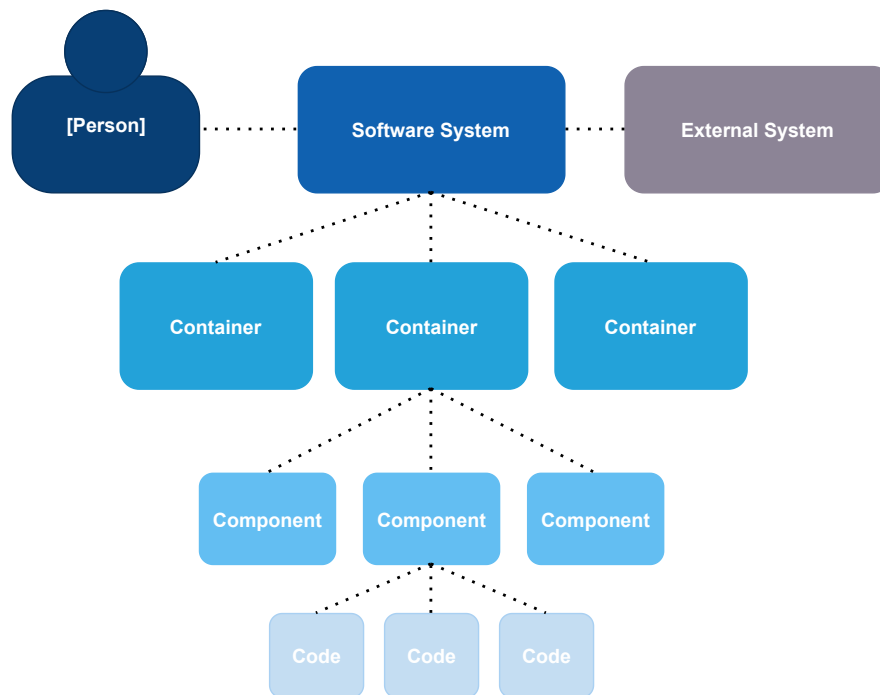


Figure 6.2: C4 architectural model label

Software System is the highest level of abstraction in the model and represents our product/service that is to be implemented.

External System is in the same detail level of the Software System element and represents external tools that our software system depends on or vice versa (e.g., mailing service, banking service).

Container represents an application or a data store. It is something that needs to be operational in order for the rest of the software system to work.

Component are the functionalities that make up the Containers. They are a group of related functions encapsulated behind a well-defined interface.

Code is the pseudo-code that runs to archive Component functionality.

6.2 Architectural Artefacts

This section contains the C4 diagrams developed to detail our proposed framework architecture.

Figure 6.3 displays the diagram with the lowest level of detail. We see that our software system runs on top of the Software Defined Network Controller (SDN-C) and the underlying topology. Users and external tools can interact with our

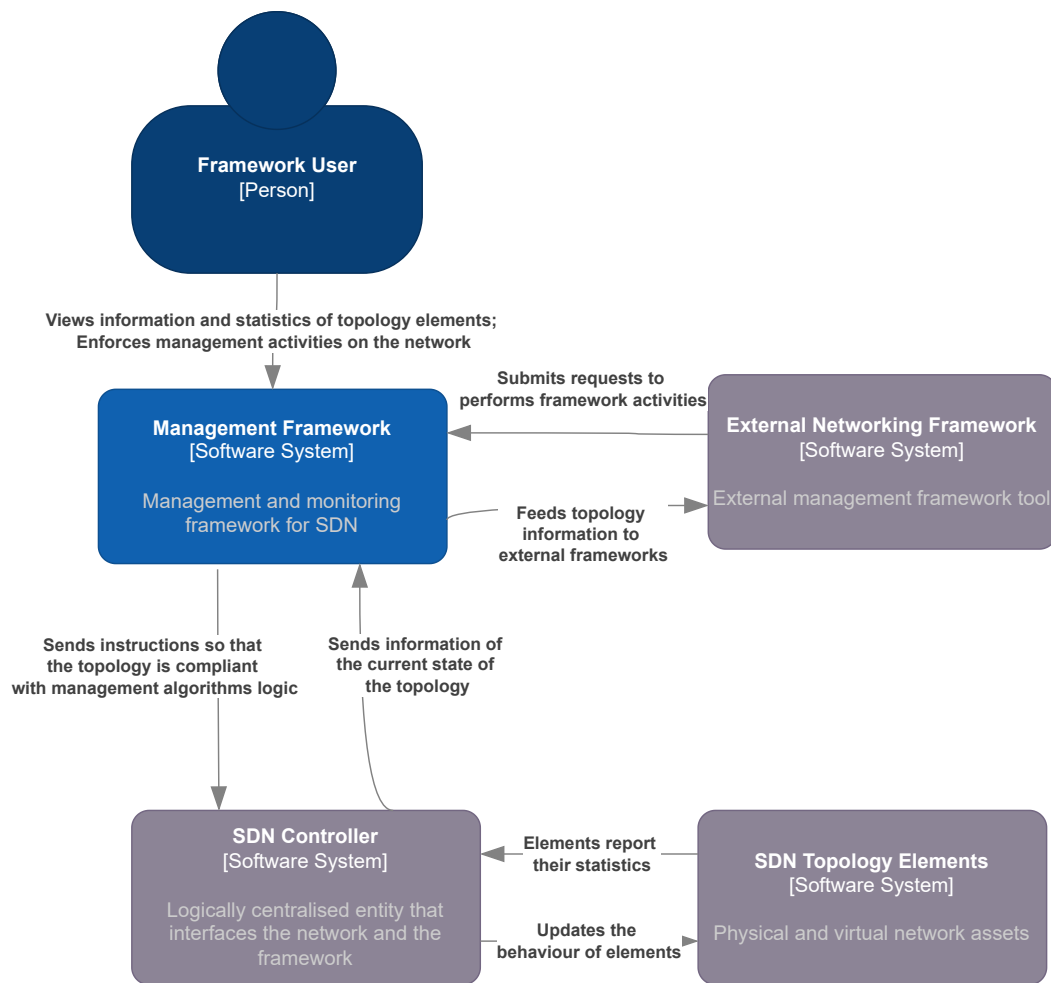


Figure 6.3: C1 architectural diagram - Context

framework to perform the available management and monitor actions.

Figure 6.4 features the container level diagram. We see that the framework communicates with the controller using Northbound SDN-C interfaces and that the business logic is programmed as an ONOS application.

We also have a database container that stores all the framework data, e.g. algorithms, assets, configurations, and a web server container that exposes REST end-points to perform management tasks. The database uses PostgreSQL. We considered also MariaDB but the prior experience of the framework authors and the open-source factor made us choose PostgreSQL. Communications with the database rely on Java Database Connectivity (JDBC), since the controller language is Java.

The Web Server uses Glassfish to run and the Jersey framework to create and maintain the REST end-points. We considered also other popular choices, like Tomcat and Jetty, but steered towards Glassfish since it is open-source. Further-

more, we had experience and were comfortable using Glassfish and the remaining integration of the Jersey framework would be effortless since we found so many helpful materials online.

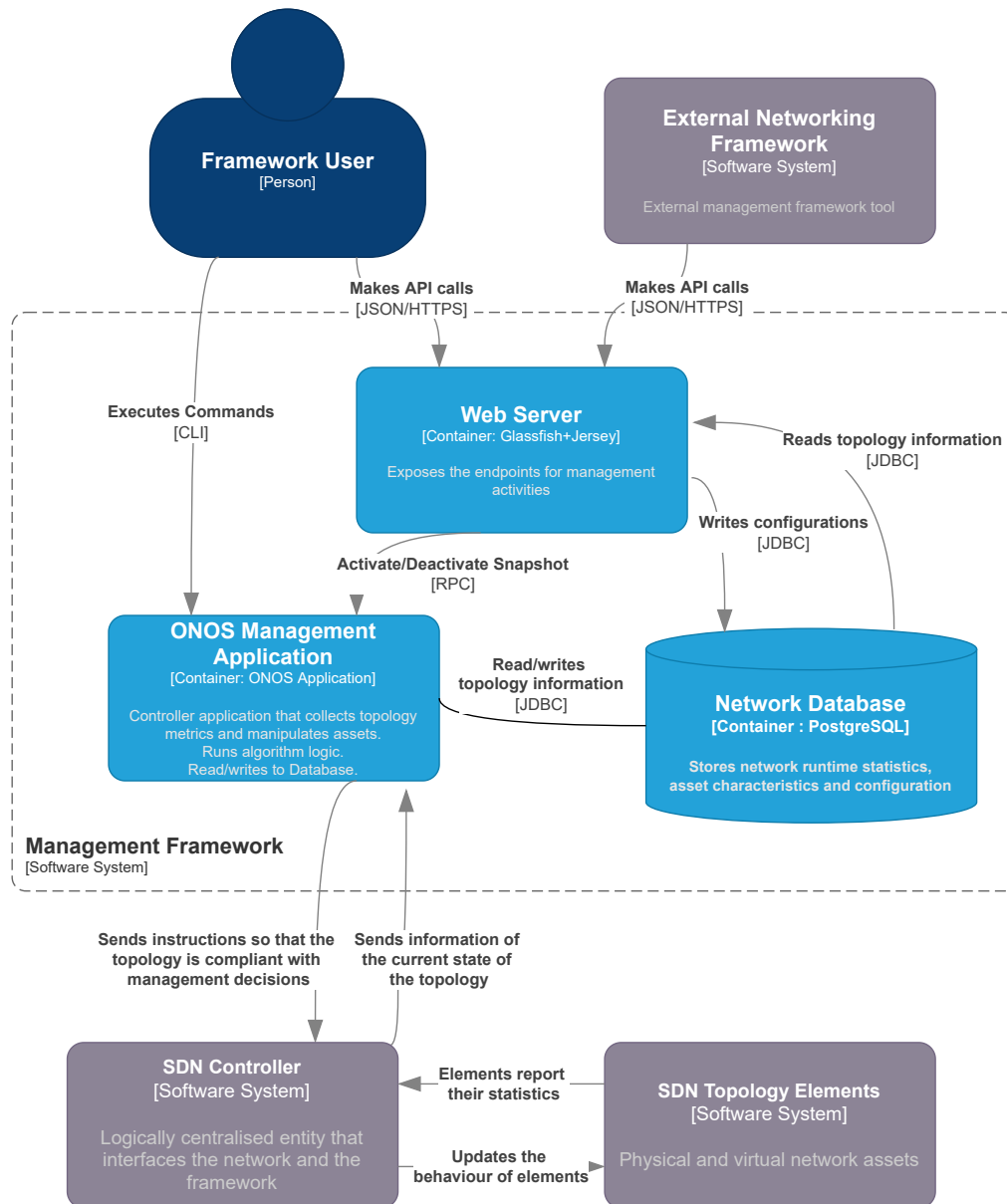


Figure 6.4: C2 architectural diagram - Container

The remaining element of the framework is the ONOS application that contains the business logic of the forwarding elements and the monitoring routines. To execute the framework functionality, users can run CLI commands or make REST Application Programming Interface (API) requests.

The only activities that require direct communication between the web server and

the controller application are the activation or deactivation of the routines that store the topology information in the database. This is due to the synchronous characteristics of this activity. This could be achieved using Remote Procedure Call (RPC), but still needs further research to confirm this hypothesis.

Figure 6.5 contains the inner elements that make up the ONOS Management Application. This container is composed of 4 components. One that handles the commands requested by users, one for metric collection from the framework and storage in a database, a third for processing incoming packets and the final one to run the forwarding algorithms and enforce the changes in the topology.

When a packet reaches the controller, the “Packet Processor” handles it. If the packet is from a registered service it requests the “Algorithm Logic” component to find a suitable path and apply the necessary Openflow (OF) flow rules in the devices. Using the CLI, the framework offers the opportunity to change the configurations of the framework, e.g. change the default forwarding algorithm, register valid service flows, and change the energy consumption of devices.

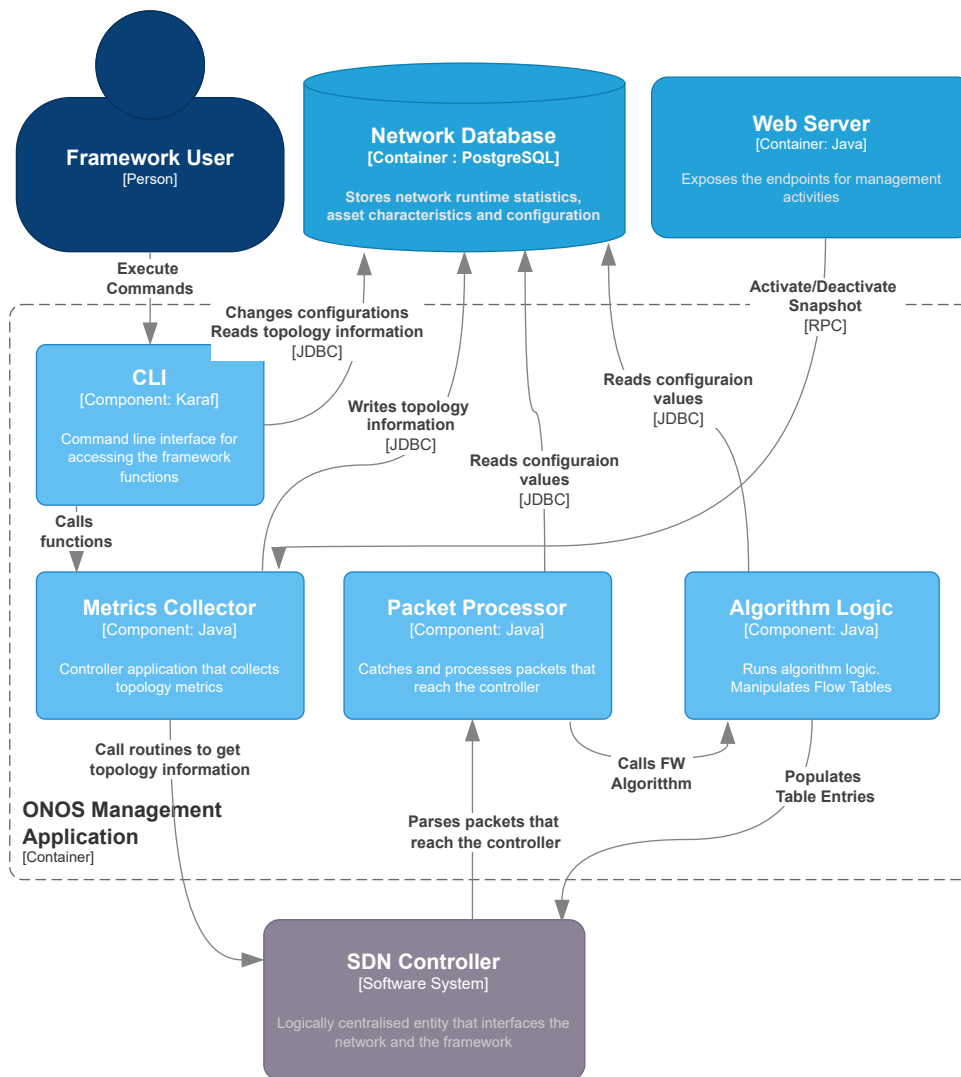


Figure 6.5: C3 architectural diagram - Component: ONOS management application

Chapter 7

Framework Development

This Chapter’s purpose is to document the development of the framework functionalities. We present the implementation of the diverse framework components documented in the previous chapters and explain how to use these services.

7.1 Permanent Storage

In the following subsections, we present how we collect network information, the reasoning behind the schema built to store data, and how users can access these storage features.

7.1.1 Database Schema

The framework provides permanent storage using a PostgreSQL database that runs inside a docker container. The database conceptual schema can be consulted in Figure 7.1. It was designed to be able to store the history of the network information in a way that allows users to keep track of the evolution of the network, previously mentioned as “Snapshots”. Furthermore, the schema also provides storage for configuration parameters that influence the behaviour of the forwarding algorithms, e.g. preferred forwarding solution, default timeout and priority of flow rules.

Network Information

When the framework takes a snapshot of the topology the controller is queried and replies with information related to network assets so that it can be stored.

The *device* table has information that characterises each switch detected by the controller. The entries of the Openflow (OF) flow tables are stored in the *flowrule* table. It contains the packet matching criteria, the treatment for matching packets and the statistics of each entry. The information of device interfaces can be consulted in the *port* table. Each port has statistics associated with it, which can be consulted using table *byte_statistics* and *packet_statistics*. Link assets information

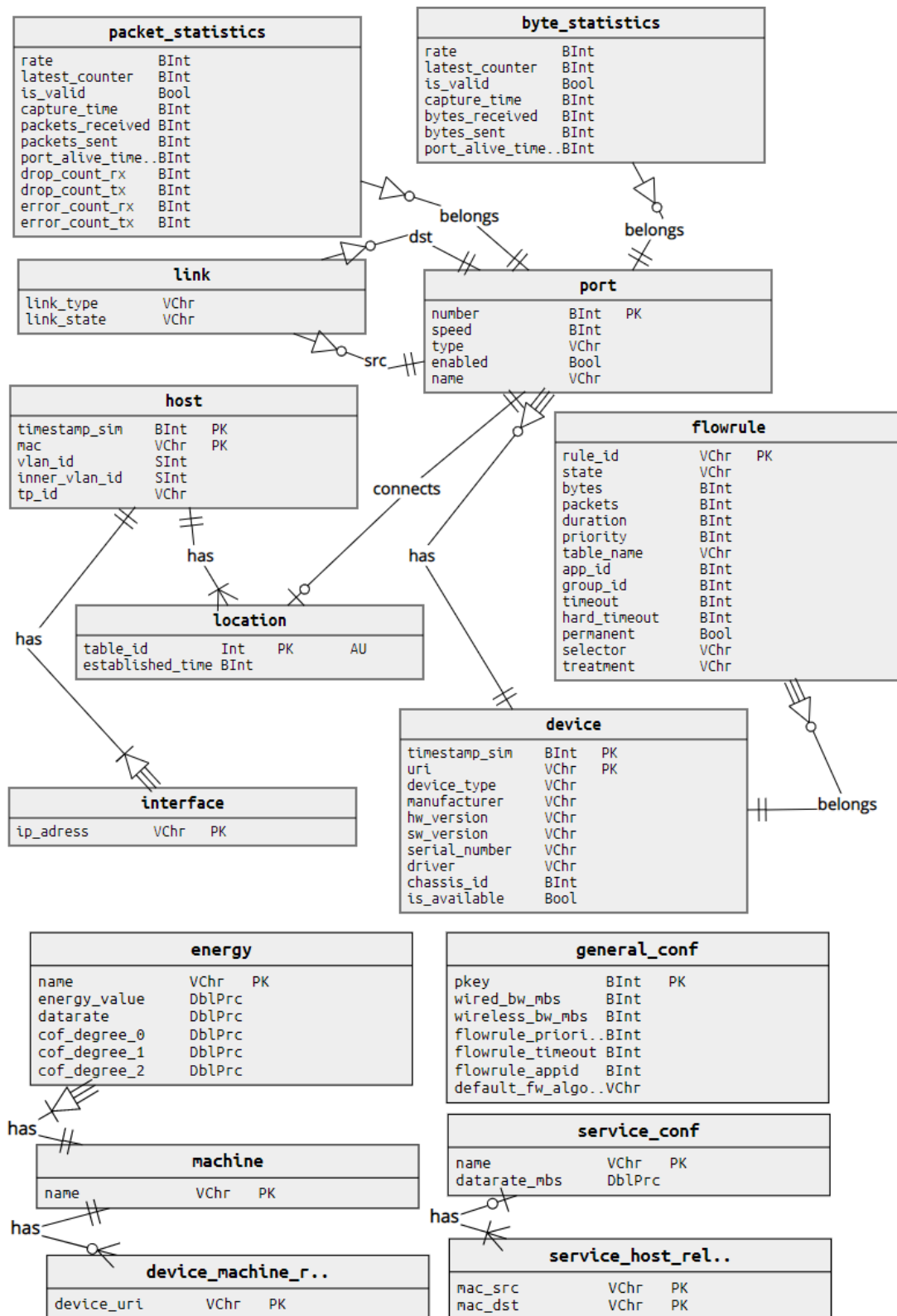


Figure 7.1: Database conceptual diagram

is stored in the *link* table. Note that each link is associated with a source and a destination port. The *host* table contains the information that characterises end machines - nodes participating in a network topology. Table *interface* has the IP address of each machine. In table *location*, we store the historical connections of

each host in the topology (store the timestamp when a host was connected with a certain device's port).

To keep logged the evolution of the network topology and behaviour, the data of each host and device is stored with the time of the collection. This allows us to know if there were elements introduced/removed from the network and also enables us to analyse the statistics collected at run-time.

Configurations

In the schema, the configuration tables are independent of the network information tables, despite referencing some of their identifiers, e.g. `device_uri`. This is to allow the configuration of the framework even before the network is running, e.g. users can account for services that might show up in the future, or devices that are installed after the initial start of the topology.

The *general_conf* table stores configurations of general use: the default forwarding algorithm to use, and default properties of flow entries (e.g. priority timeout). Keep in mind that this table is constrained to have only one entry at a time (this is intended behaviour). Also, the wired and wireless bandwidth fields were necessary to work around the limitations of OF: this protocol is not ready to recognise wireless connections and so it was reporting incorrect port speeds to the controller. The *service_conf* and *service_host_relation* tables allow users to register service flows that are allowed to be forward by the framework, recall requirements MF8 and MF21 from Section 5.5. A service can be registered by defining a name and bandwidth necessary to reserve alongside the source and destination hosts of each flow. The traffic of unregistered services will simply be dropped by the controller when packets of their flow reach the SDN-C.

Finally, the framework allows for the distinction of network switch machines. This feature allows for users to define the energy consumption of different devices on the topology. The *energy* table is the one that provided these consumption values by taking advantage of the feature of Generated Columns of PostgreSQL 12¹. The table was created in a way that by updating the value of the *datarate* in a table entry, the value of the *energy_value* parameter is updated using the polynomial formula:

$$(conf_deegree_2 * datarate^2) + (conf_deegree_1 * datarate) + conf_deegree_0$$

where `conf_deegree_2`, `conf_deegree_1` and `conf_deegree_0` are the polynomial coefficients and `datarate` is the `datarate` of the service flow. This allows users to build flexible energy consumption models.

7.1.2 Metrics collector

Another element that is part of the framework is the Open Network Operating System (ONOS) application responsible for the collection of data. Using the

¹<https://pgdash.io/blog/postgres-12-generated-columns.html>

ONOS Java API² the application communicates with the controller to request information of topology devices. This application contains the logic that allows users to periodically take snapshots of the network state. Furthermore, the application also defines Command-Line interface (CLI) commands so that users can manipulate the database schema and access its contents:

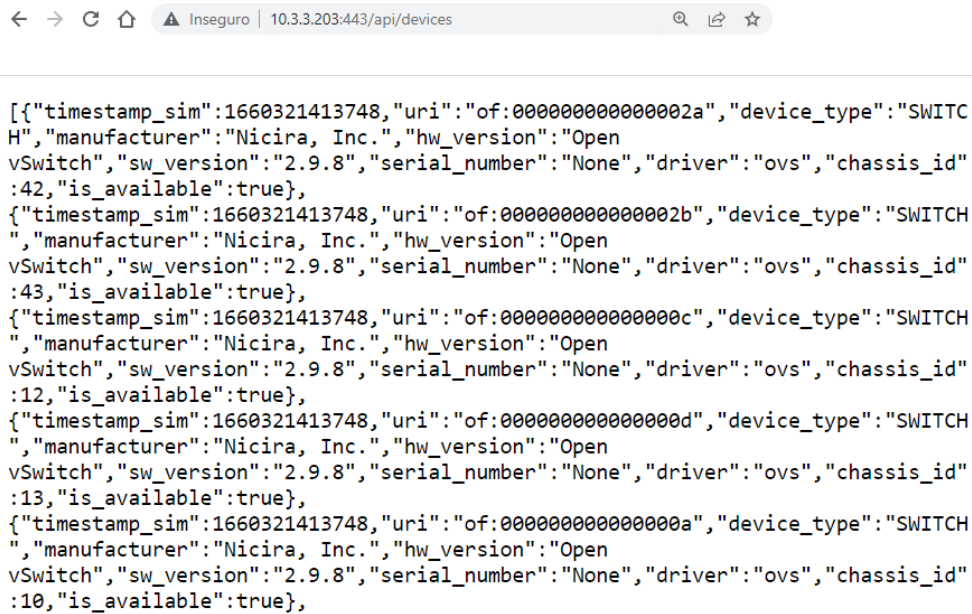
- `$onos database -read X` : Queries the database and prints the information of table X;
- `$onos database -insert X [arg1, arg2, ..]`: Queries the database and inserts in table X a new entry with the information of the list;
- `$onos database -update X [arg1, arg2, ..]`: Queries the database and updates in table X a the entry with the information of the list;
- `$onos database -reset`: Deletes all tables and then creates them afterwards;
- `$onos database -init`: Initialises the configuration tables;
- `$onos database -create`: Run the script to creates all the database tables;
- `$onos database -loop X`: Takes a snapshot of the topology each X seconds. The terminal blocks but the user can close the terminal and the looping action continues;
- `$onos database -stoploop`: Stops any looping actions of taking topology snapshots that might be running;
- `$onos database -snap`: Takes a snapshot of the topology;
- `$onos database -pullDB`: Pulls the current values from the configuration tables in the database to the framework;

7.2 Web Server

The framework contains another docker container where the web server runs. From a clean Ubuntu image, we installed the open-source Glassfish web server and the open-source Jersey RESTful Web Services framework. We also used the jOOQ Object-Relational Mapping (ORM) to help advance the implementation task by automatically generating classes to interface with database tables. The Application Programming Interface (API) follows Representational State Transfer (REST) conventions, like idempotent verbs (excluding POST), Uniform Resource Identifier (URI) naming conventions and reply codes.

