FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

An SDN-based Overlay Networking Solution for Transparent Multi-homed Vehicular Communications

Agostinho Filipe de Almeida Coimbra Maia

DISSERTATION

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Rui Lopes Campos

Second Supervisor: Helder Martins Fontes

July 15, 2021

Abstract

The train is an increasingly used transportation means for any type of travel, whether for leisure or for business trips. However, access to the Internet while traveling by train is still an issue. For example, there may be a lack of train-to-infrastructure network coverage in segments of the railway and Internet access failures when passengers move between carriages or when the communications system switches between network interfaces enabling train-to-infrastructure connectivity. This motivates the development of new communications solutions to overcome the problem.

In recent years, there has been a trend towards the Software Defined Networking (SDN) paradigm. In this new paradigm there is a central controller with an holistic view of the network, which enforces network policies and configurations defined at a logical central point. This concept has been applied in many research projects but its implementation has been mostly focused on Data Centers and telecom operator networks.

In this work, we propose a new solution, named Transparent Multi-homed Vehicular Communications (TMVC) that consists of using multiple gateways in the same train, one per carriage, which communicate with an SDN controller and establish an overlay network on top of the physical infrastructure of one or more telecom operators providing coverage to the train along the path. The SDN controller manages the overlay network based on Virtual Extensible Local Area Network (VXLAN) with the objective of increasing the reliability, Quality of Service (QoS) and availability of the Internet access service to the passengers. The TMVC solution aims to take advantage of the different mobile operators available along the path and is capable of supporting seamless handovers and load balancing between multiple Vehicle-to-Infrastructure networks.

The TMVC was evaluated experimentally. Although the implementation and evaluation were developed in a virtualized system, the results show that an SDN architecture can be used to reconfigure the overlay network with low average delay and to distribute traffic between multiple train-to-infrastructure links.

Resumo

O comboio é cada vez mais uma alternativa válida no que diz respeito a qualquer tipo de deslocação, seja para lazer ou viagens de trabalho. Contudo, o acesso à Internet durante uma viagem de comboio ainda é um problema. Por exemplo, existe falta de cobertura de rede em alguns pontos da linha ferroviária seguida pelo comboio e o accesso à Internet é interrompido quando os passageiros do comboio se movem entre carruagens ou quando o sistema de comunicação alterna entre as várias interfaces de rede do comboio para a infraestrutura. É com base nessas lacunas que surge a motivação para desenhar uma solução que resolva o problema.

Nos últimos anos, tem existido uma aposta frequente no paradigma *Software Defined Networking* (SDN). Neste novo paradigma, existe um controlador central que detém uma visão completa de toda a rede, obrigando a que as configurações de rede sejam definidas numa entidade lógica centralizada. Este conceito tem sido aplicado em muitos projetos de investigação mas a sua implementação tem sido dedicada especialmente a *Data Centers* e nas redes dos diferentes operadores de telecomunicações.

Neste trabalho, é proposta uma nova solução denominada *Transparent Multi-homed Vehicular Communications* (TMVC) que consiste no uso de múltiplos *gateways* no mesmo comboio, um por carruagem, que comunicam com o controlador central, estabelecendo uma rede *overlay* em cima de uma infraestrutura física composta por um ou mais operadores de telecomunicações que fornecem cobertura ao longo da linha ferroviária. O controlador SDN gere toda a rede *overlay* baseada em *Virtual Extensible Local Area Network* (