There is an end-point for each database table: `IPAddress/api/{tableName}`. With all of them, we can use **GET** to retrieve the information stored and with the tables of configurations, e.g `general_conf`, `service_conf`, `energy`, users can also execute

²<https://api.onosproject.org/2.7.0/apidocs/>



```

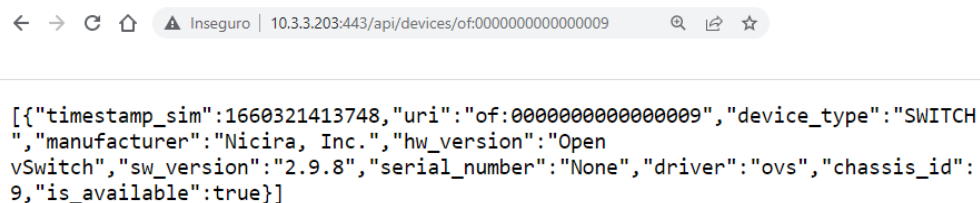
[{"timestamp_sim":1660321413748,"uri":"of:000000000000002a","device_type":"SWITCH",
"manufacturer":"Nicira, Inc.,"hw_version":"Open vSwitch","sw_version":"2.9.8",
"serial_number":"None","driver":"ovs","chassis_id":42,"is_available":true},
{"timestamp_sim":1660321413748,"uri":"of:000000000000002b","device_type":"SWITCH",
"manufacturer":"Nicira, Inc.,"hw_version":"Open vSwitch","sw_version":"2.9.8",
"serial_number":"None","driver":"ovs","chassis_id":43,"is_available":true},
{"timestamp_sim":1660321413748,"uri":"of:000000000000000c","device_type":"SWITCH",
"manufacturer":"Nicira, Inc.,"hw_version":"Open vSwitch","sw_version":"2.9.8",
"serial_number":"None","driver":"ovs","chassis_id":12,"is_available":true},
{"timestamp_sim":1660321413748,"uri":"of:000000000000000d","device_type":"SWITCH",
"manufacturer":"Nicira, Inc.,"hw_version":"Open vSwitch","sw_version":"2.9.8",
"serial_number":"None","driver":"ovs","chassis_id":13,"is_available":true},
{"timestamp_sim":1660321413748,"uri":"of:000000000000000a","device_type":"SWITCH",
"manufacturer":"Nicira, Inc.,"hw_version":"Open vSwitch","sw_version":"2.9.8",
"serial_number":"None","driver":"ovs","chassis_id":10,"is_available":true},

```

Figure 7.2: GET reply example

POST and **PUT** requests, to add configuration entries or update them, respectively. The GET actions always return information in JSON format and the PUT and POST actions always need a JSON in the request body.

Besides being able to retrieve all entries of a table with a GET request, see Figure 7.2, it is also possible to retrieve specific entries of a table, see Figure 7.3.



```

[{"timestamp_sim":1660321413748,"uri":"of:0000000000000009",
"device_type":"SWITCH",
"manufacturer":"Nicira, Inc.,"hw_version":"Open vSwitch",
"sw_version":"2.9.8","serial_number":"None","driver":"ovs",
"chassis_id":9,"is_available":true}]

```

Figure 7.3: GET reply example with filtering

The following list contains end-points exposed by the framework that allow users to filter for specific resources, recall Section 5.5.1:

List of end-points useful for filtering

```

api/devices/{uri}
api/devices/{uri}/ports
api/devices/{uri}/ports/{port}
api/devices/{uri}/ports/{port}/bytestatistics
api/devices/{uri}/ports/{port}/packetstatistics

```

where the {uri} is the device identifier, e.g. 'of:0000000000000001', and the {port} is the interface identifier, e.g '2'.

7.3 Forwarding Mechanisms

This section explains how the controller recognises a packet flow, finds a path to the target destination and enforces the path in the topology. The logic of these features is also programmed as an ONOS application. All the devices are configured to handle incoming IPv4 packets by redirecting them to the ONOS application. If these are valid packets, the application install rules with higher priority in the devices to accomplish the desired traffic forwarding.

Packet Processor

The framework implements a custom packet processor that parses through incoming packets³. From the packets, the application extracts the source and destination MAC host addresses and then queries the database *service_host_relation* table to figure out if this traffic flow corresponds to any registered service. If the packets don't match the information of any registered service, the controller will drop them.

Flow Rules

When a packet is identified as being from a registered service, the default forwarding algorithm is called and returns a path to reach the target host destination. From the output of the forwarding algorithm, we parse the solution path and install flow rules on the involved devices, to enforce the determined path.

Table 7.1 illustrates the possible rules of a flow table in a device after this process. Future packets of the service that reach the switch are forwarded out to the correct port because they match the rule with the highest priority (biggest number). Packets of other services are still redirected to the controller.

Table 7.1: Device example after forwarding decisions

State	Priority	Timeout	Table	Selector	Treatment
ADDED	20	20 sec	0	IN_PORT:2, ETH_DST:MAC2, ETH_SRC:MAC1, ETH_TYPE:ipv4	OUTPUT:3
ADDED	15	-	0	ETH_TYPE:ipv4	OUTPUT: CONTROLLER

Through the CLI users have access to forwarding related commands to manipulate the algorithm used:

- `$onos algo -getAlgos`: Returns and prints the options for default forwarding algorithms;
- `$onos algo -currentAlgo`: Returns and prints the current default forwarding algorithm;

³<https://api.onosproject.org/2.7.0/apidocs/org/onosproject/net/packet/PacketProcessor.html>

7.3.1 Custom Fairness algorithm

Fairness mechanisms are of immense importance in network management: they help maximise the utilisation of the available topology resources, can help allocate resources across different services or users and can be considered as per-flow fairness, per-link fairness, per node fairness and system-wide fairness [Shi et al., 2014]. It is necessary to go beyond the typical equal distribution of resources [Ghaleb et al., 2021] since when translated to a real scenario it performs poorly.

Our approach sees service fairness as a method to ensure that the resource are distributed throughout the network according to a set of metrics. Three objectives are considered: maximisation of reliability (through packet loss), minimisation of delay, and minimisation of energy consumption. This problem is modelled as a tri-objective multi-commodity problem, where each service corresponds to a commodity and each link has a capacity (bandwidth) constraint. Since the formulation was created by a research colleague and isn't present in the framework we won't go into more detail.

From this formulation was derived a heuristic that works as a single-path min-cost-flow algorithm that is available in our framework.

The formulation gives the optimal path for a service to take, but since finding a solution can take more than 30 minutes, it is not useful to use in a live setting. On the other hand, the heuristic gives a close to optimal path in near to real-time.

By comparing the solution paths of both approaches we can know-how close to optimal the heuristic is performing.

This is an innovative approach since it is more common for works in the literature to transform multiple objectives into a single version with weights (instead of min-max and normalisation) and to not compare the results with the optimal provided by the formulation [Godinho et al., 2022].

To help accelerate the implementation of the heuristic, we resorted to the method *CapacityScalingMinimumCostFlow()* from the Java library JgraphT. To use this method, we had to pass in some input parameters: a graph object of the network (with nodes, links and edge costs), a supply/demand function (source node is the supply, and the target node is the demand), a maximum capacity function (total available bandwidth of link) and a minimum capacity function (0 bandwidth).

Keep in mind that some adjustments were performed in the logic of the supply/demand function and a maximum capacity function to make the JgraphT method behave as we intended, see Section 9.4.

After these alterations:

1. **Minimum capacity function** always returns 0 capacity;
2. **Maximum capacity function** returns 1 capacity if the link has enough bandwidth to transport the service, and 0 capacity if it hasn't;

3. **Supply/demand function** returns 1 if the node is a source of the service that is being forwarded, -1 if the node is a target of the service that is being forwarded, and 0 if it is a different node;
4. **Graph** represents the current topology state with nodes, node connections and links weights that represent the cost of utilising the link;

The objective of the JgraphT method is to find a path that minimises the sum cost of the link weights while being constrained by bandwidth.

The logic of the cost function that sets the link's weight is represented in Algorithm 1.

Algorithm 1 Cost Function

Require: set L of links from topology, set S of registered services, service flow to transport CS

Ensure: Sets the weights of all the links in L

```

1: minimum Loss  $minL$ , Delay  $minD$ , Energy  $minE$ ;
2: maximum Loss  $maxL$ , Delay  $maxD$ , Energy  $maxE$ ;
3:  $CS$  Loss  $xL$ , Delay  $xD$ , Energy  $xE$ ;
4:  $r = \text{datarate}(CS)$ ;
5:
6: for  $l = 1, \dots, L$  do
7:    $src = \text{source}(l)$ ,  $dst = \text{destination}(l)$ 
8:
9:    $minL = 0.0$ ;
10:   $maxL = \text{size}(S) * \text{lossProb}(l)$ ;
11:   $xL = \text{lossProb}(l)$ ;
12:
13:   $minD = 0.0$ ;
14:   $maxD = 0.0$ ;
15:  for  $s = 1, \dots, S$  do
16:     $maxD += \text{datarate}(s) / \text{availableBW}(l)$ ;
17:     $xD = r / \text{availableBW}(l)$ ;
18:
19:     $minE = \text{idleIntfEnergy}(src) + \text{idleIntfEnergy}(dst)$ ;
20:     $maxE = minE$ ;
21:    for  $s = 1, \dots, S$  do
22:       $maxE += \text{UpEnergy}(src, \text{datarate}(s)) + \text{DownEnergy}(dst, \text{datarate}(s)) +$ 
         $\text{cpuLoad}(src) + \text{cpuLoad}(dst)$ ;
23:     $xE = minE + \text{UpEnergy}(src, r) + \text{DownEnergy}(dst, r) + \text{cpuLoad}(src) +$ 
         $\text{cpuLoad}(dst)$ ;
24:
25:  Loss Weight  $rW$ , Delay Weight  $dW$ , Energy Weight  $eW$ ;
26:   $rW = (xL - minL) / (maxL - minL)$ ;
27:   $dW = (xD - minD) / (maxD - minD)$ ;
28:   $eW = (xE - minE) / (maxE - minE)$ ;
29:   $\text{setCost}(\max(rW, dW, eW))$ 

```

Depending on what service needs to be forwarded, CS , the cost function sets a different weight for the link. This is because different services require different amounts of bandwidth to be forwarded (line 4).

For each link, the function calculates the normalised objective value for loss probability, delay and energy consumption, and then sets the link weight as the maximum of these three amounts: the weight is the worst objective amount. To normalise the values we resort to the equations of lines 25-28.

For the loss probability values: the theoretical minimum loss probability for any link is 0% (no loss); The upper value of the loss in a link l is achieved when all the services use the same link l ; the loss probability of service CS being transported in link l is given by the current observable loss probability of link l , see line 11.

Initially, the observable loss probability of a link was planned to be calculated as the difference between the total number of packets that the source interface sends and the total number of packets that reach the destination interface. However, we discovered that the emulator that we used in our experimental scenario, Mininet, simulates traffic losses in the nodes and not in the links: giving an example of the initially expected behaviour, device A needs to send X packets to destination B and only $X - Y$ packets reach the destination, where Y the the number of packets lost; the actual behaviour is that device A sends $X-Y$ packets and $X-Y$ packets reach B.

This means that we needed to change how we calculated the observable loss. At present, the loss probability is given by the mean of the source device losing a packet in any of its interfaces and the destination device losing a packet in any of its interfaces:

$$Sw_Src_Loss = (Sw_Src_sumReceived - Sw_Src_sumSent) / Sw_Src_sumReceived)$$

$$Sw_Dst_Loss = (Sw_Dst_sumReceived - Sw_Dst_sumSent) / Sw_Dst_sumReceived)$$

$$LossProbability = (Sw_Src_Loss + Sw_Dst_Loss) / 2 * 100$$

For the delay probability values: the theoretical minimum delay for any link is 0 seconds (no delay); the upper value of delay is given by the sum of the time that takes link l to transport data from all the services; the delay of transporting the current service in link l is given by the time that takes link l to transport service CS (division of the service datarate, r , by the available bandwidth of link l), see line 17 of Algorithm 1.

For the energy consumption values: the minimum energy consumption is given by the sum of the idle power values of maintaining the source and destination interfaces; the upper value of energy consumption is given by the addition of the minimum energy value with the sum of the energy of the source device sending traffic of all services, the energy of the destination device receiving traffic of all services, and the energy of the source and destination CPUs processing the data of all services, see lines 20-22; the energy consumption of service CS being transported in link l is given by the sum of the minimum energy value, the energy

of the source device sending traffic of *CS*, the energy of the destination device receiving traffic of *CS*, and the energy of the source and destination CPUs processing the data, see line 23.

7.3.2 K-shortest ONOS algorithm

ONOS provides off-the-shelf mechanisms for forwarding. Namely, the *PathService* class provides methods to find the K-shortest paths with regards to hop count or location distance using the *EdgeWeigher* interface class. Furthermore, developers can build custom *EdgeWeigher* interfaces to manipulate the weights of links utilised in the K-shortest method.

The framework is equipped with a custom K-shortest path algorithm that considers energy cost for the link weight. This calculation is the sum of the energy of the source device sending traffic (uplink), the destination device receiving traffic (downlink), idle power of keeping source and destination interfaces and CPU utilisation of source and destination machines:

$$\text{Weight} = \text{src, up} + \text{dst, down} + \text{src, idle} + \text{dst, idle} + \text{src, cpu} + \text{dst, cpu}$$

The algorithm will return one of the possible paths that minimise the energy consumption of devices. Using this metric instead of hop count is more appropriate since in a real scenario there are few to no reasons to calculate the cost using hop count. Opting to consider link distance, service delay, link congestion or even energy cost is more realistic.

7.4 Potentially Interesting Features

This section serves the purpose of documenting interesting features that could be added to a future version of the framework.

- The framework already saves the evolution of network asset statistics in the database. Users can see how information changes during the topology lifetime. One interesting feature could be to also track the changes in the configuration parameters;
- The current version of the forwarding algorithms in the framework don't support multiple links between the same node pair properly. In the next version this could be fixed to allow the algorithms to work with even more interesting scenarios;
- To identify service flows the framework uses the source and target nodes of the packet. This means that for each host pair there can only be communications of one service at a time. Maybe future versions of the framework could work around this limitation to work with more complex traffic events;

- When forwarding is necessary, the current default forwarding algorithm of the framework will find a suitable path. A more interesting approach would be to have a default forwarding solution for each service, or even for each host pair.

Chapter 8

SDN Experimental Scenario & Algorithm Results

In this Chapter, we present the scenario we developed to evaluate the performance of our framework and the effectiveness of our fairness mechanism. It showcases the emulation environment, the services considered and how to generate traffic.

8.1 Experimental Environment

The involvement of this work in research projects allows us to have information on the deployment layout of Internet of Things (IoT) devices in smart-city scenarios. More specifically, we have access to the topological information of the WiFi Access Points and other types of nodes (e.g. 5G capable) in the Aveiro Tech City Living Lab (ATCLL) Lab¹.

ATCLL is an advanced communications infrastructure and urban management platform deployed in the city of Aveiro, Portugal. ATCLL acts like a big scale open laboratory at the disposal of researchers and enterprises interested in developing or testing products or services.

To accomplish this, ATCLL has equipment around the city's urban area which are able to perform monitoring or interactive functions with users: smart lamps equipped with traffic sensors, like GPS; Lidars and Radars (e.g., for vehicles communications); public transport installed with sensors for location, velocity, temperature and noise levels, 4G/5G nodes, WiFi Access Point (AP), and many other sensors, interfaces and technologies that are out of the scope of this work, see Figure 8.1.

The ATCLL project is relevant to our work because of the layout of the WiFi infrastructure².

¹<https://www.aveirotechcity.pt/pt/atividades/aveiro-tech-city-living-lab>

²<https://www.google.com/maps/d/viewer?mid=1p7QSVjJk15n6IiXf8c5DY6Lr9w6aaZb5>

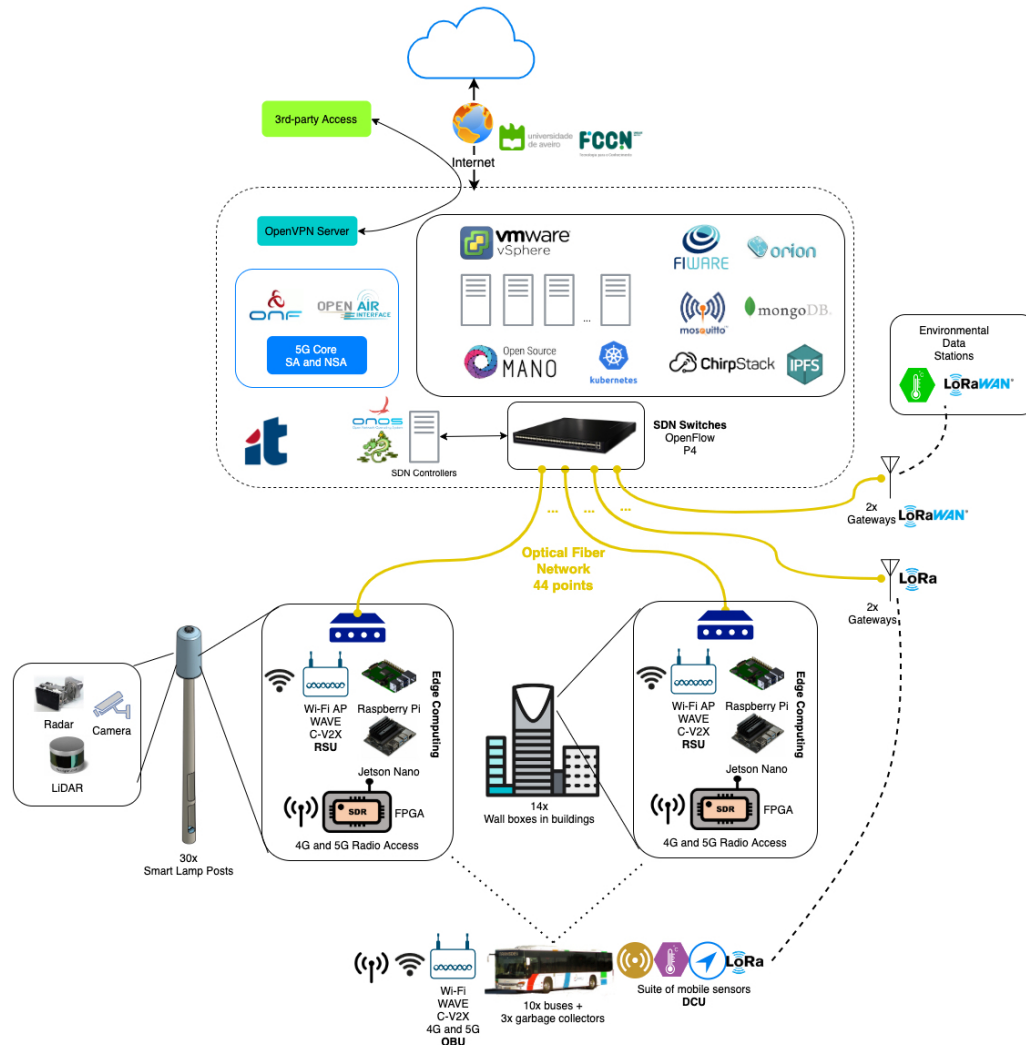


Figure 8.1: Aveiro Open Lab technologies [aveirotechcity, 2021]

It has around 32 AP in the urban area that we can use to validate our framework and test its performance, see Figure 8.2. Although we don't have physical access to the ATCLL infrastructure we compiled the required information to emulate its behaviour. By analysing the ATCLL documentation and requesting some additional artefacts we planned the features of the virtual topology to be faithful to its real-world counterpart:

- Two centralised backbone SDN switches, OF and P4 capable;
- SDN cluster switch to connect to a group of APs;
- 10Gbps Fiber optics physical connections;
- 2.4GHz wireless connections;
- APs have a physical connection to their cluster switch;
- APs have mesh wireless connections with other AP in a cluster;

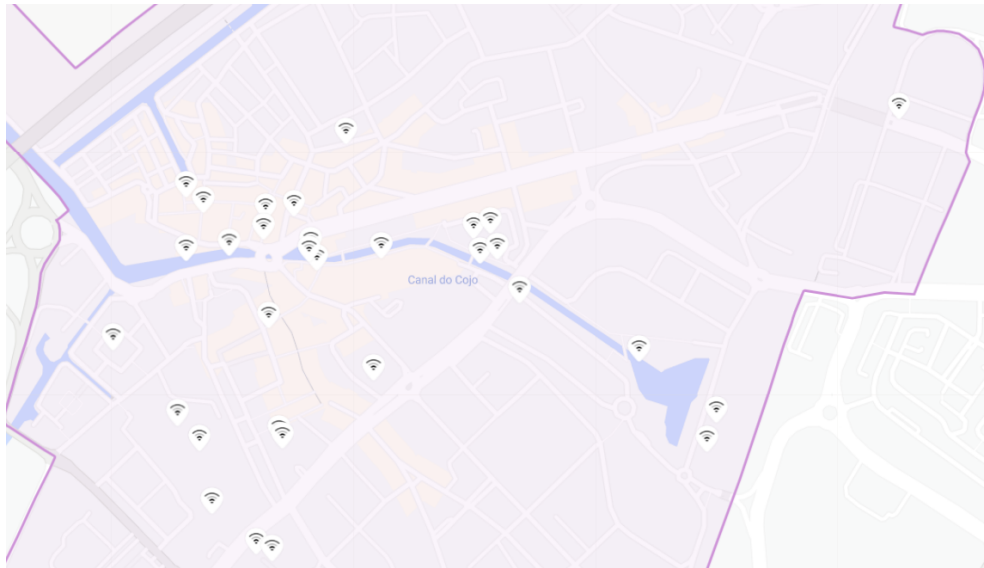


Figure 8.2: Aveiro open lab wifi topology

Keep in mind that the devices in the ATCLL also support connection frequencies of 5GHz. The initial plan was to experiment with both values but due to time constraints, see Section 4.2.3, we only utilised the frequency of 2.4GHz.

With the information gathered so far, we can build a draft of the experimental scenario in Mininet-WiFi, see Figure 8.3.

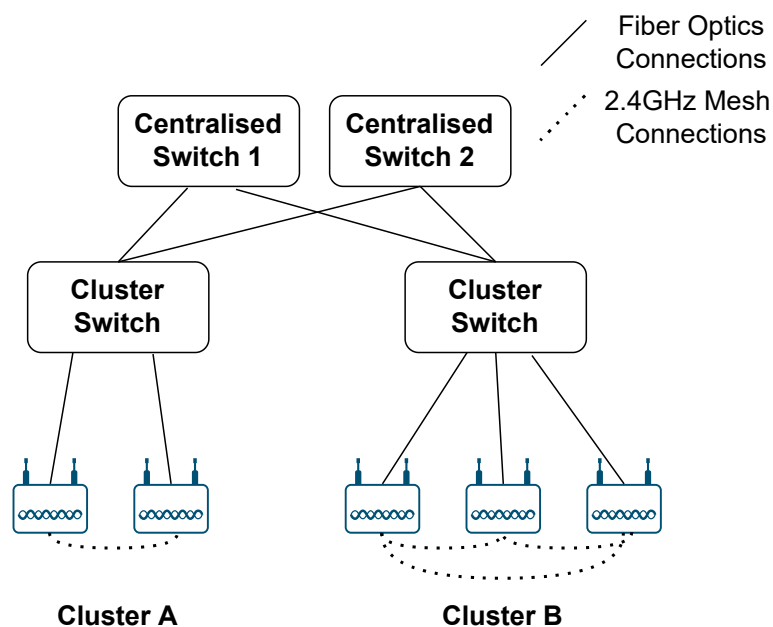


Figure 8.3: Virtual topology draft

Because the range of the APs is still missing, we are unable to distribute the devices through clusters: we don't know the neighbours that are in communication

range with a given AP. To obtain this knowledge we adopted a theoretical approach and calculated it utilising link budget equations, as follows.

Link budget equations are formulas used in telecommunication systems to aid in design choices. It involved properties of the transmitter, receiver and communication medium [Rikkinen et al., 2020].

We started with the following link budget equation [Travis Fagerness, 2015]:

$$P_{RX} = P_{TX} + G_{TX} + G_{RX} - L_M - L_{FS}$$

where:

- P_{RX} is received power strength (dBm);
- P_{TX} is transmitted output power (dBm);
- G_{TX} is transmitted antenna gain (dBi);
- G_{RX} is received antenna gain (dBi);
- L_M is miscellaneous losses (weather conditions, obstacles, antenna polarisation mismatch, multi path propagation and other losses) (dB);
- L_{FS} is path loss, the loss due to propagation between the transmitting and receiving antennas (dB).

Knowing that the ATCLL deploys devices similar to Cisco Catalyst 9130AX [Cisco, 2021], the specification values of this machine can help to resolve the link budget equation. For instance, for IEEE 802.11n it uses the HT20 (High Throughput with the channel of 20Mhz of bandwidth) and MCS31 as a Modulation Coding Scheme at a base frequency of 2.4GHz. Thus replacing the equation values:

- The P_{RX} will be replaced by the AP's *Receiver Sensibility*, the minimum amount of power needed to receive a message (**-75dBm**);
- P_{TX} for 2.4 GHz is **20dBm**;
- The peak antenna gain for 2.4 GHz is documented as 4 dBi. The average G_{TX} and G_{RX} were needed for the formula, so we considered it to be approximately **0 dBi**, using Figure 8.4 as a reference;
- L_M is assumed to be around **10dB** due to the ATCLL scenario being outdoors with many buildings blocking line-of-sight, [Aerohive Networks, 2014; Travis Fagerness, 2015];
- L_{FS} can be replaced by a formula for path loss:

$$L_{FS} = \left(\frac{\lambda}{4\pi R}\right)^2$$

where: λ is the base frequency (2.4 GHz) and R is the distance between sender and receiver;

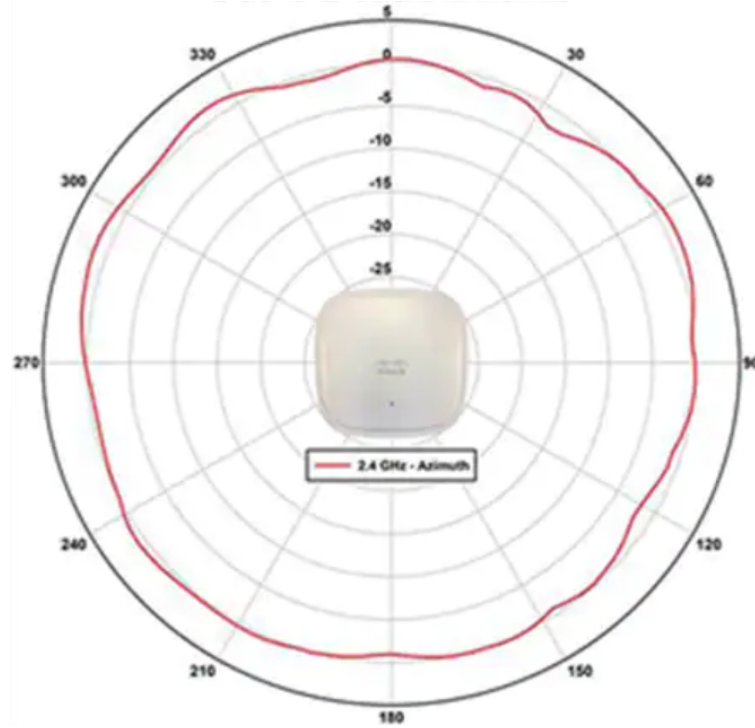


Figure 8.4: Cisco C9130 AXI 2.4GHz Antenna patterns [Cisco, 2021]

This radius variable, R , is going to give us each AP range. This way we can know the neighbouring AP that can communicate with each other.

Changing the L_{FS} formula to accept frequency in MHz and R in meters, and replacing the other variable values, we get the final equation:

$$P_{RX} = P_{TX} + G_{TX} + G_{RX} - L_M - (-27.55 + 20 \log_{10}(2400) + 20 \log_{10}(R_{meters}))$$

$$R \approx \text{antilog} \left(\frac{P_{TX} + G_{TX} + G_{RX} - L_M - P_{RX} - (-27.55 + 20 \log_{10}(2400))}{20} \right) \text{meters}$$

$$R = \text{antilog} \left(\frac{20\text{dBm} + 0\text{dBi} - 10\text{dB} + 0\text{dBi} - (-75)\text{dBm} - 40.05}{20} \right) \text{meters}$$

$$R \approx 176.72\text{meters}$$

We conclude that the range of each cisco Wifi Access Point, with outdoor obstacles and using 2.4 GHz should be around 177 meters. With this information, we can divide the devices between clusters, putting APs that are in range of a mesh connection in the same cluster.

Finally, with the longitude and latitude values of each AP collected using ATCLL resources [Aveiro Tech City Living Lab Team, 2021], a scenario for the virtual topology can be built.

The final layout, see Figure 8.5, is a balance between accuracy of the topology of Aveiro and the ideal conditions to evaluate the framework and fairness mechanism. This was done to avoid isolating APs from the rest of the topology, which is counter-intuitive in the scenario we are trying to achieve and to allow easier scaling up or down of the virtual topology if/when necessary.

There are 6 clusters, each cluster has 6 access points (total of 36 APs), wireless interfaces use IEEE 802.11n 2.4GHz, wired connections use 10Gbps fibre optics, APs are deployed in intervals of 20 meters, and clusters are spaced out 60 meters from each other.

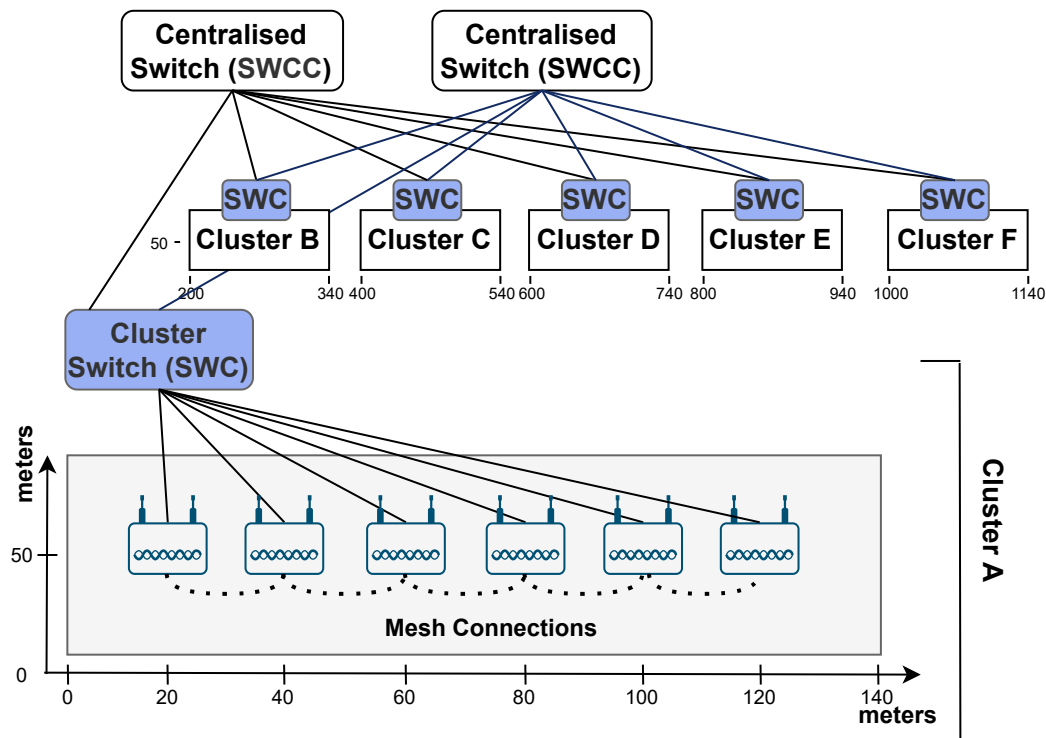


Figure 8.5: Mininet-Wifi Virtual Topology Diagram

8.1.1 Problems with Mininet-Wifi

While using iPerf to do some traffic simulations, see Section 8.1.3, some problems were revealed:

- **ARP Parsing Problems** When a source node needs to send traffic to a destination node, it first sends an ARP request to the network. The intended behaviour is for the ARP request to reach the controller, then the *HostLocationProvider* class parses the information, and finally, the packet is sent back to the network. This allows the controller to associate MAC addresses to IP addresses. Unfortunately, the *HostLocationProvider* was unable to resend the packet due to conflicts with mesh connections. We had to manually forward

the ARP requests and replies, leaving the controller incapable of associating MAC addresses to IP addresses.

- **Losses between station and AP** Sending service data using iPerf was reporting losses of up to 90% of the traffic between the wireless connection of station and AP, even though there were near 0% losses in the mesh links;
- **Using Hosts** To avoid the losses between station and AP we opted to use hosts instead of stations. These modifications weren't well handled by Mininet-Wifi, since the mesh connections would disconnect at the start of the simulation.
- **Using Wired connections** Using the *TCLink* class to have wired connections in an effort to avoid the station from AP losses was also ineffective. The target node of the traffic when prompted with an ARP request, didn't send back an ARP reply, so there was no traffic flow.

Trying to resolve these issues or work around them was deemed ineffective. The environment of the experimental setup was regarded as non deterministic and thus was abandoned.

As a result, the team shifted to the Mininet emulator. Although it does not have wireless capabilities, it gathers the necessary conditions to evaluate the fairness mechanism: Openflow (OF) enabled devices and hosts, adjustable link loss rates, bandwidth and delays, and flexible workload of traffic for different services. As way to gather feedback on this emulator change, we participated in the seminar Rede Temática de Comunicações Móveis (RTCM) 2022. The advice of the participants helped us to build a more adequate scenario of the emulated topology.

8.1.2 Mininet Environment

The environment built in Mininet keeps the same cluster layout, the same number of clusters and devices, and the same number of connections, see Figure 8.6. The differences are that we use hosts instead of stations, switches instead of APs, links now use the *TCLink* class, switch to switch connections have 54Mbps of bandwidth and 1.2% loss probability, switch to host connections have 54Mbps of bandwidth and are lossless, and the remaining links have 1Gbps of bandwidth and are also lossless.

The "loss probability" is a parameter of the *TCLink* class that informs Mininet on the average packet loss of a link when a single flow is transported through that connection, assuming the link has enough available bandwidth. The actual observable loss of a link will depend on the "loss probability" value, the link congestion, and the size of the service flow. The loss probability values came from [Miguel, 2020].

The 1Gbps of bandwidth is the maximum of the *TCLink* class and is used to represent Fiber optics wired connections, the 54Mbps is used to simulate the wireless connections (theoretical maximum for IEEE 802.11g [Intel, 2022]).

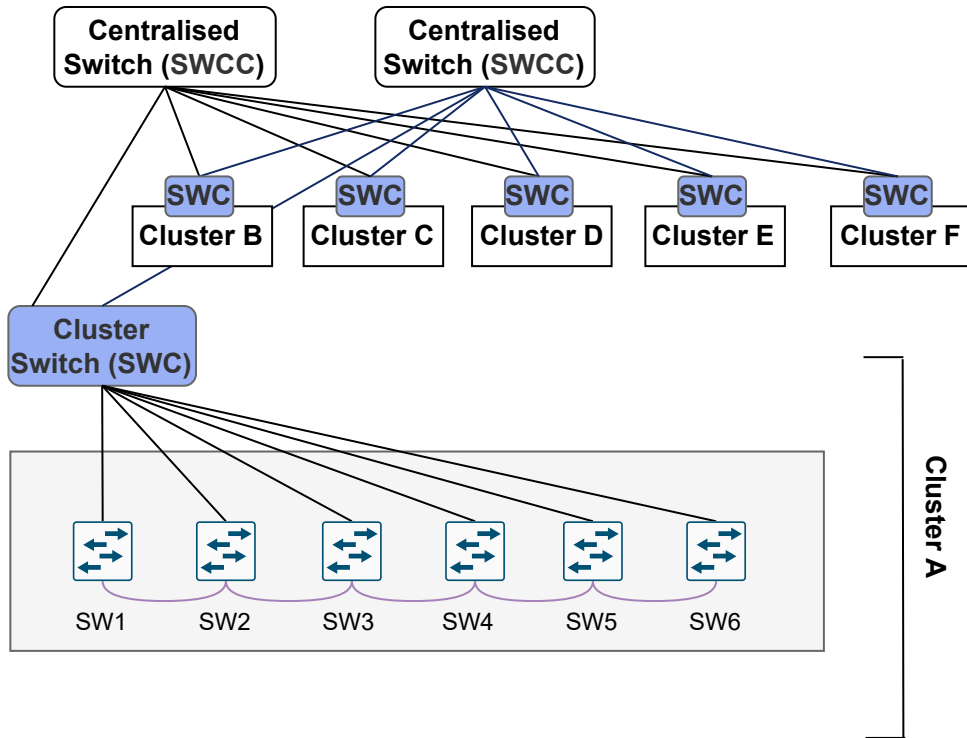


Figure 8.6: Mininet Virtual Topology Diagram

8.1.3 Generate Traffic

There are three types of services considered: real-time video (RT-VT), voice traffic and burst-user-driven traffic (BUD), which is mainly associated with patterns of web traffic. To model these services we consider the traffic models proposed in [Navarro-Ortiz et al., 2020]. The settings considered per service are summarised in Table 8.1.

Table 8.1: Services and traffic models [Navarro-Ortiz et al., 2020]

Service	Duration	Protocol	Required bandwidth
RT-VT	202 s	UDP	CBR \approx 1.5Mbps
Voice	202 s	UDP	EVS-WB codec with 24.4 Kbps
BUD	n/a	TCP	With a maximum of 12 Mbps

The RT-VT and voice services are modelled using the UDP transport protocol, while the BUD uses the TCP protocol.

To generate the traffic of each service we resorted to iPerf. Furthermore, we adapted the GitHub project MineEvents³, which provides a tool to define iPerf events in Mininet networks. With this tool it is possible to simulate traffic for each of the services, assuming we provide it with a JSON file of the events in the simulation:

³<https://github.com/cgiraldo/minievents>

```
[{ "time": 0, "type": "iperf", "params": { "src": "host1",
"dst": "host4", "protocol": "UDP", "bw": 1500000, "duration": 202
} },

{ "time": 1, "type": "iperf", "params": { "src": "host2",
"dst": "host5", "protocol": "UDP", "bw": 24400, "duration": 202
} },

{ "time": 2, "type": "iperf", "params": { "src": "host3",
"dst": "host6", "protocol": "TCP", "num": 10, "buff": 1536000
} },

{ "time": 250, "type": "wait", "params": {} } ]
```

Each event contains a simulation time to start sending packets, host and source nodes, and protocol type. With UDP the JSON also contains bandwidth value in bits of the flow and the duration of the traffic flow in seconds. With TCP it contains the information buffer size in bytes (simulating Web Traffic files), and the number of times the buffer will be sent (simulating different Web Traffic resources).

8.1.4 Energy Formulas

To populate the energy table of the database, the energy models of [Kaup et al., 2018] were applied. Two types of devices are considered for the switches, as they led to different costs in the wired and wireless links: a Raspberry Pi 3B model and a Cubiboard 3. Table 8.2 presents the power consumption in Watts of the ethernet and wlan interfaces, in downlink (dn) and uplink (up) traffic, according to data rate (r) in Mbps. Furthermore, it also documents the energy consumption of CPU loads and the idle cost of interfaces.

The switches on the edge of each cluster can be Raspberry Pi or Cubiboard machines: we considered that switches with odd identifiers were Raspberry Pi machines, e.g. $sw1$, $sw3$, $sw9$, and switches with even identifiers were Cubiboard machines, e.g. $sw2$, $sw14$, $sw20$. The remaining devices (cluster switches and centralised switches) were assumed to be Raspberry Pi machines since they are the machines with more computational power and are more easily found for purchase.

8.2 Experiences & Results

This section documents each of the experiments conducted to evaluate the performance of the framework and the fairness mechanism. It details the characteristics of the environment and provides analyses of the achieved results.

We aim to present experiments with different service flows, a diverse number of

Table 8.2: Power model (Watts), with r - datarate in Mbps [Kaup et al., 2018]

Raspberry Pi		
$P_{eth,up}(r)$	$= 26.2e^{-06}r^2 + 0.357e^{-03}r + 0.007$	
$P_{eth,dn}(r)$	$= -4.33e^{-06}r^2 + 0.485e^{-03}r - 0.007$	
$P_{wlan,up}(r)$	$= -0.25e^{-06}r^2 + 1.99e^{-03}r - 0.072$	
$P_{wlan,dn}(r)$	$= 1.85e^{-03}r^2 + 13.5e^{-03}r + 0.072$	
P_{cpu}	$= 0.6191$	$P_{wlan,idle} = 0.7645$ $P_{eth,idle} = -0.1176$
Cubiboard 3		
$P_{eth,up}(r)$	$= -17.6e^{-06}r^2 + 6.13e^{-03}r - 0.056$	
$P_{eth,dn}(r)$	$= -20.9e^{-06}r^2 + 2.50e^{-03}r + 0.056$	
$P_{wlan,up}(r)$	$= -0.307e^{-03}r^2 + 22.8e^{-03}r + 0.011$	
$P_{wlan,dn}(r)$	$= 0.137e^{-03}r^2 + 6.33e^{-03}r - 0.011$	
P_{cpu}	$= 1.037$	$P_{wlan,idle} = 0.3060$ $P_{eth,idle} = 0.2240$

active host machines and scenarios with *inter* cluster service flows (flows between different clusters) and scenarios with *intra* cluster service flows (flows with source and target host within the same cluster). The *inter* cluster scenarios also included a *partial* - with a reduced set of nodes and a *full* validation - with all the nodes and clusters.

In the majority of experiences, the focus is to compare the performance of our custom fairness heuristic, see Section 7.3.1, against the K-shortest path that Open Network Operating System (ONOS) provides, see Section 7.3.2, with regards to performance metrics like **link utilisation**, **jitter** and **packet loss**.

Link utilisation can be considered by looking at the distribution of the number of packets that a switch transports in all the appropriate interfaces, for a given flow. Flows with different host targets, when reaching the same switch, might have a different number of appropriate ports to use to forward packets: a traffic flow that reaches a cluster switch (swc#) has 5 potential ports to forward packets to if the target flow is in the same cluster and only 2 ports if the flow needs to reach a different cluster. A just/fair algorithm should try to approximate the values of these ports.

The simulation results of iPerf report useful information regarding the flow of a UDP service. This is another way to compare the two forwarding solutions since we can collect data from the perspective of the video and voice services: total information sent in KBytes, average transfer speed in Kbps, **average jitter** in milliseconds, **percentage of lost** datagrams and total datagrams generated.

In each one of these experiments, the preferred forwarding algorithm is selected in the platform, then the Mininet topology is initiated, and finally, the traffic simulations of iPerf start. This means that the forwarding algorithms are compared in the most similar conditions possible.

Regarding the iPerf generated traffic, consecutive service flows are spaced out

5 seconds between each other to circumvent ONOS limitations. When trying to generate traffic events each second, the controller would end up being bombarded with packets, and eventually would stop installing new forwarding rules. This is because the first packet would reach the controller, the forwarding algorithm would find a path and the necessary flow rules would be installed in the required devices. The issue is on the fact that these rules become installed but are not yet active, they are on a `PENDING_ADD` state. An ONOS structure, the Flow Rule Subsystem⁴, is responsible for observing the rules of the device's tables and assessing their adequate state. The rule will only be active when this subsystem, which acts independently from the structure that installs the rules, changes the state of the rule to `ADDED`. This can take some time, and in that time the devices still send incoming packets to the controller. With more incoming packets, the flow rule subsystem becomes slower, until ONOS stops operating properly.

Trying to see if our algorithm was the root problem of the ONOS slowdown, we conducted the same experiences with the ONOS default settings and the default forwarding algorithm that uses hop count (which is of low algorithm complexity), but the same problem persisted.

Furthermore, we also tried to work around this issue by generating “pre-traffic” before each iPerf event: for each service event in the JSON, we also generate a few packets with the same source and destination hosts. This was done with the intent of giving more time between the first packets of a service that reach the controller, and the time the flow entries become active.

Generating iPerf events with 5 seconds intervals and adding the “pre-traffic”

⁴<https://wiki.onosproject.org/display/ONOS/Flow+Rule+Subsystem>

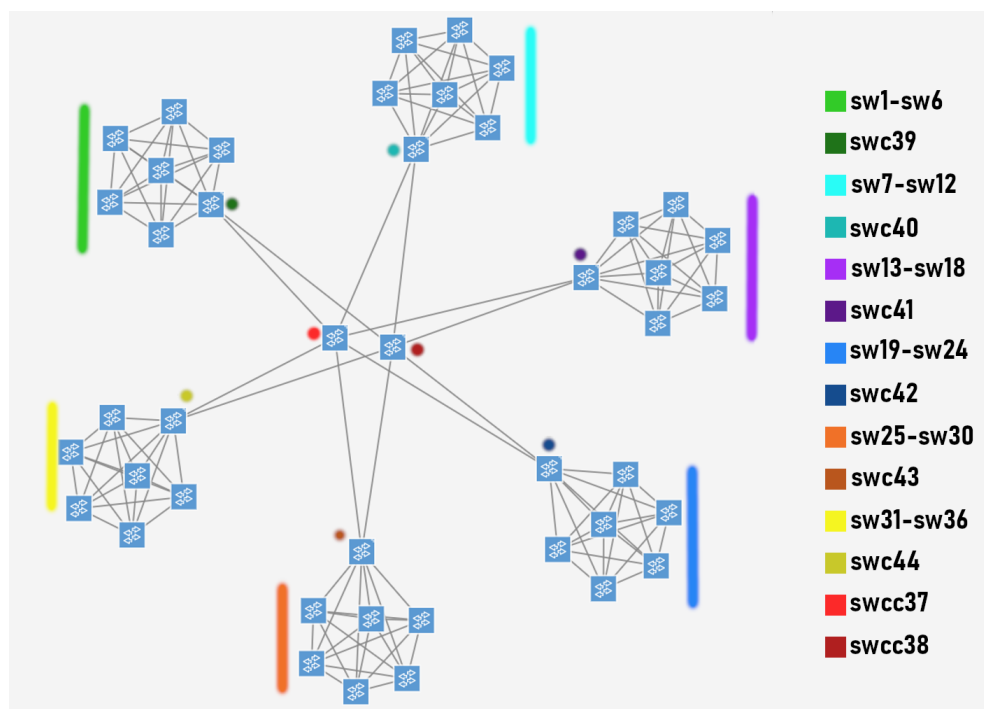


Figure 8.7: Topology Representation in the ONOS GUI

events seem to solve the previous issues.

If the description of an experiment doesn't say otherwise, the reader should assume that there are 6 clusters (cluster identifier 1 through 6), and each cluster has 6 edge switches (e.g. sw1 through sw6 in cluster1, sw7 through sw12 in cluster2). Each edge switch has 3 hosts connected to it for video, voice and BUD services.

There is 1 "cluster switch" in each cluster (e.g. swc39 in cluster 1, swc40 in cluster 2, swc44 in cluster 6) and different clusters are connected with 2 "central switches" (swcc37 and swcc38). Figure 8.7 contains an image of the topology provided by the ONOS Graphical User Interface (GUI), with the respective colour scheme.

Additionally, keep in mind that each target host of the BUD traffic also responds to the source host, generating an additional flow that needs to be forwarded. In the experiments, we identify this flow by BUDr.

8.2.1 Heuristic Validation

This experiment aims to theoretically validate how good the solutions that the fairness heuristic gives are compared to the optimal solutions of the formulation.

Due to the fact that the formulation's execution time scales with the complexity of the scenario (more nodes and service flows), we opted to be conservative and pragmatic and devised the following scenario to avoid unreasonable execution durations.

The experiment considers three service flows, one for video, one for voice, and one for BUD: the video service is generated by a machine connected to sw1 and targets a machine connected with sw3; the voice service is generated by a machine connected to sw2 and targets a machine connected with sw5; the BUD service is generated by a machine connected to sw3 and targets a machine connected to sw6;

After running the formulation solver and the implemented heuristic, the results show that the heuristic obtained the same value for the objective function as the formulation, which is the best possible result. These values are documents in the conference paper to submit, see Section 1.2. The value of the objective function is given by the sum of link weights, while the link weight is the maximum normalised value of the three objectives.

This experiment confirms that the heuristic produces very good results and that it can be useful in realistic scenarios. Increasing the number of service flows might cause some discrepancies between the values of the objective function, which is expected.

8.2.2 Partial Inter Cluster

In this experiment, we reproduce the environment where service traffic needs to be forwarded to another cluster, so only two clusters are involved in this scenario.

Host machines connected to the edge switches of cluster 1 produce traffic that needs to reach the hosts of cluster 3. For example, sw1 has host1, host2 and host3 producing traffic of video, voice and BUD, respectively. This traffic needs to reach the hosts connected to sw13 (host37, host38 and host39) in cluster 3. In total, there are 18 hosts producing traffic in cluster 1 (6 video services, 6 voice services and 6 BUD services), plus 6 hosts in cluster 3 that respond to BUD traffic, BUDr.

Table 8.3 provides useful information detailing the iPerf traffic events of the simulation.

Table 8.3: Partial Inter Cluster Flows

ID	Start Time	Source Target	Service
1	0	sw1 : sw13	video
2	5		voice
3	10		BUD
4	-	sw13 : sw1	BUDr
5	15	sw2 : sw14	video
6	20		voice
7	25		BUD
8	-	sw14 : sw2	BUDr
9	30	sw3 : sw15	video
10	35		voice
11	40		BUD
12	-	sw15 : sw3	BUDr
13	45	sw4 : sw16	video
14	50		voice
15	55		BUD
16	-	sw16 : sw4	BUDr
17	60	sw5 : sw17	video
18	65		voice
19	70		BUD
20	-	sw17 : sw5	BUDr
21	75	sw6 : sw18	video
22	80		voice
23	85		BUD
24	-	sw18 : sw6	BUDr

The resulting paths that the fairness algorithm and the K-shortest path found for each flow are illustrated in Table 8.4.

Looking at the paths obtained by the fairness heuristic we can see that 5 out of the 18 flows in the iPerf JSON used swcc38 (flows 2, 5, 17, 18 and 19) while the remaining used swcc37. Since both switches use the Raspberry Pi energy formulas, this separation is caused by the loss and delay objectives that variate throughout the iPerf simulation. When the links that pass through swcc37 are too congested, the heuristic opts to use swcc38 in the service paths instead.

Table 8.4: Partial Inter Cluster Paths

ID	Path Fairness	Path K-shortest
1	{sw1, swc39, swcc37, swc41, sw13}	{sw1, swc39, swcc38, swc41, sw14, sw13}
2	{sw1, swc39, swcc38, swc41, sw13}	{sw1, swc39, swcc38, swc41, sw13}
3	{sw1, swc39, swcc37, swc41, sw13}	{sw1, sw4, swc39, swcc38, swc41, sw14, sw13}
4	{sw13, swc41, swcc37, swc39, sw1}	{sw13, sw14, swc41, swcc38, swc39, sw4, sw1}
5	{sw2, swc39, swcc38, swc41, sw14}	{sw2, swc39, swcc38, swc41, sw14}
6	{sw2, swc39, swcc37, swc41, sw14}	{sw2, swc39, swcc38, swc41, sw14}
7	{sw2, swc39, swcc37, swc41, sw14}	{sw2, swc39, swcc38, swc41, sw14}
8	{sw14, swc41, swcc37, swc39, sw2}	{sw14, swc41, swcc38, swc39, sw2}
9	{sw3, swc39, swcc37, swc41, sw15}	{sw3, swc39, swcc38, swc41, sw14, sw15}
10	{sw3, swc39, swcc37, swc41, sw15}	{sw3, swc39, swcc38, swc41, sw15}
11	{sw3, swc39, swcc37, swc41, sw15}	{sw3, sw4, swc39, swcc38, swc41, sw14, sw15}
12	{sw15, swc41, swcc37, swc39, sw3}	{sw15, sw14, swc41, swcc38, swc39, sw4, sw3}
13	{sw4, swc39, swcc37, swc41, sw16}	{sw4, swc39, swcc38, swc41, sw16}
14	{sw4, swc39, swcc37, swc41, sw16}	{sw4, swc39, swcc38, swc41, sw16}
15	{sw4, swc39, swcc37, swc41, sw16}	{sw4, swc39, swcc38, swc41, sw16}
16	{sw16, swc41, swcc37, swc39, sw4}	{sw16, swc41, swcc38, swc39, sw4}
17	{sw5, swc39, swcc38, swc41, sw17}	{sw5, swc39, swcc38, swc41, sw14, sw17}
18	{sw5, swc39, swcc38, swc41, sw17}	{sw5, swc39, swcc38, swc41, sw17}
19	{sw5, swc39, swcc38, swc41, sw17}	{sw5, sw4, swc39, swcc38, swc41, sw14, sw17}
20	{sw17, swc41, swcc37, swc39, sw5}	{sw17, sw14, swc41, swcc38, swc39, sw4, sw5}
21	{sw6, swc39, swcc37, swc41, sw18}	{sw6, swc39, swcc38, swc41, sw18}
22	{sw6, swc39, swcc37, swc41, sw18}	{sw6, swc39, swcc38, swc41, sw18}
23	{sw6, swc39, swcc37, swc41, sw18}	{sw6, swc39, swcc38, swc41, sw18}
24	{sw18, swc41, swcc37, swc39, sw6}	{sw18, swc41, swcc38, swc39, sw6}

In contrast, the ONOS K-shortest algorithm chooses the path with less energy consumption, so using `swcc37` or `swcc38` should be of equal cost. However, because the algorithm always uses the first path retrieved, when multiple paths have the minimum energy cost, the only switch ever used in the paths is `swcc38`.

Figure 8.8 gives us the number of packets sent by `swc39` in the simulations, which helps to illustrate the behaviour described: with the K-shortest algorithm, the `swc39` switch never uses the `swc39-swcc37` link so all the packets are sent through `swc39-swcc38`; with our fairness heuristic, there is shared link utilisation since around 2/3 of all packets are sent using `swc39-swcc37` and 1/3 using `swc39-swcc38`.

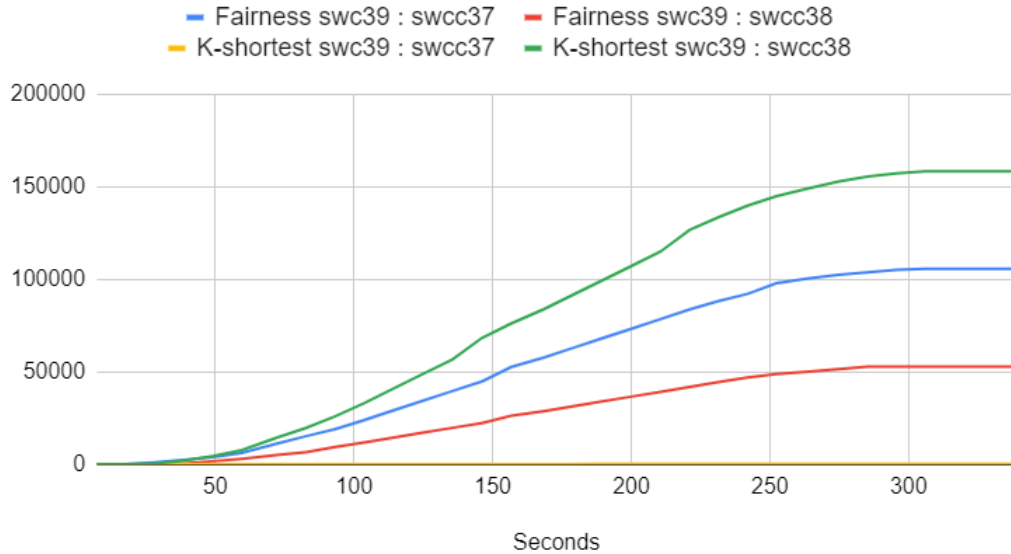


Figure 8.8: Partial Inter Cluster - Packets Sent Comparison

Furthermore, the paths obtained by the K-shortest path algorithm follow a pattern. The flows where the source and target edge switches have even identifiers (Cubiboard machines) always opt to utilise the wired link between `sw` and `swc` (Cubiboard to Raspberry Pi wired and Raspberry Pi to Cubiboard wired), independently of the service, see flows 5-8, 13-16 and 21-24. The flows where edge switches have odd identifiers (Raspberry Pi) and the service is voice (flows 2, 10 and 18) retrieved paths that prefer the wired link connections, as well.

Additionally, for all flows where edge switches have odd identifiers and the service is video, BUD or BUDr, it is cheaper to use the Raspberry Pi to Cubiboard wired link plus Cubiboard to Raspberry Pi wireless link (flows 1, 9 and 17 for video service) than pay the cost of Raspberry Pi to Raspberry Pi wired.

Finally, for all paths where edge switches have odd identifiers and the service is BUD or BUDr, besides the link preferences described previously, it is cheaper to use Raspberry Pi to Cubiboard wireless link plus Cubiboard to Raspberry Pi wired link (flows 3, 11 and 19 for BUD) than pay the cost of Raspberry Pi to Raspberry Pi wired.

Regarding the observed loss and jitter of the UDP services, Table 8.5 contains the calculated mean and standard deviation of the values of the iPerf metrics for the generated video and voice services. Analysing the data, the average information

Table 8.5: Partial Inter Cluster - iPerf UDP Metrics

Service	Information Sent (KB)	Speed (Kbps)	Jitter (ms)	Lost Percentage	Datagrams Sent
Fairness Heuristic					
video	36846.9 ±41.805	1508.7 ±8.361	0.02 ±0.015	0.41% ±0.046	25767.8 ±0.408
voice	601.5 ±1.23	24.2 ±0.16	0.026 ±0.008	0.48% ±0.149	421.2 ±0.408
K-shortest ONOS					
video	36625.1 ±264.396	1510.4 ±15.530	0.03 ±0.025	1.01% ±0.684	25767.3 ±0.516
voice	603.8 ±1.169	24.15 ±0.217	0.023 ±0.038	0.20% ±0.178	421.5 ±0.548

sent (bytes and datagrams) and the transfer speed reports similar values, independently of the forwarding solution, which is expected. Due note, the standard deviation of the video service, using the K-shortest path, for these metrics is significantly higher.

The jitter values are also identical between services and forwarding mechanisms and are of overall negligible magnitude. Maybe in a more complex scenario, with heavier loads and lengthier service events, a distinction might become apparent.

The loss values for video and voice, while using the fairness heuristic, are similar probably because both services use identical paths: both use the wired link from swc to sw in cluster 3. In contrast, the voice service of the K-shortest path, which also uses this link, reports a smaller observable loss. This is caused by the fact that the video service, with the K-shortest path algorithm, uses a different path and so the voice flow doesn't compete or interfere with it. This also justifies the higher loss value of the video service with the K-shortest path: it uses wireless links, that the voice service doesn't use, that have a higher base loss probability, recall Section 8.1.2.

8.2.3 Full Inter Cluster

This experiment is similar to the one performed in the previous section, where there is traffic that needs to be forward to another cluster in the topology. The difference in on the fact that now, all of the clusters generate or receive service traffic. This was done in an effort to see if the framework would handle an amount of traffic more reminiscent of what can happen in a smart-city environment and to analyse how the forwarding decisions would change.

Host machines connected to the edge switches of cluster 1 generate traffic that needs to reach the hosts of cluster 4, traffic of cluster 2 targets hosts of cluster 5 and traffic of cluster 3 is directed to the hosts of cluster 6.

Clusters 1 through 3 have 18 hosts each producing traffic, 6 video services, 6 voice

Table 8.6: Full Inter Cluster Flows

ID	Start Time	Source Target	ID	Start Time	Source Target	ID	Start Time	Source Target	Service
25	0	sw1 : sw19	49	90	sw7 : sw25	73	180	sw13 : sw31	video
26	5		50	95		74	185		voice
27	10		51	100		75	190		BUD
28	-	sw19 : sw1	52	-	sw25 : sw7	76	-	sw31 : sw13	BUDr
29	15	sw2 : sw20	53	105	sw8 : sw26	77	195	sw14 : sw32	video
30	20		54	110		78	200		voice
31	25		55	115		79	205		BUD
32	-	sw20 : sw2	56	-	sw26 : sw8	80	-	sw32 : sw14	BUDr
33	30	sw3 : sw21	57	120	sw9 : sw27	81	210	sw15 : sw33	video
34	35		58	125		82	215		voice
35	40		59	130		83	220		BUD
36	-	sw21 : sw3	60	-	sw27 : sw9	84	-	sw33 : sw15	BUDr
37	45	sw4 : sw22	61	135	sw10 : sw28	85	225	sw16 : sw34	video
38	50		62	140		86	230		voice
39	55		63	145		87	235		BUD
40	-	sw22 : sw4	64	-	sw28 : sw10	88	-	sw34 : sw16	BUDr
41	60	sw5 : sw23	65	150	sw11 : sw29	89	240	sw17 : sw35	video
42	65		66	155		90	245		voice
43	70		67	160		91	250		BUD
44	-	sw23 : sw5	68	-	sw29 : sw11	92	-	sw35 : sw17	BUDr
45	75	sw6 : sw24	69	165	sw12 : sw30	93	255	sw18 : sw36	video
46	80		70	170		94	260		voice
47	85		71	175		95	265		BUD
48	-	sw24 : sw6	72	-	sw30 : sw12	96	-	sw36 : sw18	BUDr

services and 6 BUD services, for a total of 54 hosts, and clusters 4 through 6 also produce BUDr responses to reply to BUD traffic.

Useful and detailed information about the iPerf traffic events can be found in Table 8.6 while the resulting paths for each of the flows are documented in Table 8.7.

Table 8.7: Full Inter Cluster Paths

ID	Path Fairness	Path K-shortest
25	{sw1, swc39, swcc38, swc42, sw19}	{sw1, swc39, swcc38, swc42, sw24, sw19}
26	{sw1, swc39, swcc38, swc42, sw19}	{sw1, swc39, swcc38, swc42, sw19}
27	{sw1, swc39, swcc37, swc42, sw19}	{sw1, sw4, swc39, swcc38, swc42, sw24, sw19}

Continuation of Table 8.7		
ID	Path Fairness	Path K-shortest
28	{sw19, swc42, swcc37, swc39, sw1}	{sw19, sw24, swc42, swcc38, swc39, sw2, sw1}
29	{sw2, swc39, swcc38, swc42, sw20}	{sw2, swc39, swcc38, swc42, sw20}
30	{sw2, swc39, swcc37, swc42, sw20}	{sw2, swc39, swcc38, swc42, sw20}
31	{sw2, swc39, swcc37, swc42, sw20}	{sw2, swc39, swcc38, swc42, sw20}
32	{sw20, swc42, swcc37, swc39, sw2}	{sw20, swc42, swcc38, swc39, sw2}
33	{sw3, swc39, swcc37, swc42, sw21}	{sw3, swc39, swcc38, swc42, sw22, sw21}
34	{sw3, swc39, swcc37, swc42, sw21}	{sw3, swc39, swcc38, swc42, sw21}
35	{sw3, swc39, swcc37, swc42, sw21}	{sw3, sw4, swc39, swcc38, swc42, sw22, sw21}
36	{sw21, swc42, swcc37, swc39, sw3}	{sw21, sw24, swc42, swcc38, swc39, sw2, sw3}
37	{sw4, swc39, swcc37, swc42, sw22}	{sw4, swc39, swcc38, swc42, sw22}
38	{sw4, swc39, swcc37, swc42, sw22}	{sw4, swc39, swcc38, swc42, sw22}
39	{sw4, swc39, swcc38, swc42, sw22}	{sw4, swc39, swcc38, swc42, sw22}
40	{sw22, swc42, swcc37, swc39, sw4}	{sw22, swc42, swcc38, swc39, sw4}
41	{sw5, swc39, swcc38, swc42, sw23}	{sw5, swc39, swcc38, swc42, sw24, sw23}
42	{sw5, swc39, swcc37, swc42, sw23}	{sw5, swc39, swcc38, swc42, sw23}
43	{sw5, swc39, swcc37, swc42, sw23}	{sw5, sw4, swc39, swcc38, swc42, sw24, sw23}
44	{sw23, swc42, swcc37, swc39, sw5}	{sw23, sw24, swc42, swcc38, swc39, sw4, sw5}
45	{sw6, swc39, swcc38, swc42, sw24}	{sw6, swc39, swcc38, swc42, sw24}
46	{sw6, swc39, swcc38, swc42, sw24}	{sw6, swc39, swcc38, swc42, sw24}
47	{sw6, swc39, swcc38, swc42, sw24}	{sw6, swc39, swcc38, swc42, sw24}
48	{sw24, swc42, swcc37, swc39, sw6}	{sw24, swc42, swcc38, swc39, sw6}
49	{sw7, swc40, swcc37, swc43, sw25}	{sw7, swc40, swcc38, swc43, sw26, sw25}
50	{sw7, swc40, swcc38, swc43, sw25}	{sw7, swc40, swcc38, swc43, sw25}
51	{sw7, swc40, swcc37, swc43, sw25}	{sw7, sw12, swc40, swcc38, swc43, sw26, sw25}
52	{sw25, swc43, swcc37, swc40, sw7}	{sw25, sw28, swc43, swcc38, swc40, sw8, sw7}
53	{sw8, swc40, swcc38, swc43, sw26}	{sw8, swc40, swcc38, swc43, sw26}
54	{sw8, swc40, swcc37, swc43, sw26}	{sw8, swc40, swcc38, swc43, sw26}
55	{sw8, swc40, swcc37, swc43, sw26}	{sw8, swc40, swcc38, swc43, sw26}
56	{sw26, swc43, swcc37, swc40, sw8}	{sw26, swc43, swcc38, swc40, sw8}
57	{sw9, swc40, swcc37, swc43, sw27}	{sw9, swc40, swcc38, swc43, sw26, sw27}
58	{sw9, swc40, swcc37, swc43, sw27}	{sw9, swc40, swcc38, swc43, sw27}
59	{sw9, swc40, swcc37, swc43, sw27}	{sw9, sw12, swc40, swcc38, swc43, sw26, sw27}

Continuation of Table 8.7		
ID	Path Fairness	Path K-shortest
60	{sw27, swc43, swcc37, swc40, sw9}	{sw27, sw28, swc43, swcc38, swc40, sw8, sw9}
61	{sw10, swc40, swcc38, swc43, sw28}	{sw10, swc40, swcc38, swc43, sw28}
62	{sw10, swc40, swcc37, swc43, sw28}	{sw10, swc40, swcc38, swc43, sw28}
63	{sw10, swc40, swcc37, swc43, sw28}	{sw10, swc40, swcc38, swc43, sw28}
64	{sw28, swc43, swcc37, swc40, sw10}	{sw28, swc43, swcc38, swc40, sw10}
65	{sw11, swc40, swcc37, swc43, sw29}	{sw11, swc40, swcc38, swc43, sw26, sw29}
66	{sw11, swc40, swcc37, swc43, sw29}	{sw11, swc40, swcc38, swc43, sw29}
67	{sw11, swc40, swcc37, swc43, sw29}	{sw11, sw12, swc40, swcc38, swc43, sw26, sw29}
68	{sw29, swc43, swcc37, swc40, sw11}	{sw29, sw28, swc43, swcc38, swc40, sw8, sw11}
69	{sw12, swc40, swcc37, swc43, sw30}	{sw12, swc40, swcc38, swc43, sw30}
70	{sw12, swc40, swcc37, swc43, sw30}	{sw12, swc40, swcc38, swc43, sw30}
71	{sw12, swc40, swcc37, swc43, sw30}	{sw12, swc40, swcc38, swc43, sw30}
72	{sw30, swc43, swcc37, swc40, sw12}	{sw30, swc43, swcc38, swc40, sw12}
73	{sw13, swc41, swcc37, swc44, sw31}	{sw13, swc41, swcc38, swc44, sw36, sw31}
74	{sw13, swc41, swcc38, swc44, sw31}	{sw13, swc41, swcc38, swc44, sw31}
75	{sw13, swc41, swcc37, swc44, sw31}	{sw13, sw14, swc41, swcc38, swc44, sw36, sw31}
76	{sw31, swc44, swcc37, swc41, sw13}	{sw31, sw32, swc44, swcc38, swc41, sw14, sw13}
77	{sw14, swc41, swcc38, swc44, sw32}	{sw14, swc41, swcc38, swc44, sw32}
78	{sw14, swc41, swcc38, swc44, sw32}	{sw14, swc41, swcc38, swc44, sw32}
79	{sw14, swc41, swcc37, swc44, sw32}	{sw14, swc41, swcc38, swc44, sw32}
80	{sw32, swc44, swcc37, swc41, sw14}	{sw32, swc44, swcc38, swc41, sw14}
81	{sw15, swc41, swcc38, swc44, sw33}	{sw15, swc41, swcc38, swc44, sw32, sw33}
82	{sw15, swc41, swcc38, swc44, sw33}	{sw15, swc41, swcc38, swc44, sw33}
83	{sw15, swc41, swcc37, swc44, sw33}	{sw15, sw14, swc41, swcc38, swc44, sw32, sw33}
84	{sw33, swc44, swcc37, swc41, sw15}	{sw33, sw32, swc44, swcc38, swc41, sw14, sw15}
85	{sw16, swc41, swcc38, swc44, sw34}	{sw16, swc41, swcc38, swc44, sw34}
86	{sw16, swc41, swcc38, swc44, sw34}	{sw16, swc41, swcc38, swc44, sw34}
87	{sw16, swc41, swcc37, swc44, sw34}	{sw16, swc41, swcc38, swc44, sw34}
88	{sw34, swc44, swcc37, swc41, sw16}	{sw34, swc44, swcc38, swc41, sw16}
89	{sw17, swc41, swcc38, swc44, sw35}	{sw17, swc41, swcc38, swc44, sw32, sw35}
90	{sw17, swc41, swcc38, swc44, sw35}	{sw17, swc41, swcc38, swc44, sw35}
91	{sw17, swc41, swcc37, swc44, sw35}	{sw17, sw14, swc41, swcc38, swc44, sw32, sw35}

Continuation of Table 8.7		
ID	Path Fairness	Path K-shortest
92	{sw35, swc44, swcc37, swc41, sw17}	{sw35, sw32, swc44, swcc38, swc41, sw14, sw17}
93	{sw18, swc41, swcc38, swc44, sw36}	{sw18, swc41, swcc38, swc44, sw36}
94	{sw18, swc41, swcc38, swc44, sw36}	{sw18, swc41, swcc38, swc44, sw36}
95	{sw18, swc41, swcc37, swc44, sw36}	{sw18, swc41, swcc38, swc44, sw36}
96	{sw36, swc44, swcc37, swc41, sw18}	{sw36, swc44, swcc38, swc41, sw18}

The behaviour of both, the fairness heuristic, and the K-shortest path algorithm is very similar to the one documented in section 8.2.2, for the partial inter cluster scenario. The fairness algorithm chooses paths that use the wired link that connects sw with swc and utilises both the swcc37 and swcc38 switches (21 out of the 72 flows in the tables pass through swcc38). Regarding the K-shortest path of ONOS, the paths follow the same pattern detected in the previous section. Only the swcc38 is ever used, and depending on the models of the machines (Raspberry Pi or Cubiboard) and the service type (video, voice or BUD), different links are used to leave/arrive at the source/target switch (the switch connected to the source/target hosts, respectively).

One difference from the previous experiments is that in the paths of the K-shortest algorithm when multiple detours occur inside the same cluster, the same switch would be used (sw14 for flows 1, 3, 4, 9, etc or sw4 for flows 3, 4, 11, 12, etc). In these experiments, different switches are used for detour, e.g. flows 25 and 27 use sw24 and flows 33 and 35 use sw22. Since the energy cost is the same, we conclude that when multiple paths have the minimum energy cost, the K-shortest path doesn't retrieve the list in any particular order.

Additionally, Figures 8.9, 8.9 and 8.11 illustrate the discrepancy in the utilisation of the swcc redundant nodes between the forwarding solutions by reporting the number of packets sent to each one throughout the experiment.

In the two first clusters, the traffic follows similar distributions: 1/3 of the traffic uses one swcc and 2/3 passes in the other one. Keeping in mind that the majority of the packets sent are from the video service, see Table 8.8, cluster 1 reports more packets in swcc38, since 4 out of the 6 video service flows generated in this cluster pass through it, while cluster 2 reports more packets in swcc37, since also 4 out of the 6 video flows generated use it in their paths. The discrepancy in the number of packets is bigger in the third cluster since only 1 out of the 6 video flows uses swcc37.

In regards to the observed loss and jitter of the UDP services, Table 8.8 contains the calculated average values of the iPerf metrics for the generated video and voice services.

The average information sent (bytes and datagrams) and the transfer speed is similar between the two forwarding solutions, which is expected since the flows are generated using the same JSON file.

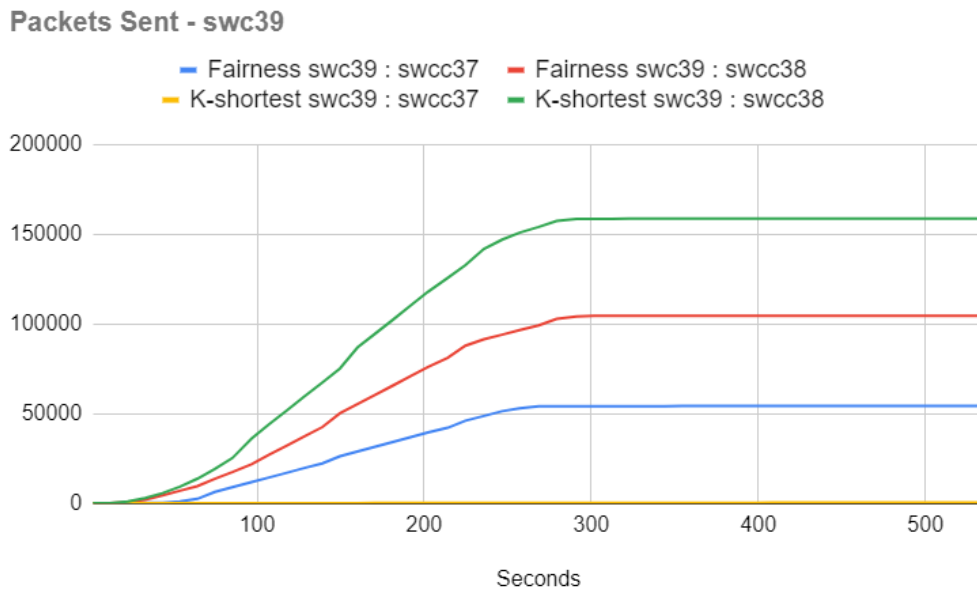


Figure 8.9: Full Inter Cluster - Packets Sent Comparison - Sent by swc39

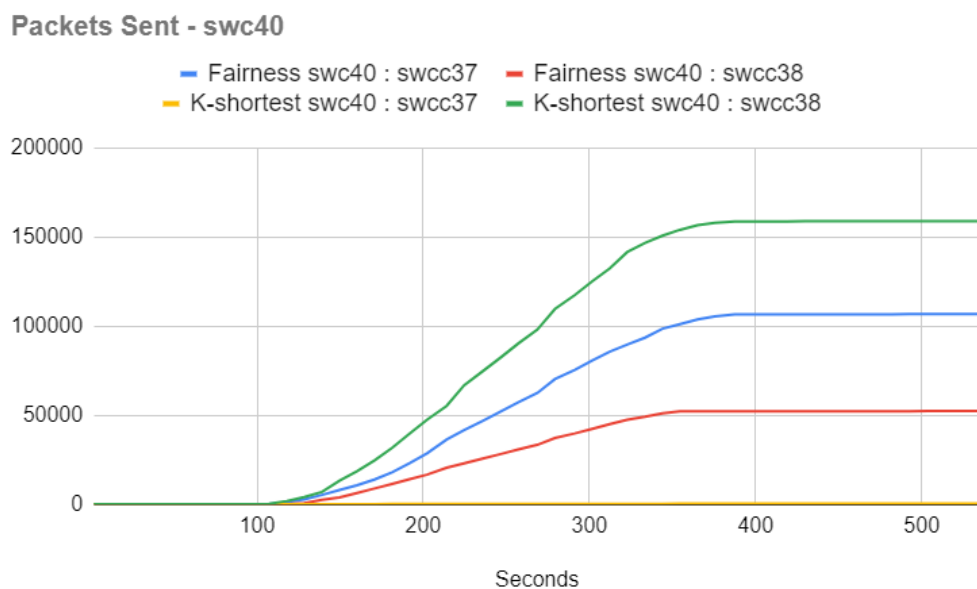


Figure 8.10: Full Inter Cluster - Packets Sent Comparison - Sent by swc40

All the average jitter values are negligible, but in the K-shortest path, the average jitter of the voice service the double of the video value. This is probably an outlier caused by ONOS: e.g. a flow entry being installed a bit later than usual.

The distributions of the loss probability are similar to the ones in Section 8.2.2: the fairness heuristic gives similar values for loss in video and loss in voice, probably because both paths use wired links; the K-shortest path reports higher loss in the video service, because it uses paths that utilise wireless links, which are more likely to lose packets.

Packets Sent - swc41

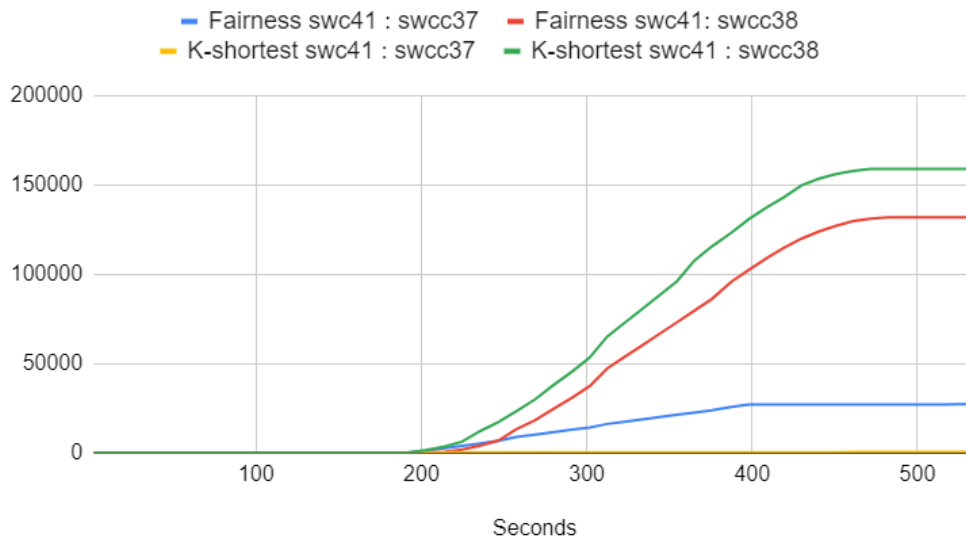


Figure 8.11: Full Inter Cluster - Packets Sent Comparison - Sent by swc41

Table 8.8: Full Inter Cluster - iPerf UDP Metrics

Service	Information Sent (KB)	Speed (Kbps)	Jitter (ms)	Lost Percentage	Datagrams Sent
Fairness Heuristic					
video	36858.3 ±24.14	1520.1 ±9.44	0.02 ±0.02	0.40% ±0.04	25767.3 ±0.46
voice	601.6 ±1.93	24.2 ±0.22	0.029 ±0.02	0.53% ±0.35	421.3 ±0.47
K-shortest ONOS					
video	36636.4 ±237.06	1513.3 ±14.29	0.022 ±0.02	0.98% ±0.6	25767.4 ±0.61
voice	603 ±1.73	24.01 ±0.21	0.041 ±0.04	0.34% ±0.25	421.6 ±0.51

8.2.4 Intra Cluster

After analysing the data produced in the previous experiences, we felt that a more extensive *intra* cluster scenario was pertinent to test how the forwarding algorithms would react to flows in this medium: links with less bandwidth capacity, links with a higher probability of losing packets, energy consumption of wireless interfaces.

In Table 8.9 we can find the iPerf traffic events of this simulation.

The service flows were placed in a manner that allows us to visualise the consequences of the arrival of consecutive service instances and analyse some machine combinations not tested before: three sequential video traffic flows from hosts in sw1 to hosts in sw3, Raspberry Pi to Raspberry Pi; three sequential voice traffic flows from hosts in sw2 to hosts in sw5, Cubiboard to Raspberry Pi; three sequen-

Table 8.9: Intra Cluster Flows

ID	Start Time	Source Target	Service
97	0	sw1 : sw3	video
98	5		
99	10		
100	15	sw2 : sw5	voice
101	20		
102	25		
103	30	sw4 : sw6	BUD
104	-	sw6 : sw4	BUDr
105	35	sw4 : sw6	BUD
106	-	sw6 : sw4	BUDr
107	40	sw4 : sw6	BUD
108	-	sw6 : sw4	BUDr

Table 8.10: Intra Cluster Paths

ID	Path Fairness	Path K-short
97	{sw1, sw3}	{sw1, sw3}
98	{sw1, sw3}	{sw1, sw3}
99	{sw1, sw3}	{sw1, sw3}
100	{sw2, sw5}	{sw2, sw5}
101	{sw2, sw5}	{sw2, sw5}
102	{sw2, swc39, sw5}	{sw2, sw5}
103	{sw4, sw6}	{sw4, sw6}
104	{sw6, sw4}	{sw6, sw4}
105	{sw4, sw6}	{sw4, sw6}
106	{sw6, sw4}	{sw6, sw4}
107	{sw4, sw6}	{sw4, sw6}
108	{sw6, sw4}	{sw6, sw4}

tial BUD traffic flows from hosts in sw4 to hosts in sw6, Cubiboard to Cubiboard.

The resulting flow paths that each forwarding solution retrieves can be found in Table 8.10.

Analysing this data, the first thing that is noticed is the distinct path of the third voice flow (flow id 102), where the fairness heuristic opted to change the traffic path from the ones used in flows 100 and 101.

Firstly, we know that the path that minimises the energy weight is the one that uses the sw2-sw5 link, because it is the one that K-shortest path uses. Additionally, voice flows have a very small datarate when compared with the other services, which means that the energy and delay weights for voice flows produce small values, recall Section 7.3.1: the energy weight for voice in sw2-sw5 is 0.0425, while for video is 0.11 and for BUD 0.847; the delay weight for voice in sw2-sw5 is 0.0017, while for video is 0.1 and for BUD 0.889;

The behaviour of the fairness heuristic can be understated when we consider that

for flows 100 and 101 the weight of the link was the value of the energy weight (since it is higher than the delay weight and the loss is probably 0). When flow 102 reaches the controller, there was observable loss in the link *sw2-sw5* caused by the previous voice flows, and thus the weight of the link is now the value of the loss weight. So the heuristic chooses to use the path that utilises *swc39* since it has 0 loss and minimal energy consumption.

This behaviour doesn't apply to the other services since the energy costs are much higher (because they have higher data), and the cost of paths with the *swc39* node always surpasses the cost of using the direct wireless mesh link.

The distribution of the packets sent by *sw2* is documented in 8.12. The data reflects the service paths documented above: the K-shortest path sends all the packets directly to *sw6*, while the heuristic was able to distribute the load throughout the resources.

Packets Sent - sw2

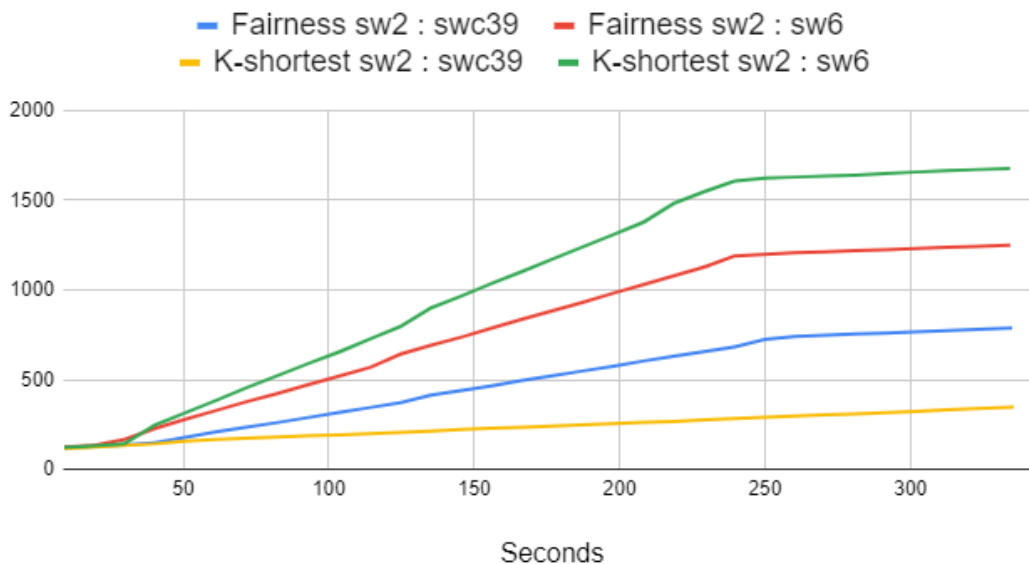


Figure 8.12: Intra Cluster - Packets Sent Comparison

Notice that some packets were recorded leaving *sw2* to *swc39* with the K-shortest path algorithm. These aren't datagrams of service traffic, but are instead passive packets that ONOS subsystems generate necessary for the operation of the Software Defined Network Controller (SDN-C).

Table 8.11 records the average values of the iPerf metrics for the generated video and voice services. For the most part, the table contains identical values to the ones of previous experiments. Some standard deviations are equal to zero since this experience has fewer iPerf events than the others.

The novelty is that now services are reporting more losses, due to the fact that only wireless links are being used. The exception is the path of flow 102, which uses wired links, and that contributes to lowering the average value of losses in the service of voice with the fairness heuristic.

Table 8.11: Intra Cluster - iPerf UDP Metrics

Service	Information Sent (KB)	Speed (Kbps)	Jitter (ms)	Lost Percentage	Datagrams Sent
Fairness Heuristic					
video	36556.8 ±0.0	1495.04 ±17.762	0.010 ±0.004	1.2% ±0.057	25767.7 ±0.577
voice	600.7 ±2.08	24.1 ±0.26	0.017 ±0.003	0.6% ±0.36	421 ±0.0
K-shortest ONOS					
video	36556.8 ±0.0	1515.5 ±0	0.02 ±0.0263	1.2% ±0	25767 ±0.0
voice	593.3 ±8.326	23.97 ±0.321	0.016 ±0.0201	1.8% ±1.34	421 ±0.0

8.3 Discussion

With the overall results of the experiments, it can be concluded that the framework performed its tasks properly.

The framework was successfully deployed in the ONOS controller and handled the characteristics of an emulated scenario of a real smart-city environment: different machines with heterogeneous computing capabilities, distinct links properties and traffic of prevalent services. It was able to react upon the detection of new service flows, and handle their respective packets, accordingly to the specified configurations of the user: dropping packets of unregistered services and forwarding flows of expected services. The necessary rules were installed in topology devices to enforce traffic paths. The snapshots of the topology were successfully stored in the database enabling users to visualise the evolution of topology behaviour, while the configuration parameters could be edited to allow managing of forwarding solutions and configuration of authorised traffic.

Regarding the algorithms, the heuristic provided service paths with fairness concerns enabling the distribution of the load and resource utilisation, while the K-shortest path implemented the paths that minimised energy cost: the framework gives forwarding solutions with heterogeneous capabilities, which users can choose according to their needs.

Discussing the fairness aspect of the heuristic, for simple *intra* cluster scenarios, we found that the heuristic achieves the same objective function value as the formulation, which is the best possible scenario. For more complex scenarios, where running the formulation is more cumbersome, we opted to analyse this aspect by performing experiences. We found that instances of a traffic flow, with the same service type and the same source and target switches, were taking different paths: in the *inter* cluster experiences, recall Sections 8.2.2 and 8.2.3, both *swcc37* and *swcc38* nodes were utilised. For the *intra* cluster experiment, see Section 8.2.4, some paths used the wireless link between switches, while others passed through the cluster switch (*swc*).

Finally, the results showed the potential gain of using our fairness proposal, as a feasible forwarding solution, over typical out-of-the-shelf mechanisms that come with the SDN-C. There are circumstances where the presented K-shortest path approach is preferable (e.g. devices need to save battery), but to manage a real topology it is flawed.

Chapter 9

Feature Testing & Validation

In this Chapter, we document the steps involved in the validation of the correctness of requirements and framework routines.

Keep in mind, that this is not an exhaustive list of testing of all the components of the framework. The aim is to illustrate a formal testing approach and cover the routines we considered more critical: the ones directly related to requirements (e.g. utilise end-points, save snapshots); validating the behaviour of external works (e.g. JGraphT library).

We will be taking advantage of the **BlackBox Testing** approach. This is a technique in which the functionalities can be tested without having access or knowledge of the internal code structure. To perform this testing approach it is only necessary to have the requirements of a routine and to know the expected output for a given combination of input parameters [Khan and Khan, 2012].

In this *testing suite*, we want to reach coverage of the *bounding box* valid input parameters. This means that for each one of the routines, the tests will use combinations of the minimum and maximum valid value for each input parameter.

9.1 Web Server End-points

This section documents tests done against the end-points that the Web server uses to expose functionality for users and external tools. To help build these Representational State Transfer (REST) requests, the tool Postman¹ was used.

Table 9.1 documents the tests done to retrieve information from the database. The REST verb GET is used in the request and depending on the URI of the end-points used the information of a different table is retrieved. The database had information of multiple snapshots stored when we run the tests:

- *1 Returns code 200; Retrieves, in JSON format, all the entries of the database table that the URI identifies.

¹<https://www.postman.com/product/tools/>

Table 9.1: BlackBox Tests - Retrieving Information with End-points

ID	Input		Expected Outcome	Pass/ Fail
	Verb	End - Point		
1	GET	api/devices	*1	PASS
2	GET	api/ports	*1	PASS
3	GET	api/links	*1	PASS
4	GET	api/flowrules	*1	PASS
5	GET	api/bytestatistics	*1	PASS
6	GET	api/packetstatistics	*1	PASS
7	GET	api/hosts	*1	PASS
8	GET	api/interfaces	*1	PASS
9	GET	api/locations	*1	PASS
10	GET	api/energy	*1	PASS
11	GET	api/machines	*1	PASS
12	GET	api/devicemachinerelations	*1	PASS
13	GET	api/generalconfigurations	*1	PASS
14	GET	api/serviceconfigurations	*1	PASS
15	GET	api/servicehostrelations	*1	PASS

All fifteen tests of Table 9.1 retrieved the expected output.

Table 9.2 documents the performed tests to retrieve and filter information from the database. The REST verb GET is used in the request and depending on the URI of the end-points, and the values of the two input parameters, the information of a different table is retrieved. The *uri* parameter identifies a switch in the database and the *port* is the identifier of one of the ports of the selected device. The database had information from multiple snapshots stored when we run the tests:

- *2 Returns code 200; Retrieves, in JSON format, all the entries of the database table *device* where the *uri* is the identifier of the switch.
- *3 Returns code 200; Retrieves, in JSON format, all the entries of the database table *port* where the *uri* is the identifier of the switch.
- *4 Returns code 200; Retrieves, in JSON format, all the entries of the database table *port* where the *uri* is the identifier of the switch and *port* is the port number of the device.
- *5 Returns code 200; Retrieves, in JSON format, all the entries of the database table *byte_statistics* where the *uri* is the identifier of the switch and *port* is the port number of the device.
- *6 Returns code 200; Retrieves, in JSON format, all the entries of the database table *port_statistics* where the *uri* is the identifier of the switch and *port* is the port number of the device.

All the tests in Table 9.2 passed successfully.

Table 9.2: BlackBox Tests - Retrieving Filtered Information with End-points

ID	Input				Expected Outcome	Pass/Fail
	Verb	End - Point	uri	port		
16	GET	api/devices/{uri}	of:0000000000000001	-	*2	PASS
17	GET	api/devices/{uri}	of:0000000000000009	-	*2	PASS
18	GET	api/devices/{uri}	of:000000000000000a	-	*2	PASS
19	GET	api/devices/{uri}	of:0000000000000002c	-	*2	PASS
20	GET	api/devices/{uri}/ports	of:0000000000000001	-	*3	PASS
21	GET	api/devices/{uri}/ports	of:0000000000000009	-	*3	PASS
22	GET	api/devices/{uri}/ports	of:000000000000000a	-	*3	PASS
23	GET	api/devices/{uri}/ports	of:0000000000000002c	-	*3	PASS
24	GET	api/devices/{uri}/ports/{port}	of:0000000000000001	1	*4	PASS
25	GET	api/devices/{uri}/ports/{port}	of:0000000000000001	9	*4	PASS
26	GET	api/devices/{uri}/ports/{port}	of:0000000000000002c	1	*4	PASS
27	GET	api/devices/{uri}/ports/{port}	of:0000000000000002c	8	*4	PASS
28	GET	api/devices/{uri}/ports/{port}/bytestatistics	of:0000000000000001	1	*5	PASS
29	GET	api/devices/{uri}/ports/{port}/bytestatistics	of:0000000000000001	9	*5	PASS
30	GET	api/devices/{uri}/ports/{port}/bytestatistics	of:0000000000000002c	1	*5	PASS
31	GET	api/devices/{uri}/ports/{port}/bytestatistics	of:0000000000000002c	8	*5	PASS
32	GET	api/devices/{uri}/ports/{port}/packetstatistics	of:0000000000000001	1	*6	PASS
33	GET	api/devices/{uri}/ports/{port}/packetstatistics	of:0000000000000001	9	*6	PASS
34	GET	api/devices/{uri}/ports/{port}/packetstatistics	of:0000000000000002c	1	*6	PASS
35	GET	api/devices/{uri}/ports/{port}/packetstatistics	of:0000000000000002c	8	*6	PASS

The last functionalities exposed by the web server are related to data manipulation, see Table 9.3. The REST verb POST is used to add entries to the tables in the database, while the PUT is used to update the information of existing entries in the tables. To update/insert an entry in the database, a JSON must be sent in the request body, as exemplified in the following list. The tests are documents in the way that they are executed, due to some dependencies of the database schema, and foreign key constraints:

Table 9.3: BlackBox Tests - Insert and Update Information with End-points

ID	Input			Expected Outcome	Pass/Fail
	Verb	End - Point	Body JSON		
36	POST	api/generalconfigurations	**JSON1	*7	PASS
37	POST	api/generalconfigurations	**JSON1	*8	PASS
38	PUT	api/generalconfigurations	**JSON2	*9	PASS
39	POST	api/machine	**JSON3	*7	PASS
40	POST	api/machine	**JSON4	*7	PASS
41	POST	api/energy	**JSON5	*7	PASS
42	POST	api/energy	**JSON5	*8	PASS
43	PUT	api/energy	**JSON6	*9	PASS
44	POST	api/devicemachinerelation	**JSON7	*7	PASS
45	POST	api/devicemachinerelation	**JSON7	*8	PASS
46	PUT	api/devicemachinerelation	**JSON8	*9	PASS
47	POST	api/serviceconfiguration	**JSON9	*7	PASS
48	POST	api/serviceconfiguration	**JSON9	*8	PASS
49	POST	api/serviceconfiguration	**JSON10	*7	PASS
50	PUT	api/serviceconfiguration	**JSON11	*9	PASS
51	POST	api/servicehostrelation	**JSON12	*7	PASS
52	POST	api/servicehostrelation	**JSON12	*8	PASS
53	PUT	api/servicehostrelation	**JSON13	*9	PASS

Possible JSONs in the Request Body

```

**JSON1
{"pkey": 1, "wired_bw_mbs": 1000000, "wireless_bw_mbs": 1000000,
  "flowrule_priority": 1000000, "flowrule_timeout":1000000,
  "flowrule_appid":1000000, "default_fw_algorithm": "3-obj-fairness"}

**JSON2
{"pkey": 1, "wired_bw_mbs": 1, "wireless_bw_mbs": 1,
  "flowrule_priority": 1, "flowrule_timeout":1,
  "flowrule_appid":1, "default_fw_algorithm": "3-obj-fairness"}

**JSON3
{"name": "PI"}

**JSON4
{"name": "Cubi"}

**JSON5
{ "name": "energy_wireless_u", "energy_value": 0.0, "datarate": 0.0,

```



```
"cof_degree_0": -0.00000001, "cof_degree_1": -0.00000001,
"cof_degree_2": -0.00000001, "machine_name": "PI" }

**JSON6
{ "name": "energy_wireless_u", "energy_value": 0.0, "datarate": 1000000,
  "cof_degree_0": 1000000, "cof_degree_1": 1000000,
  "cof_degree_2": 1000000, "machine_name": "PI" }

**JSON7
{"device_uri": "of:0000000000000001", "machine_name": "Cubi" }

**JSON8
{"device_uri": "of:0000000000000001", "machine_name": "PI" }

**JSON9
{"name": "video", "datarate_mbs": 0.00000001}

**JSON10
{"name": "voice", "datarate_mbs": 1000000}

**JSON11
{"name": "voice", "datarate_mbs": 0.00000001}

**JSON12
{"mac_src": "AA:BB:CC:DD:00:01", "mac_dst": "AA:BB:CC:DD:00:37",
  "service_conf_name": "video"}

**JSON13
{"mac_src": "AA:BB:CC:DD:00:01", "mac_dst": "AA:BB:CC:DD:00:37",
  "service_conf_name": "voice"}
```

- *7 Returns code 201; Inserts a table entry, with the information of the JSON, in the respective table that the end-point Uniform Resource Identifier (URI) identifies.
- *8 Returns code 500; Internal server error because of duplicate primary Keys in the database.
- *9 Returns code 201; Updates the information of a table entry that the end-point URI and the parameters of the JSON identify.

All the tests of Table 9.3 retrieved the expected output.

These tests helped validate the non functional requirement of the framework related to interoperability. External users can execute framework functionalities by performing REST requests.

9.2 CLI Commands

The framework contains a range of commands for users to monitor the framework and configure the behaviour of the topology, recall Chapter 7. This section documents the tests done to validate the correctness of these functionalities accessible via Command-Line interface (CLI).

Table 9.4 consists of the tests done to retrieve information from the database. Using the flag “-read” users can retrieve information of table arg1 from the database. The database had information from multiple snapshots stored when we run the tests:

Table 9.4: BlackBox Tests - Retrieving Information with CLI

ID	Input		Expected Outcome	Pass/Fail
	flag	arg1		
54	-read	device	*10	PASS
55	-read	port	*10	PASS
56	-read	link	*10	PASS
57	-read	flowrule	*10	PASS
58	-read	byte_statistics	*10	PASS
59	-read	packet_statistics	*10	PASS
60	-read	host	*10	PASS
61	-read	interface	*10	PASS
62	-read	location	*10	PASS
63	-read	energy	*10	PASS
64	-read	machine	*10	PASS
65	-read	device_machine_relation	*10	PASS
66	-read	general_conf	*10	PASS
67	-read	service_conf	*10	PASS
68	-read	service_host_relation	*10	PASS

- ***10** Retrieves, and prints in the shell, all the entries of the database table that arg1 identifies.

All the tests documented in Table 9.4 performed as expected.

Additionally, tests for the commands related to updating the database and populating its tables are documented in Table 9.5. For the flag -loop, the arg1 represents the seconds between snapshot storage in the database, while for the -insert and -update flags, it identifies the table to manipulate. The record in the table to insert/update is identified by the information of the list of arg2.

Table 9.5: BlackBox Tests - CLI for Management

ID	Input			Expected Outcome	Pass/ Fail
	flag	arg1	arg2		
69	-reset	-	-	*11	PASS
70	-init	-	-	*12	PASS
71	-create	-	-	*13	PASS
72	-loop	5	-	*14	PASS
73	-loop	10	-	*14	PASS
74	-stoploop	-	-	*15	PASS
75	-snap	-	-	*16	PASS
76	-insert	general_conf	**List1	*17	PASS
77	-insert	general_conf	**List1	*18	PASS
78	-update	general_conf	**List2	*19	PASS
79	-insert	machine	**List3	*17	PASS
80	-insert	machine	**List4	*17	PASS
81	-insert	energy	**List5	*17	PASS
82	-insert	energy	**List5	*18	PASS
83	-update	energy	**List6	*19	PASS
84	-insert	device_machine_relation	**List7	*17	PASS
85	-insert	device_machine_relation	**List7	*18	PASS
86	-update	device_machine_relation	**List8	*19	PASS
87	-insert	service_conf	**List9	*17	PASS
88	-insert	service_conf	**List9	*18	PASS
89	-insert	service_conf	**List10	*17	PASS
90	-update	service_conf	**List11	*19	PASS
91	-insert	service_host_relation	**List12	*17	PASS
92	-insert	service_host_relation	**List12	*18	PASS
93	-update	service_host_relation	**List13	*19	PASS

List of input values for Table 9.5

```
**List1
[ 1, 1000000, 1000000, 1000000, 1000000, 1000000, 3-obj-fairness]

**List2
[ 1, 1, 1, 1, 1, 1, 3-obj-fairness]

**List3
[ PI ]

**List4
[ Cubi ]

**List5
[ energy_wireless_u, 0, 0, -0.00000001, -0.00000001, -0.00000001, PI ]

**List6
[ energy_wireless_u, 0, 1000000, 1000000, 1000000, 1000000, PI ]

**List7
[ of:000000000000000001, Cubi ]

**List8
[ of:000000000000000001, PI ]

**List9
[ video, 0.00000001 ]

**List10
[ voice, 1000000 ]

**List11
[ voice, 0.00000001 ]

**List12
[ AA:BB:CC:DD:00:01, AA:BB:CC:DD:00:37, video]

**List13
[ AA:BB:CC:DD:00:01, AA:BB:CC:DD:00:37, voice]
```

- ***11** Resets the database schema by deleting all the information of previous tables and then creating the same database structure.
- ***12** The tables `general_conf`, `machine`, `energy`, `device_machine_relation`, `service_conf`, `service_conf` are initialised with predefined values.
- ***13** Creates the database tables.
- ***14** The framework should take a snapshot each `arg1` seconds;
- ***15** If the framework was taking snapshots of the topology, this command should stop this looping action.
- ***16** Takes a single snapshot of the topology.

- *17 A new record is added to the database table `arg1`; The entry values are the one of `arg2`.
- *18 An internal server error should be returned, warning the user that a record with the same primary key exists already in the database.
- *19 A record is updated in the database table `arg1`; The entry values are the one of `arg2`.

These tests performed helped validate the functional requirement of the framework related to monitoring and management activities. Users can record topology information and adjust the configurations of the framework to change the behaviour of the topology.

9.3 Energy Formulas

Our fairness algorithm calculates the weights of links using the objectives of energy, loss probability and delay. Regarding the energy objective, we used the formulas that [Kaup et al., 2018] proposed, recall Section 8.1.4. Due to the complexity of the equations and the energy cost calculations, we devise some tests, see Table 9.6, to confirm the correctness of our implementation.

- *20 For each of the tests, it is expected that the value of the normalised energy objective is between $]0.0, 1.0[$.

The results were unexpected, tests 106, 107 and 108 failed. For Cubiboard to Cubiboard wired connections, the value of the energy consumption for the BUD service returns a negative cost, and for the video and voice services, it returns a cost bigger than the upper bound value used for normalisation, recall Algorithm 1 of Section 7.3.1.

To resolve this problem, we searched for alternatives for the energy formulas, and also tried to detect a possible error in our implementation when adapting it from the original work. Because we didn't detect any implementation issues in our work, the other related works considered weren't as suitable, and the problematic combination wasn't present in the experiences to be performed, the final decision was to use the same formulas.

Furthermore, to perform the experiences, the only requirement was to have machine heterogeneity. The exact formulas of energy consumption are not of extreme significance.

Finally, we recommend that the reader should look at the energy formulas presented, and the exact values of energy consumption documented with rational and critical analysis.

Table 9.6: BlackBox Tests - Normalised Objective Values

ID	Input			Expected Outcome	Pass/ Fail	
	source	target	link			
94	PI	PI	wired	video	*20	PASS
95				voice	*20	PASS
96				BUD	*20	PASS
97	PI	PI	wireless	video	*20	PASS
98				voice	*20	PASS
99				BUD	*20	PASS
100	PI	Cubi	wired	video	*20	PASS
101				voice	*20	PASS
102				BUD	*20	PASS
103	PI	Cubi	wireless	video	*20	PASS
104				voice	*20	PASS
105				BUD	*20	PASS
106	Cubi	Cubi	wired	video	*20	FAIL
107				voice	*20	FAIL
108				BUD	*20	FAIL
109	Cubi	Cubi	wireless	video	*20	PASS
110				voice	*20	PASS
111				BUD	*20	PASS
112	Cubi	PI	wired	video	*20	PASS
113				voice	*20	PASS
114				BUD	*20	PASS
115	Cubi	PI	wireless	video	*20	PASS
116				voice	*20	PASS
117				BUD	*20	PASS

9.4 Algorithms

This section documents the tests done to validate routines related to the forwarding algorithms, and the packet processor, recall Section 7.3.

One of the first steps was to validate the behaviour of the *CapacityScalingMinimumCostFlow()* method of the JGraphT library, recall Section 7.3.1.

We needed to guarantee that it behaved as intended: when a service needs to be forward, the algorithm will return the path that minimises the links cost, and where each link has enough bandwidth to transport the full service.

Figure 9.1 represents the scenario of the test performed to validate the behaviour: there is a service, with size 16, that needs to go from node A to node C. The path that minimises the cost is A-D-C, with cost of 20, but the links in this path don't have enough capacity to transport the service. Thus, it is expected that the path A-B-C, with cost of 100 is used instead.

After running the algorithm, we found that the previous test **FAILED**: the service

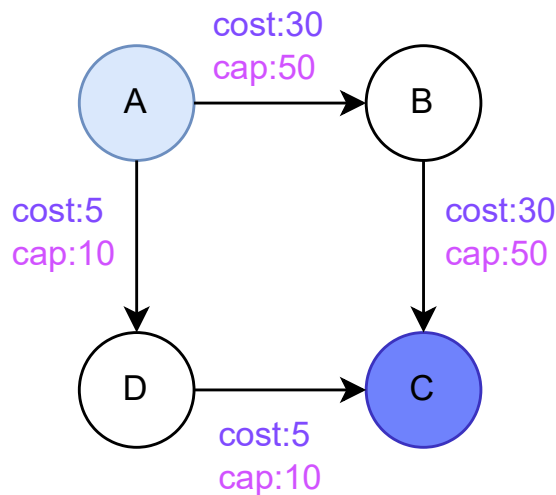


Figure 9.1: Structure of the JGraphT Test

was being divided to fit in both paths; 10 units of the size of the service were using the path A-D-C, which minimises cost, and the remaining 6 units of service size were using the path A-B-C.

To handle this issue, we proposed some modifications that might enforce the desired behaviour: the size of the service is now 1 unit; the capacity of each link is now 1 when the link has enough bandwidth to transport the whole service, and 0 otherwise. The caveat is that now we have to perform some additional calculations before we set the capacity of each link, and store the services that each link is transporting in a given moment.

Due to the adjustments done in the algorithm, it now requires validation to en-

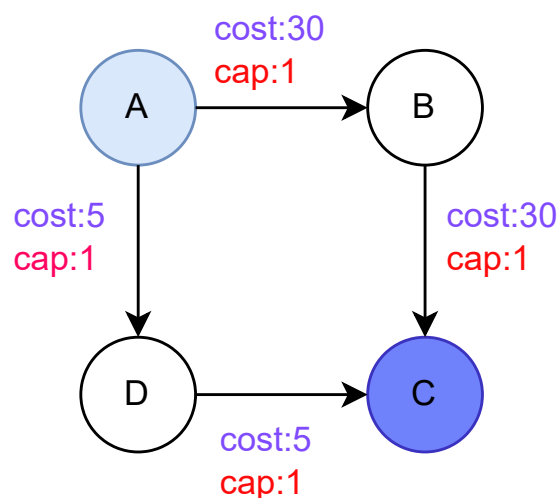


Figure 9.2: Structure of the JGraphT Modified Test

sure that it performs as expected, see Figure 9.2. The service needs to go from node A to node c, it now has size of 1 and the capacity of all links is 1. The expected outcome is for the A-D-C to be selected.

After running the algorithm, we found that the previous test **PASSED**. The JGraphT method now behaves as expected.

The next logical step is to test the implementation of the routine that performs these additional steps. It should store the available bandwidth of every link in the topology, decrease this value when a new service uses the link, and increase it when a service stops using the link.

Table 9.7 contains the tests performed to validate the routine. In test 118, three services, video, voice and BUD are generated in hosts connected in sw1 and need to reach sw2. In test 119, three services, video, voice and BUD are generated in hosts connected in sw1 and need to reach sw13.

Table 9.7: BlackBox Tests - Reserve and Replenish Bandwidth

ID	Input			Expected Outcome	Pass/ Fail
	source	target	service		
118	host1	host4	video	*21	PASS
	host2	host5	voice		
	host3	host6	BUD		
119	host1	host37	video	*21	PASS
	host2	host38	voice		
	host3	host39	BUD		

- ***21** We expect that the available bandwidth of the links that constitute the video path decreased by 1.5Mbits, the links of the voice path decreased by 0.0244Mbits and the link of the BUD path decreased by 12.88Mbits; After the end of the service events, it is expected that the bandwidth that was reserved is replenished.

The tests obtained the expected results, replenishing the correct amount of bandwidth in links after the service ended.

Chapter 10

Conclusion

With this work, we showed the current importance and necessity of network management. Recent and upcoming network services give emphasis to this need, demanding more rigorous requirements and expecting more available bandwidth with lower latency. The SDN paradigm is a powerful ally of network administrators providing tools to more flexibly manage the behaviour of network devices.

We provide a management framework for the popular ONOS SDN-C, that maintains a collection of distinct forwarding algorithms in an effort to streamline the activities of the forwarding life cycle. After the user selects the more appropriate forwarding solution to use, the framework handles the traffic detection, path selection and rule instalment automatically. Our platform has already been used by research partners to accelerate their endeavours.

The results of the experiences conducted confirmed the usefulness and suitability of our work for real scenarios. We performed tests in an ambient that resembles a smart-city with regrades to device count and characteristics, traffic services and wired and wireless connection medium. This ambient is also pertinent to show the advantages of our fairness proposal when compared with out-of-the-shelf options provided by SDN controllers.

Our proposed mechanism aims to maximise fairness of energy consumption, delay, and observable loss. The results obtained show that this approach obtains traffic flows with higher fairness than the K-shortest path algorithm existing in the ONOS framework.

The framework was able to monitor network resources and adjust the solution path accordingly. Our proposal was innovative since it is not common to see, in the literature, multiple objectives with a min-max technique.

There is definitely a place in the state-of-the-art for more straightforward approaches like the K-shortest path algorithm that we adapted from ONOS. However, to use these “simple-minded” mechanisms to manage the traffic in such complex and dynamic environments is not a good fit, since our experiments found deterioration of the quality of services in the scenarios that resort to using them.

This paradigm of supplying network administrators with multiple distinct forwarding approaches is very powerful and flexible since they can effortlessly choose the one more appropriate to use for each occasion.

10.1 Future work

This dissertation provides the possibility of several future work directions. The most relevant are highlighted. One of the first approaches would be to deploy the framework in a scenario with a wireless medium. The issues with Mininet-Wifi prevented the current work from doing so, and we ended up simulating this environment. This could provide more accurate confirmations of the suitability of the framework for this medium. Deploying the framework on topologies with different characteristics (e.g. devices, services, connections) would also increase the reputation of the usefulness of our work.

Furthermore, a future release could include the distinction between different user roles. Introducing authentication could help differentiate between users that can only monitor the network assets and users that manipulate the behaviour of the topology. Also adding to the value of this work, more forwarding solutions could be added to the algorithm collection. Including other forwarding solutions, that specialise in resolving prevalent network problems (e.g. minimise delay), means that administrators have more flexibility over the behaviour of the topology.

In the current version of the framework, some features cannot be accessed through the REST end-points. Due to project delays, we did not have enough budget to research a way to integrate the RPC technology in the ONOS controller, necessary to expose these features. In the near future, we will need to perform this research for the SNOB-5G project. The acquired knowledge could be used to expose the missing features in a future version of the framework.

Finally, the proposed ONOS management framework could be transformed into the target tool that we initially proposed (see Appendix D): a modular framework and SDN-C agnostic that could be deployed independently of the underlying technologies.

References

- Thomas Vachuska. Overview of ONOS architecture. <https://wiki.onosproject.org/display/ONOS/Overview+of+ONOS+architecture>, 2015. Last visited 2021-08-16.
- David Perez Abreu, Karima Velasquez, Luís Paquete, Marilia Curado, and Edmundo Monteiro. Resilient service chains through smart replication. *IEEE Access*, 8:187021–187036, 2020. doi: 10.1109/ACCESS.2020.3030537.
- Admin. Introducing 5G mmWave. <https://www.5gmmwave.com/5g-mmwave/introducing-5g-mmwave/>, 2020. Last visited 2021-12-27.
- Aerohive Networks. Link Budget Calculations and Choosing the Correct Antenna. https://docs.aerohive.com/330000/docs/guides/Aerohive_LinkBudgetCalculations.pdf, 2014. Last visited 2021-12-20.
- Basem Almadani, Abdurrahman Beg, and Ashraf Mahmoud. Dsf: A distributed sdn control plane framework for the east/west interface. *IEEE Access*, 9:26735–26754, 2021. doi: 10.1109/ACCESS.2021.3057690.
- Andrea Campanella. Southbound protocols. <https://wiki.onosproject.org/display/ONOS/Southbound+protocols>, 2016. Last visited 2021-08-18.
- APS Networks. BF2556X-1T - Advanced Programmable Switch. <https://www.edge-core.com/productsInfo.php?cls=1&cls2=5&cls3=181&id=335>, 2022. Last visited 2022-05-006.
- Arun K. Arahunashi, S. Neethu, and H. V. Ravish Aradhya. Performance analysis of various sdn controllers in mininet emulator. In *2019 4th International Conference on Recent Trends on Electronics, Information, Communication Technology (RTEICT)*, pages 752–756, 2019. doi: 10.1109/RTEICT46194.2019.9016693.
- Safa Ben Atitallah, Maha Driss, Wadii Boulila, and Henda Ben Ghézala. Leveraging deep learning and iot big data analytics to support the smart cities development: Review and future directions. *Computer Science Review*, 38:100303, 2020. ISSN 1574-0137. doi: <https://doi.org/10.1016/j.cosrev.2020.100303>. URL <https://www.sciencedirect.com/science/article/pii/S1574013720304032>.
- Aveiro Tech City Living Lab Team. Aveiro Tech City Living Lab Map. <https://www.google.com/maps/d/viewer?mid=1p7QSVjJk15n6IiXf8c5DY6Lr9w6aaZb5>, 2021. Last visited 2022-08-15.

aveirotechcity. Aveiro Tech City Living Lab - Caracterização Técnica. <https://www.aveirotechcity.pt/application/files/5716/0554/0015/AnexoII.PDF>, 2021. Last visited 2022-01-22.

Riccardo Bassoli, Hugo Marques, Jonathan Rodriguez, Kenneth W. Shum, and Rahim Tafazolli. Network coding theory: A survey. *IEEE Communications Surveys Tutorials*, 15(4):1950–1978, 2013. doi: 10.1109/SURV.2013.013013.00104.

Fouad Benamrane, Mouad Ben mamoun, and Redouane Benaini. An east-west interface for distributed sdn control plane: Implementation and evaluation. *Computers & Electrical Engineering*, 57:162–175, 2017. ISSN 0045-7906. doi: <https://doi.org/10.1016/j.compeleceng.2016.09.012>. URL <https://www.sciencedirect.com/science/article/pii/S0045790616302798>.

Kamal Benzekki, Abdeslam El Fergougui, and Abdelbaki Elbelrhiti Elalaoui. Software-defined networking (sdn): a survey. *Security and communication networks*, 9(18):5803–5833, 2016.

Deval Bhamare, Raj Jain, Mohammed Samaka, and Aiman Erbad. A survey on service function chaining. *Journal of Network and Computer Applications*, 75:138–155, 2016. ISSN 1084-8045. doi: <https://doi.org/10.1016/j.jnca.2016.09.001>. URL <https://www.sciencedirect.com/science/article/pii/S1084804516301989>.

Bill Snow. ONOS EMU Release. <https://wiki.onosproject.org/display/ONOS/Release+Notes?preview=%2F2129999%2F10159028%2FEmu+Release+Summary+%281%29.pptx>, 2015. Last visited 2021-08-16.

Nikos Bizanis and Fernando A. Kuipers. Sdn and virtualization solutions for the internet of things: A survey. *IEEE Access*, 4:5591–5606, 2016. doi: 10.1109/ACCESS.2016.2607786.

Cisco. Cisco Annual Internet Report (2018–2023). <https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.pdf>, 2020. Last visited 2021-12-27.

Cisco. Cisco Catalyst 9130AX Series Access Points Data Sheet. <https://www.cisco.com/c/en/us/products/collateral/wireless/catalyst-9100ax-access-points/nb-06-cat-9130-ser-ap-ds-cte-en.html>, 2021. Last visited 2022-01-22.

Alejandro Cohen, Homa Esfahanizadeh, Bruno Sousa, João P. Vilela, Miguel Luís, Duarte Raposo, François Michel, Susana Sargento, and Muriel Médard. Bringing network coding into SDN: A case-study for highly meshed heterogeneous communications. *CoRR*, abs/2010.00343, 2020. URL <https://arxiv.org/abs/2010.00343>.

Computer Hope Team. Computer terms, dictionary, and glossary. <https://www.computerhope.com/jargon.htm>, 2022. Last visited 2022-01-09.

- Christoph Dietzel, Gianni Antichi, Ignacio Castro, Eder L. Fernandes, Marco Chiesa, and Daniel Kopp. Sdn-enabled traffic engineering and advanced black-holing at ixps. In *Proceedings of the Symposium on SDN Research, SOSR '17*, page 181–182, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450349475. doi: 10.1145/3050220.3060601. URL <https://doi.org/10.1145/3050220.3060601>.
- Jianbo Du, Liqiang Zhao, Jie Feng, and Xiaoli Chu. Computation offloading and resource allocation in mixed fog/cloud computing systems with min-max fairness guarantee. *IEEE Transactions on Communications*, 66(4):1594–1608, 2018. doi: 10.1109/TCOMM.2017.2787700.
- Edge-core Networks. DCS800 - 6.4T PROGRAMMABLE DATA CENTER SWITCH. <https://www.aps-networks.com/products/bf2556x-1t/>, 2022. Last visited 2022-05-006.
- Farzaneh Pakzad from Aptira. COMPARISON OF SOFTWARE DEFINED NETWORKING (SDN) CONTROLLERS. <https://ryu-sdn.org/>, 2021. Last visited 2022-01-17.
- Faucet Organisation. What's Ryu. <https://github.com/faucetsdn/ryu>, 2022. Last visited 2022-01-17.
- Lyndon Fawcett, Sandra Scott-Hayward, Matthew Broadbent, Andrew Wright, and Nicholas Race. Tension: A distributed sdn framework for scalable network security. *IEEE Journal on Selected Areas in Communications*, 36(12):2805–2818, 2018. doi: 10.1109/JSAC.2018.2871313.
- Abderrahime Filali, Amine Abouaomar, Soumaya Cherkaoui, Abdellatif Kobbane, and Mohsen Guizani. Multi-access edge computing: A survey. *IEEE Access*, 8:197017–197046, 2020. doi: 10.1109/ACCESS.2020.3034136.
- Ramon R. Fontes, Samira Afzal, Samuel H. B. Brito, Mateus A. S. Santos, and Christian Esteve Rothenberg. Mininet-wifi: Emulating software-defined wireless networks. In *2015 11th International Conference on Network and Service Management (CNSM)*, pages 384–389, 2015. doi: 10.1109/CNSM.2015.7367387.
- Paula Fraga-Lamas, Mikel Celaya-Echarri, Leyre Azpilicueta, Peio Lopez-Iturri, Francisco Falcone, and Tiago M. Fernández-Caramés. Design and empirical validation of a lorawan iot smart irrigation system. *Proceedings*, 42(1), 2020. ISSN 2504-3900. doi: 10.3390/ecsa-6-06540. URL <https://www.mdpi.com/2504-3900/42/1/62>.
- Ali Ghaffari. Congestion control mechanisms in wireless sensor networks: A survey. *Journal of Network and Computer Applications*, 52:101–115, 2015. ISSN 1084-8045. doi: <https://doi.org/10.1016/j.jnca.2015.03.002>. URL <https://www.sciencedirect.com/science/article/pii/S1084804515000557>.
- Fuad A. Ghaleb, Bander Ali Saleh Al-Rimy, Wadii Boulila, Faisal Saeed, Maznah Kamat, Mohd Foad Rohani, and Shukor Abd Razak. Fairness-Oriented Semichaotic Genetic Algorithm-Based Channel Assignment Technique for

- Node Starvation Problem in Wireless Mesh Networks. *Computational Intelligence and Neuroscience*, 2021:1–19, aug 2021. ISSN 1687-5273.
- Syed Sherjeel A. Gilani, Amir Qayyum, Rao Naveed Bin Rais, and Mukhtiar Bano. Sdnmesh: An sdn based routing architecture for wireless mesh networks. *IEEE Access*, 8:136769–136781, 2020. doi: 10.1109/ACCESS.2020.3011651.
- Noé Godinho, Henrique Silva, Marilia Curado, and Luís Paquete. A reconfigurable resource management framework for fog environments. *Future Generation Computer Systems*, 133:124–140, 2022. ISSN 0167-739X. doi: <https://doi.org/10.1016/j.future.2022.03.015>. URL <https://www.sciencedirect.com/science/article/pii/S0167739X22000905>.
- P. Goransson, C. Black, and T. Culver. *Software Defined Networks: A Comprehensive Approach*. Elsevier Science, 2016. ISBN 9780128045794. URL <https://books.google.pt/books?id=u010DAAAQBAJ>.
- Yosra Hajjaji, Wadii Boulila, Imed Riadh Farah, Imed Romdhani, and Amir Husain. Big data and iot-based applications in smart environments: A systematic review. *Computer Science Review*, 39:100318, 2021. ISSN 1574-0137. doi: <https://doi.org/10.1016/j.cosrev.2020.100318>. URL <https://www.sciencedirect.com/science/article/pii/S1574013720304184>.
- Hajar Hantouti, Nabil Benamar, Tarik Taleb, and Abdelquoddous Laghrissi. Traffic steering for service function chaining. *IEEE Communications Surveys Tutorials*, 21(1):487–507, 2019. doi: 10.1109/COMST.2018.2862404.
- Frederik Hauser, Marco Häberle, Daniel Merling, Steffen Lindner, Vladimir Gurevich, Florian Zeiger, Reinhard Frank, and Michael Menth. A survey on data plane programming with P4: fundamentals, advances, and applied research. *CoRR*, abs/2101.10632, 2021. URL <https://arxiv.org/abs/2101.10632>.
- Intel. Different Wi-Fi Protocols and Data Rates. <https://www.intel.com/content/www/us/en/support/articles/000005725/wireless/legacy-intel-wireless-products.html>, 2022. Last visited 2022-04-27.
- International Telecommunication Union. IMT Vision – Framework and overall objectives of the future development of IMT for 2020 and beyond. https://www.itu.int/dms_pubrec/itu-r/rec/m/R-REC-M.2083-0-201509-I!!PDF-E.pdf, 2015. Last visited 2021-12-27.
- Internet Engineering Task Force (IETF). Network Configuration Protocol (NETCONF). <https://www.rfc-editor.org/rfc/rfc6241>, 2022a. Last visited 2022-05-05.
- Internet Engineering Task Force (IETF). RESTCONF Protocol. <https://www.rfc-editor.org/rfc/rfc8040>, 2022b. Last visited 2022-05-05.
- Internet Engineering Task Force (IETF). The YANG 1.1 Data Modeling Language. <https://datatracker.ietf.org/doc/html/rfc7950>, 2022c. Last visited 2022-05-05.

- Indrarini Irawati and Mohammad Nuruzzamanirridha. Spanning tree protocol simulation based on software defined network using mininet emulator. volume 516, pages 395–403, 03 2015. ISBN 978-3-662-46741-1. doi: 10.1007/978-3-662-46742-8_36.
- Md. Tariqul Islam, Nazrul Islam, and Md. Al Refat. Node to node performance evaluation through ryu sdn controller. *Wireless Personal Communications*, 112(1):555–570, 2020. doi: 10.1007/s11277-020-07060-4.
- Einollah Jafarnejad Ghomi, Amir Masoud Rahmani, and Nooruldeen Nasih Qader. Load-balancing algorithms in cloud computing: A survey. *Journal of Network and Computer Applications*, 88:50–71, 2017. ISSN 1084-8045. doi: <https://doi.org/10.1016/j.jnca.2017.04.007>. URL <https://www.sciencedirect.com/science/article/pii/S1084804517301480>.
- RhongHo Jang, DongGyu Cho, Youngtae Noh, and DaeHun Nyang. Rflow⁺: An sdn-based wlan monitoring and management framework. In *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, pages 1–9, 2017. doi: 10.1109/INFOCOM.2017.8056995.
- Michael Jarschel, Thomas Zinner, Tobias Hossfeld, Phuoc Tran-Gia, and Wolfgang Kellerer. Interfaces, attributes, and use cases: A compass for sdn. *IEEE Communications Magazine*, 52(6):210–217, 2014. doi: 10.1109/MCOM.2014.6829966.
- Mike Jia, Weifa Liang, Zichuan Xu, and Meitian Huang. Cloudlet load balancing in wireless metropolitan area networks. In *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, pages 1–9, 2016. doi: 10.1109/INFOCOM.2016.7524411.
- Raj Joshi, Ting Qu, Mun Choon Chan, Ben Leong, and Boon Thau Loo. Burstradar: Practical real-time microburst monitoring for datacenter networks. APSys '18, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450360067. doi: 10.1145/3265723.3265731. URL <https://doi.org/10.1145/3265723.3265731>.
- Fabian Kaup, Stefan Hacker, Eike Mentzendorff, Christian Meurisch, and David Hausheer. Energy models for NFV and service provisioning on fog nodes. In *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*, pages 1–7. IEEE, apr 2018. ISBN 978-1-5386-3416-5.
- Mohd Ehmer Khan and Farmeena Khan. A comparative study of white box, black box and grey box testing techniques. *International Journal of Advanced Computer Science and Applications*, 3(6), 2012.
- Zohaib Latif, Kashif Sharif, Fan Li, Md Monjurul Karim, Sujit Biswas, and Yu Wang. A comprehensive survey of interface protocols for software defined networks. *Journal of Network and Computer Applications*, 156:102563, 2020. ISSN 1084-8045. doi: <https://doi.org/10.1016/j.jnca.2020.102563>. URL <https://www.sciencedirect.com/science/article/pii/S1084804520300370>.

- Xinlu Li, Brian Keegan, Fredrick Mtenzi, Thomas Weise, and Ming Tan. Energy-efficient load balancing ant based routing algorithm for wireless sensor networks. *IEEE Access*, 7:113182–113196, 2019. doi: 10.1109/ACCESS.2019.2934889.
- LightReading Team. OpenDaylight Project Founded. <https://wiki.onosproject.org/display/ONOS/Southbound+protocols>, 2013. Last visited 2022-01-16.
- Felipe A. Lopes, Marcelo Santos, Robson Fidalgo, and Stenio Fernandes. A software engineering perspective on sdn programmability. *IEEE Communications Surveys Tutorials*, 18(2):1255–1272, 2016. doi: 10.1109/COMST.2015.2501026.
- Michael Menth, Habib Mostafaei, Daniel Merling, and Marco Häberle. Implementation and evaluation of activity-based congestion management using p4 (p4-abc). *Future Internet*, 11(7), 2019. ISSN 1999-5903. doi: 10.3390/fi11070159. URL <https://www.mdpi.com/1999-5903/11/7/159>.
- Moreira Miguel. Yawmd: multiple medium support and performance improvements for wmediumd. Master’s thesis, FCUP, 2020.
- Mininet Team. Mininet. <http://mininet.org/>, 2022a. Last visited 2022-01-11.
- Mininet Team. Mininet Wifi Manual. <https://usermanual.wiki/Pdf/mininetwifidraftmanual.297704656/html#pf9>, 2022b. Last visited 2022-01-11.
- Mirantis Blog. What’s in OpenDaylight? <https://www.mirantis.com/blog/whats-opensdaylight/>, 2022. Last visited 2022-01-17.
- Dritan Nace and Michal Piore. Max-min fairness and its applications to routing and load-balancing in communication networks: a tutorial. *IEEE Communications Surveys & Tutorials*, 10(4):5–17, 2008. doi: 10.1109/SURV.2008.080403.
- National Institute of Standards and technology. COMPUTER SECURITY RESOURCE CENTER). <https://csrc.nist.gov/glossary>, 2022. Last visited 2022-05-04.
- Jorge Navarro-Ortiz, Pablo Romero-Diaz, Sandra Sendra, Pablo Ameigeiras, Juan J. Ramos-Munoz, and Juan M. Lopez-Soler. A Survey on 5G Usage Scenarios and Traffic Models. *IEEE Communications Surveys & Tutorials*, 22(2):905–929, 2020. ISSN 1553-877X.
- Musa Ndiaye, Adnan M. Abu-Mahfouz, and Gerhard P. Hancke. Sdnmm—a generic sdn-based modular management system for wireless sensor networks. *IEEE Systems Journal*, 14(2):2347–2357, 2020. doi: 10.1109/jsyst.2019.2927946.
- Network Working Group. Internet Security Glossary, Version 2). <https://www.ietf.org/rfc/rfc4949.txt>, 2007. Last visited 2022-05-04.
- Niu Y., Li Y., Jin D., Su L., Vasilakos A. V. A survey of millimeter wave communications (mmwave) for 5g: opportunities and challenges. *Wireless Networks*, 21(8):2657–2676, 2015.

- ONAP Team. ONAP Architecture. <https://docs.onap.org/en/guilin/guides/onap-developer/architecture/onap-architecture.html>, 2022a. Last visited 2022-01-16.
- ONAP Team. ONAP Overview. <https://docs.onap.org/en/guilin/guides/overview/overview.html>, 2022b. Last visited 2022-01-16.
- Open Networking Foundation. OpenFlow Switch Specification v1.5.1. <https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf>, 2015. Last visited 2022-01-09.
- Open Networking Foundation. ONOS FEATURES. https://opennetworking.org/wp-content/uploads/2019/12/ONOS-Features_v1.pdf, 2019. Last visited 2021-08-16.
- Open Networking Foundation. P4 Language Tutorial. https://opennetworking.org/wp-content/uploads/2020/12/P4_D2_East_2018_01_basics.pdf, 2020. Last visited 2021-10-19.
- Open Networking Foundation. ONF OpenFlow Conformance: Certified Product List. <https://opennetworking.org/product-registry/>, 2022a. Last visited 2022-01-10.
- Open Networking Foundation. SD-CORE. <https://opennetworking.org/sd-core/>, 2022b. Last visited 2022-01-17.
- Open Networking Foundation. SD-RAN. <https://opennetworking.org/sd-ran/>, 2022c. Last visited 2022-01-17.
- OpenDaylight Project. OpenDaylight Controller Overview. <https://docs.opendaylight.org/en/stable-silicon/user-guide/opendaylight-controller-overview.html>, 2021. Last visited 2022-01-16.
- OpenDaylight Team. P4 Language Simple Router. <https://github.com/opendaylight/p4plugin/blob/master/apps/simple-router/src/main/resources/p4/simple-router.p4>, 2018. Last visited 2022-01-11.
- OpenDaylight Team. OpenDaylight Magnesium. <https://www.opendaylight.org/what-we-do/current-release/magnesium>, 2020. Last visited 2022-01-16.
- Overlaid. OpenFlow – Basic Concepts and Theory. <https://overlaid.net/2017/02/15/openflow-basic-concepts-and-theory/>, 2017. Last visited 2022-01-10.
- PANTHEONtech. FRINX's UniConfig is now powered by PANTHEON.tech's lighty.io. <https://www.globenewswire.com/news-release/2018/10/16/1622110/0/en/FRINX-s-UniConfig-is-now-powered-by-PANTHEON-tech-s-lighty-io.html>, 2018. Last visited 2021-08-01.
- PANTHEONtech. lighty.io 15. <https://github.com/PANTHEONtech/lighty>, 2021a. Last visited 2021-08-01.

- PANTHEONtech. About lighty.io. <https://lighty.io/lighty-io-sdn-made-easy/>, 2021b. Last visited 2021-08-01.
- PANTHEONtech. Software-Defined Networking: Made Easy. <https://lighty.io/>, 2021c. Last visited 2021-08-01.
- PANTHEONtech. OUR PARTNERS. <https://lighty.io/references/>, 2021d. Last visited 2021-08-01.
- PANTHEONtech. LIGHTY.IO COMPONENTS. <https://lighty.io/components/>, 2021e. Last visited 2021-08-01.
- PANTHEONtech. ONAP SDN-C. <https://lighty.io/onap-sdnc-on-lighty-io-use-case/>, 2022. Last visited 2022-01-16.
- RFC7950. The YANG 1.1 Data Modeling Language. <https://datatracker.ietf.org/doc/html/rfc7950>, 2016. Last visited 2022-01-16.
- Kari Rikkinen, Pekka Kyosti, Marko E. Leinonen, Markus Berg, and Aarno Parssinen. Thz radio communication: Link budget analysis toward 6g. *IEEE Communications Magazine*, 58(11):22–27, 2020. doi: 10.1109/MCOM.001.2000310.
- Ryu Team. WHAT’S RYU? <https://ryu-sdn.org/>, 2017. Last visited 2022-01-17.
- Salman, Ola and Elhadj, Imad H. and Kayssi, Ayman and Chehab, Ali. Sdn controllers: A comparative study. In *2016 18th Mediterranean Electrotechnical Conference (MELECON)*, pages 1–6, 2016. doi: 10.1109/MELCON.2016.7495430.
- Huaizhou Shi, R. Venkatesha Prasad, Ertan Onur, and I.G.M.M. Niemegeers. Fairness in wireless networks: issues, measures and challenges. *IEEE Communications Surveys & Tutorials*, 16(1):5–24, 2014. doi: 10.1109/SURV.2013.050113.00015.
- Miguel Silva, Pedro Teixeira, Christian Gomes, Duarte Dias, Miguel Luís, and Susana Sargento. Exploring software defined networks for seamless handovers in vehicular networks. *Vehicular Communications*, 31:100372, 2021. doi: 10.1016/j.vehcom.2021.100372.
- Sejun Song, Hyungbae Park, Baek-Young Choi, Taesang Choi, and Henry Zhu. Control path management framework for enhancing software-defined network (sdn) reliability. *IEEE Transactions on Network and Service Management*, 14(2): 302–316, 2017. doi: 10.1109/TNSM.2017.2669082.
- Thanos G. Stavropoulos, Asterios Papastergiou, Lampros Mpaltadoros, Spiros Nikolopoulos, and Ioannis Kompatsiaris. Iot wearable sensors and devices in elderly care: A literature review. *Sensors*, 20(10), 2020. ISSN 1424-8220. doi: 10.3390/s20102826. URL <https://www.mdpi.com/1424-8220/20/10/2826>.
- Hadar Sufiev, Yoram Haddad, Leonid Barenboim, and José Soler. Dynamic sdn controller load balancing. *Future Internet*, 11(3), 2019. ISSN 1999-5903. doi: 10.3390/fi11030075. URL <https://www.mdpi.com/1999-5903/11/3/75>.

- PanJun Sun. Security and privacy protection in cloud computing: Discussions and challenges. *Journal of Network and Computer Applications*, 160:102642, 2020. ISSN 1084-8045. doi: <https://doi.org/10.1016/j.jnca.2020.102642>. URL <https://www.sciencedirect.com/science/article/pii/S1084804520301168>.
- José Suárez-Varela and Pere Barlet-Ros. Towards a netflow implementation for openflow software-defined networks. In *2017 29th International Teletraffic Congress (ITC 29)*, volume 1, pages 187–195, 2017. doi: 10.23919/ITC.2017.8064355.
- THE LINUX FOUNDATION. ONOS Project Joins Linux Foundation in Strategic Partnership. <https://www.linuxfoundation.org/press-release/onos-project-joins-linux-foundation-in-strategic-partnership/>, 2015. Last visited 2021-08-16.
- The Linux Foundation. ODL User Stories. <https://www.opendaylight.org/use-cases/stories>, 2021a. Last visited 2022-01-16.
- The Linux Foundation. Platform Overview. <https://datatracker.ietf.org/doc/html/rfc7950>, 2021b. Last visited 2022-01-16.
- Brian Trammell and Elisa Boschi. An introduction to ip flow information export (ipfix). *Communications Magazine, IEEE*, 49:89 – 95, 05 2011. doi: 10.1109/MCOM.2011.5741152.
- Travis Fagerness. Estimating Wireless Range. <https://www.allaboutcircuits.com/technical-articles/wireless-range/>, 2015. Last visited 2021-12-20.
- Ning Wang, Kin Hon Ho, George Pavlou, and Michael Howarth. An overview of routing optimization for internet traffic engineering. *IEEE Communications Surveys Tutorials*, 10(1):36–56, 2008. doi: 10.1109/COMST.2008.4483669.
- Paul Zanna, Pj Radcliffe, and Karina Gomez Chavez. A method for comparing openflow and p4. In *2019 29th International Telecommunication Networks and Applications Conference (ITNAC)*, pages 1–3, 2019. doi: 10.1109/ITNAC46935.2019.9077951.
- Guowei Zhang, Fei Shen, Yang Yang, Hua Qian, and Wei Yao. Fair task offloading among fog nodes in fog computing networks. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–6, 2018. doi: 10.1109/ICC.2018.8422316.
- Yuming Zhang, Yan Liu, Lingfeng Guo, and Jack Y. B. Lee. Measurement of a large-scale short-video service over mobile and wireless networks. *IEEE Transactions on Mobile Computing*, pages 1–1, 2022. doi: 10.1109/TMC.2021.3139893.

Appendices

Appendix A

Terminology

This appendix serves the purpose of listing and explaining critical concepts necessary to understand the dissertation work.

We used literature work, protocol documentation, and online glossaries, like [National Institute of Standards and technology, 2022], [Network Working Group, 2007] and [Computer Hope Team, 2022], as references for the terminology concepts.

- **Action Set** is an Openflow (OF) pipeline concept. Each packet that enters a device for pipeline processing has an action set as a way to accumulate actions while the packet is processed by each table. These actions are executed in a specified order when pipeline processing ends [Open Networking Foundation, 2015].
- **Advanced Blackholing** is a network management activity. Standard blackholing allows redirection of traffic as a means to protect infrastructure from congestion, while data analysis is conducted. Advanced Blackholing uses the Software Defined Network (SDN) paradigm to allow operators to specify fine-grained drop policies as a way to automate the process [Dietzel et al., 2017].
- **Anomaly detection** “An intrusion detection method that searches for activity that is different from the normal behaviour of system entities and system resources” [Network Working Group, 2007]. By constantly monitoring and analysing traffic, these devices can detect when an unusual type of data is transferred or odd data volumes are sent.
- **Broadcast prevention** is a feature of Software Defined Network Controller (SDN-C) that blocks broadcast messages from typical discovery protocols. These protocols exist in traditional networking paradigms to discover new elements of the network, but in SDN, this responsibility lies in the controller.
- **Congestion control** manages incoming traffic in an effort to avoid service degradation caused by congestion. This can be accomplished by monitoring packet delay and reducing sent rates or dropping queued packets [Ghaffari, 2015]

- **Fairness** A property of an access protocol for a system resource whereby the resource is made equitably or impartially available to all eligible users [Network Working Group, 2007].
- **Load balancing** is a network management activity. It is the ability of a network device efficiently distribute incoming traffic across redundant connections or multiple servers depending on how busy each device is, thus making the links and server instances less clogged [Computer Hope Team, 2022].
- **Multi-access Edge Computing (MEC)** is a paradigm whereby operators open their edge Radio Access Network (RAN) to authorised third-parties cloud applications and services, enabling faster and flexible deployability to subscribers. MEC utilises 5G technologies to provide low latency and high bandwidth of cloud capabilities in edge infrastructure, closer to where data is produced [Filali et al., 2020].
- **Millimeter Wave (mmWave)** is a radio access technology that operates in high frequency bands, from 24 to 100 GHz but can go up to 300 GHz [Admin, 2020]. Compared with traditional 2.4 and 5 GHz communications, it has smaller coverage and smaller penetration but gives higher data rates and less interference, due to reduced spectrum competition. [Niu Y., Li Y., Jin D., Su L., Vasilakos A. V., 2015]. This technology emerged as a way to enable large bandwidths and high data rates in 5G wireless networks.
- **Network Coding** is a networking technique whereby accumulating the information of various transmissions, network throughput is increased, outperforming routing solutions [Bassoli et al., 2013]. A traditional example is in Figure A.1, where network coding surpasses routing throughput in a butterfly network. Node **S** generates traffic types **x** and **y**. Each sink, **R1** and **R2** needs to receive both traffic types but each link can only transmit either **x** or **y** in one time unit. Using routing, all the information reaches the sinks in 5 time units, but network coding only needs 4 time units. This is because node 3 encodes the information, link 3-4 transmit the two types of data and node 4 decodes the data.
- **Service Function Chaining** “(SFC) is a mechanism that allows various service functions to be connected to each other to form a service enabling carriers to benefit from virtualized software defined infrastructure” [Bhamare et al., 2016].
- **Traffic engineering** is a technique of network evaluation and consequently network optimisation whereby an operator can manually define traffic flow paths in an effort to more efficiently use network resources. Normally this is an offline process that helps predict the behaviour of the network in certain conditions [Wang et al., 2008].
- **Traffic Steering** is the activity of enforcing the desired filtering, modification or optimisation of traffic when traffic engineering evaluations are performed. “Defined as the forwarding and routing logic of traffic among service Functions” [Hantouti et al., 2019].

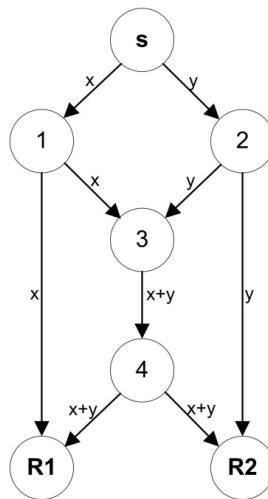


Figure A.1: Network coding in a butterfly network [Bassoli et al., 2013]

Appendix B

Openflow Pipeline and Control Channel

This appendix serves the purpose of detailing the steps involved in the OF pipeline and also the interface between OF enabled devices and the SDN controller. Gathering this knowledge was crucial to reaching our goals but it isn't necessary to understand this work.

Openflow Pipeline

The OF packet pipeline can be divided into ingress processing and egress processing. The Ingress processing is mandatory and egress processing is optional. In the ingress processing, a packet enters a device via an ingress port, and packet information may be modified, such as updating certain fields in the packet header, see Figure B.1.

In the egress processing, packets before being sent to the output port are matched against egress tables which can also update packet headers, or even drop the packet.

The ingress and output ports can be of different types: **physical ports** are “real” hardware ports of OF devices; **logical ports** are device-dependent abstractions that can map to multiple hardware ports and perform some modifications on packets; **reserved ports** are defined by the OF specification and represent special actions (e.g, reserved port **ALL** can be used as an output port to mirror the packet, reserved port **CONTROLLER** represents the OF channel used to communicate with the controller) [Open Networking Foundation, 2015].

A packet starts **ingress processing** at flow table 0 where it tries to match with a table entry, highest priority entry first. Matching of packets against table entries can use packet headers, ingress port, metadata field (used to pass information between 2 flow tables), and other pipeline fields (values attached to the packet for pipeline processing and not associated with packet headers [Open Networking Foundation, 2015]).

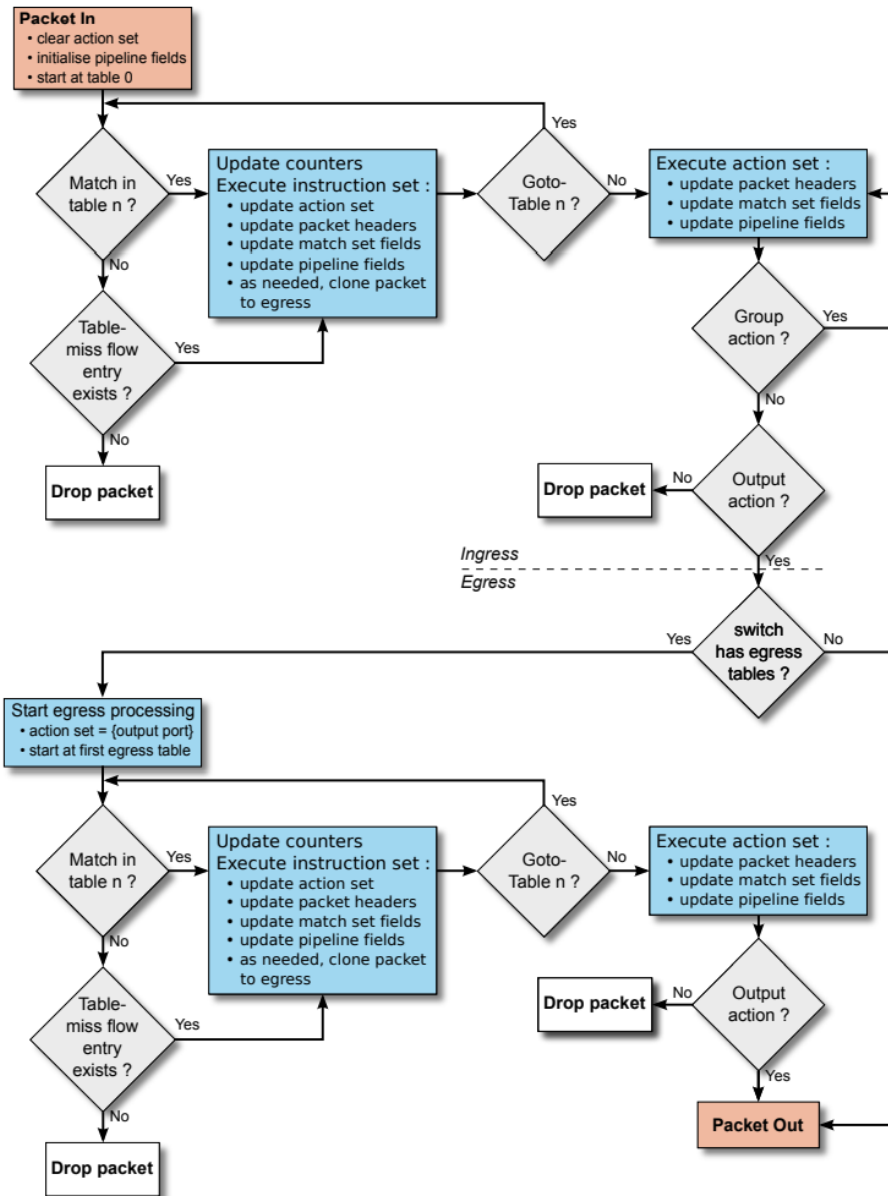


Figure B.1: Flowchart of packet pipeline in an Openflow switch [Open Networking Foundation, 2015]

If no match is found a table miss entry is triggered. This is a special table entry, with priority 0, that can be introduced in tables to work as a fallback in the case no matching occurs in the other table entries. This entry can be configured to drop the packet, send it to the controller over the OF channel, or could continue to the next step in pipeline processing [Overlaid, 2017].

When matching occurs the **instruction set** of the entry is executed. These instructions either modify the pipeline processing (e.g. direct the packet to another flow table), contain a set of actions to add to the **action set** of the packet or contain a list of actions to perform immediately to the packet.

The packet **action set** is a list of actions that each ingress packet has. It starts empty and during processing, the action set of each packet can be modified and

populated with actions to be executed in the specified order when ingress pipeline processing terminates.

If one of the instructions in the instruction set is the `Goto-Table` the processing continues at the specified table. The instruction set of the last flow table doesn't have this instruction, thus pipeline processing terminates.

At the end of the ingress pipeline, the actions of the actions set are executed. Within these actions, if a `Group` action is executed the packet is then forward to a group to be further processed, and if a `Output` action is executed the packet is forward to the specified output port to start egress processing [Open Networking Foundation, 2015].

If the switch has egress tables, egress processing starts. If no valid egress table is configured as the first egress table the packet is forward out through the specified output port. Note that the egress tables identification attribute needs to have a higher value than the identifier of ingress tables. This is important to stop loops in the pipeline, because packets can only be forward, using the `Goto-Table` action, to a table with higher identifiers [Open Networking Foundation, 2015].

At the beginning of egress processing, the action set contains only an output action for the current output port, and the values of the pipeline fields are preserved from ingress processing [Open Networking Foundation, 2015]. For the most part egress processing is similar to ingress processing. At the end, the pack is forward out of the switch.

Instructions, Actions, and Counters

Each flow entry can be populated with instructions to be executed when a packet matches the entry. There are different types of instructions available in the latest version of OF, some are required to be implemented by switches and some are optional. Table B.1 compiles a list of possible instructions to use with OF. Execution order is given by the order of instructions in the table.

Table B.1: Instructions of Openflow version 1.5.1 [Open Networking Foundation, 2015]

Requirement	Instruction	Description
Optional	<code>Apply-Actions</code> action(s)	Applies the specific action(s) immediately to the packet. May be used to modify the packet between two tables or to execute multiple actions of the same type.
Required	<code>Clear-Actions</code>	Clears all the actions in the action set immediately. Support of this instruction is required only for table-miss flow entries and is optional for the rest.
Required	<code>Write-Actions</code> action(s)	Merges the specified set of action(s) into the current action set. If an action of the given type exists in the current set, overwrite it.
...		

Table B.1: Instructions of Openflow version 1.5.1 (continued)

Requirement	Instruction	Description
Optional	Write-Metadata metadata / mask	Writes the masked metadata value into the metadata field.
Optional	Stat-Trigger stat thresholds	Generate an event to the controller if some of the flow statistics cross one of the stat threshold values.
Required	Goto-Table next-table-id	Indicates the next table in the processing pipeline. The table-id must be greater than the current table-id. This instruction is optional if the pipeline only has one table.

Some instructions require a list of actions in their configurations. The action set also keeps a list of actions to execute. There are different types of actions available in the latest version of OF, some are required to be implemented by switches and some are optional. Table B.2 compiles a list of possible actions to use with OF. Execution order is given by the order of actions in the table.

Table B.2: Actions of Openflow version 1.5.1 [Open Networking Foundation, 2015]

Requirement	Action	Description
Required	Output port_no	This action forwards a packet to a specified Openflow port where it starts egress processing.
Required	Group group_id	Process the packet through the specified group.
Required	Drop	There is no explicit action to represent drops. Instead, packets whose action sets have no output action and no group action must be dropped
Optional	Set-Queue queue_id	This action sets the queue id for a packet. The queue id value determines the queue used for scheduling and forwarding the packet when the output action is used. It is used to provide Quality of Service (QoS).
Optional	Meter meter_id	Directs packet to the specified meter. If the switch supports meters, this action is mandatory.
Optional	Push-Tag/Pop-Tag ethertype	Switches may support the ability to push/pop tags(e.g. push/pop a VLAN header onto the packet, push/pop a new MPLS shim header onto the packet).
Optional	Set-Field field type value	The various Set-Field actions are identified by their field type and are used to modify the values of respective header fields in the packet.
Optional	Copy-Field src field type dst field type	This action copies data between any header or pipeline fields.
Optional	Change-TTL ttl	Modifies the values of some packet properties (e.g. IPv4, Time to Live (TTL), IPv6, Hop Limit).

Several counters can be used in OF, some are required to be implemented by switches and some are optional. Table B.3 compiles a list of possible counters to use with OF.

Table B.3: Counters of Openflow version 1.5.1 [Open Networking Foundation, 2015]

Requirement	Counter	Bits
Per Flow Table		
Required	Reference Count (active entries)	32
Optional	Packet Lookups	64
Optional	Packet Matches	64
Per Flow Entry		
Optional	Received Packets	64
Required	Received Bytes	64
Optional	Duration (seconds) ^d	32
Optional	Duration (nanoseconds) ^d	32
Per Port		
Required	Received Packets	64
Required	Transmitted Packets	64
Optional	Received Bytes	64
Optional	Transmitted Bytes	64
Optional	Receive Drops	64
Optional	Transmit Drops	64
Optional	Receive Errors ^r	64
Optional	Transmit Errors	64
Optional	Receive Frame Alignment Errors	64
Optional	Receive Overrun Errors	64
Optional	Receive CRC Errors	64
Optional	Collisions	64
Required	Duration (seconds) ^d	32
Optional	Duration (nanoseconds) ^d	32
Per Queue		
Required	Transmit Packets	64
Optional	Transmit Bytes	64
Optional	Transmit Overrun Errors	64
Required	Duration (seconds) ^d	32
Optional	Duration (nanoseconds) ^d	32
Per Group		
Optional	Reference Count (flow entries)	32
Optional	Packet Count	64
Optional	Byte Count	64
Required	Duration (seconds) ^d	32
Optional	Duration (nanoseconds) ^d	32
Per Group Bucket		
Optional	Packet Count	64
Optional	Byte Count	64
...		

^d The amount of time the {*} entry has been installed in the switch

^r The total of all receive and collision errors

Table B.3: Counters of Openflow version 1.5.1 (continued)

Requirement	Counter	Bits
Per Meter		
Optional	Flow Count	32
Optional	Input Packet Count	64
Optional	Input Byte Count	64
Required	Duration (seconds) ^d	32
Optional	Duration (nanoseconds) ^d	32
Per Meter Band		
Optional	In Band Packet Count	64
Optional	In Band Byte Count	64

^d The amount of time the {*} entry has been installed in the switch

^r The total of all receive and collision errors

Openflow Control Channel

The OF control channel is the interface that allows OF enabled devices to talk with the SDN controller. Devices can support multiple interfaces to communicate with controllers as a way to share management of the switch. Through this interface, three message types can be exchanged, Controller-to-switch, Asynchronous, and Symmetric. This channel is usually encrypted using Transport Layer Security (TLS) but may be run directly over TCP [Overlaid, 2017].

Controller-to-switch messages are initiated by the controller and used to directly manage or inspect the switch. Switch response might be necessary [Open Networking Foundation, 2015; Overlaid, 2017]. The possible messages of this type are:

- **Feature** where the controller requests the identity and basic capabilities of a switch. The switch responds to the request. Commonly performed when establishing a Control Channel connection;
- **Configuration** serves to set or query configuration parameters of the switch;
- **Modify-State** used to manage flow/group entries or action buckets of a group;
- **Read-States** used to retrieve statistics of the switch (e.g current configuration, statistics, and capabilities);
- **Packet Outs** used by the controller to send a packet out of a switch port. The message sent must also contain a list of actions to be applied to the packet;
- **Barrier** request and reply messages are used by the controller to guarantee that message dependencies have been met or to receive notifications for completed operations;
- **Role-Request** used when a switch connects to a different controller as a way to define the role of each of them (e.g. equal, slave, master);

- **Asynchronous-Configuration** used by the controller to create an additional filter for asynchronous messages received. It is also used to query an existing filter.

Asynchronous messages are initiated by the switch, without controller solicitation, and are used to update the controller on a switch state change or to announce a packet arrival [Open Networking Foundation, 2015; Overlaid, 2017]. The possible messages of this type are:

- **Packet-in** is used to transfer a packet to the controller. Used when the switch forwards a packet to the reserved port CONTROLLER;
- **Flow-Removed** informs the controller that flow entry has been removed. This notification is only sent to the controller if the flow entry was configured with a flag OFPFF_SEND_FLOW_REM;
- **Port Status** informs the controller of a change in port configuration or state changes;
- **Role-status** informs the controller of a change of its role (e.g when a new controller elects itself master);
- **Controller-Status** informs the controller when the status of an OF channel changes;
- **Flow Monitoring** used to inform the controller of changes in a flow table if the controller defines monitors to track changes in that table.

Symmetric messages are initiated either by the switch or controller and sent without solicitation [Open Networking Foundation, 2015; Overlaid, 2017] These messages include:

- **Hello** messages are exchanged upon connection startup;
- **Echo** verifies the liveness of connection and can be used to measure latency or bandwidth. An echo reply is expected to be received;
- **Error** are used to notify problems to the other party;
- **Experimenter** provides a standard way for OF switches to offer additional functionality OF message type space.

Appendix C

P4 Pipeline and Example

This appendix explains the behaviour of the data plane programming model of the Programming Protocol-Independent Packet Processors (P4) language, the pipeline steps of the model and also contains an example of a P4 program that does simple routing. This information is needed to fully understand the protocol, but it is not necessary to understand this work.

P4 Pipeline

There are different data plane programming models but we will focus on Protocol-Independent Switching Architecture (PISA) because it is the model of the P4 language.

PISA consists of three major programmable components, a programmable parser, a programmable match-action pipeline, and a programmable deparser.

The **parser** handles the field extraction of the incoming serialised packet information. The programmer can declare headers of protocols that will be extracted to an organised structure with help of a directed flow graph.

The **match-action** pipeline consists of a sequence of match-action units where each unit can have one or more match-action-tables (MATs), see Figure C.1.

The MATs consist of lookup keys and corresponding actions. Parsed packet information is matched against the lookup keys of the MATs to try to find an entry that matches. When an entry is selected, packet header information can be modified by the corresponding actions. The control plane can influence the run-time behaviour by modifying the table's lookup key entries and corresponding actions that populate the MATs

The **deparser** converts the packet headers of the organised structure back into serialised information.

Besides the programmable components, there can be fixed-function components in the pipeline. Depending on the device's architecture, it can have these fixed function blocks between MATs, before the parser, or after the deparser. Some

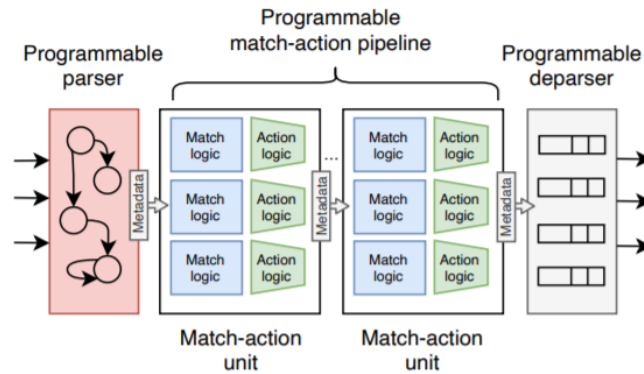


Figure C.1: PISA model [Hauser et al., 2021]

fixed function examples would be header checksums, ingress/egress port blocks, packet replication mechanisms for multicast, and traffic managers to handle packet buffering, queuing, and scheduling [Hauser et al., 2021].

P4 Routing Template

This means that P4 programs need to be designed to comply with the target’s architecture: Figure C.2 represents the components that make up a popular P4 pipeline model, *V1model*, and in pages 157 through 160 we present a P4 template for routing program, to use in with *V1model* architecture model [OpenDayLight Team, 2018].

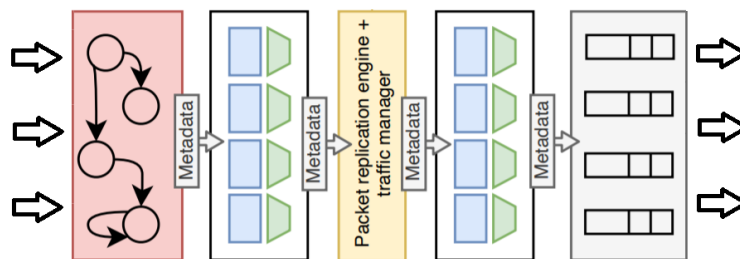


Figure C.2: V1model pipeline architecture [Hauser et al., 2021]

The *V1model* architecture is constituted by a parser, ingress MATs, ingress checksum, traffic manager, egress MATs, egress checksum, and deparser.

The P4 template program starts by defining a constant variable to represent the IPv4 ethertype that will be useful in the parser. Then the header structures for ethernet and ipv4 are defined and used to for the packet header structure.

```

Header definition
1 #include <core.p4>                22         bit<8>    diffserv;
2 #include <v1model.p4>             23         bit<16>   totalLen;
3                                     24         bit<16>   identification;
4 const bit<16> TYPE_IPV4 = 0x800;  25         bit<3>    flags;
5                                     26         bit<13>   fragOffset;
6 /***** H E A D E R S *****/      27         bit<8>    ttl;
7                                     28         bit<8>    protocol;
8 typedef bit<9>  egressSpec_t;      29         bit<16>   hdrChecksum;
9 typedef bit<48> macAddr_t;         30         ip4Addr_t srcAddr;
10 typedef bit<32> ip4Addr_t;        31         ip4Addr_t dstAddr;
11                                     32     }
12 header ethernet_t {               33     struct metadata {
13     macAddr_t dstAddr;             34         /* empty */
14     macAddr_t srcAddr;             35     }
15     bit<16>  etherType;            36
16 }                                   37     struct headers {
17                                     38         ethernet_t  ethernet;
18 header ipv4_t {                   39         ipv4_t      ipv4;
19     bit<4>    version;              40     }
20     bit<4>    ihl;                  41
21

```

The next block has the packet parser and the ingress checksum definition. The parser works like a flow graph. It starts in the *state start* and successfully ends in an *acceptable state*. This parser will first go to the ethernet parser block to extract the corresponding packet header. Afterwards, the flow graph goes to the IPV4 parser, if the ethernet etherType indicates that there is one, if not it goes to an accept state and finishes. In the IPV4 parser, the packet IPV4 header information is extracted and the flow graph successfully terminates.

The ingress checksum is also defined but there is no logic in the method in this implementation.

Appendix C

```
Parser+Checksum definition
42 /***** P A R S E R *****/
43 parser MyParser(packet_in packet,
44     out headers hdr,
45     inout metadata meta,
46     inout standard_metadata_t
47     standard_metadata) {
48
49     state start {
50         transition parse_ethernet;
51     }
52
53     state parse_ethernet {
54         packet.extract(hdr.ethernet);
55         transition select(
56             hdr.ethernet.etherType) {
57             TYPE_IPV4: parse_ipv4;
58             default: accept;
59         }
60     }
61
62     state parse_ipv4 {
63         packet.extract(hdr.ipv4);
64         transition accept;
65     }
66 }
67
68 /****C H E C K S U M****/
69
70 control MyVerifyChecksum(
71     inout headers hdr,
72     inout metadata meta) {
73     apply { }
74 }
75
```

The ingress control block is defined in the next snippet. The apply method checks if the IPV4 header was successfully extracted during packet parsing, to then perform packet matching with table `ipv4_lpm`. The table structure is defined (possible matching parameters and possible actions to apply to packets) as well as the logic of the actions that can be applied to packets (drop and `ipv4_forward`). Taking a look at `ipv4_forward`, this function changes header parameters to perform forwarding of the packet to another switch.

```

_____ Ingress Processing _____
76 /**** I N G R E S S ****/
77
78 control MyIngress(inout headers hdr,
79     inout metadata meta,          100
80     inout standard_metadata_t     101
81     standard_metadata) {          102
82                                     103
83     action drop() {                104
84         mark_to_drop();            105
85     }                                106
86                                     107
87     action ipv4_forward(           108
88         macAddr_t dstAddr,         109
89         egressSpec_t port) {       110
90                                     111
91         standard_metadata.egress_spec 112
92             = port;                113
93         hdr.ethernet.srcAddr         114
94             = hdr.ethernet.dstAddr;  115
95         hdr.ethernet.dstAddr         116
96             = dstAddr;              117
97         hdr.ipv4.ttl
98             = hdr.ipv4.ttl - 1;
99     }

```

```

table ipv4_lpm {
    key = {
        hdr.ipv4.dstAddr: lpm;
    }
    actions = {
        ipv4_forward;
        drop;
        NoAction;
    }
    size = 1024;
    default_action = NoAction();
}
apply {
    if (hdr.ipv4.isValid()) {
        ipv4_lpm.apply();
    }
}

```

The egress control block and egress checksum of the template are present next. The egress control block is empty, thus there is no egress processing. The egress checksum component has some logic to perform checksum validation in the IPV4 header of the packet. This is important because this header might have been modified in previous control blocks.

```

_____ Egress+Checksum definition _____
118 /**** E G R E S S ****/
119 control MyEgress(inout headers hdr,
120     inout metadata meta,          136
121     inout standard_metadata_t     137
122     standard_metadata) {          138
123     apply { }                      139
124 }                                  140
125                                  141
126 /**** C H E C K S U M ****/      142
127 control MyComputeChecksum(       143
128     inout headers hdr,            144
129     inout metadata meta) {        145
130                                     146
131     apply {                         147
132         update_checksum(           148
133             hdr.ipv4.isValid(),    149
134             { hdr.ipv4.version,
135             hdr.ipv4.ihl,

```

```

        hdr.ipv4.diffserv,
        hdr.ipv4.totalLen,
        hdr.ipv4.identification,
        hdr.ipv4.flags,
        hdr.ipv4.fragOffset,
        hdr.ipv4.ttl,
        hdr.ipv4.protocol,
        hdr.ipv4.srcAddr,
        hdr.ipv4.dstAddr },
        hdr.ipv4.hdrChecksum,
        HashAlgorithm.csum16);
    }
}

```

This last template segment contains the deparser block and the the V1Switch

function. The deparser reconstructs the indicated headers before the packet is forwarded out of the switch and the V1Switch function declares all the functions necessary for the V1model.

```

    _____ Deparser+Main definition _____
150 /**** D E P A R S E R ****/          159 /**** S W I T C H ****/
151                                     160 V1Switch(
152 control MyDeparser(packet_out packet, 161     MyParser(),
153     in headers hdr) {                162     MyVerifyChecksum(),
154     apply {                            163     MyIngress(),
155     packet.emit(hdr.ethernet);         164     MyEgress(),
156     packet.emit(hdr.ipv4);            165     MyComputeChecksum(),
157     }                                  166     MyDeparser()
158 }                                     167 ) main;
```


Appendix D

Framework Vision in the First Semester

This appendix contains detailed information documenting the vision of the dissertation in the first semester and the planning done to achieve our objectives.

Our main goal was to **develop a modular framework layer** above the typical SDN-C for network monitoring and management **independently of the SDN-C** and related technologies used in topologies.

Having highly interchangeable framework components enables easier integration of a new technology when necessary: to manage the network with a new controller, a new module needs to be implemented to act as an interface for the rest of the framework components, see Figure D.1

For example, the blocks inside the bracket communicate with each other to enable the management functionalities of the framework. If the network needs to change from using controller A to using controller B, the only block that needs replacement is the orange block “Interface Controller A”. The business logic of the management and monitoring algorithms built in the framework (green blocks) could be still utilised.

To reach our main objective at the time, we defined the following goals:

1. **Information collection:** elaborate a module to collect and store network statistics and topology information required for management and monitoring;
2. **Engine for algorithms:** build and deploy a server to import and run the custom management algorithms.
3. **Framework asset management:** build a framework module to list all assets and to approve/remove them as devices that can act on the network;
4. **Framework user management:** build a framework module to list and manage user accounts with permission to act on the topology, or use the data in the framework;

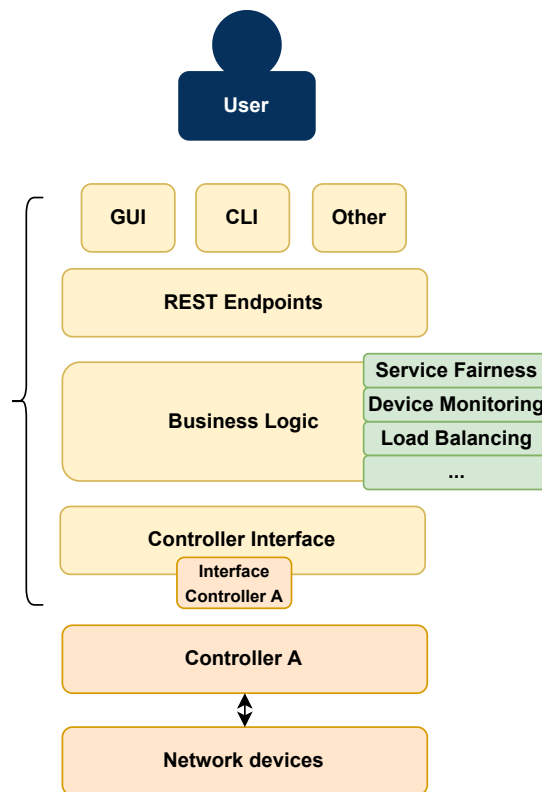


Figure D.1: Vision for the Framework in the First semester

5. **End-points for monitoring operations:** expose end-points to retrieve information from the framework, in order to be used by external tools. Exposes user and asset information, network statistics and available algorithms.
6. **End-points for management operations:** exposes end-points so that external tools can orchestrate network elements. Allows management of user and asset information and activation/deactivation and configuration for algorithms.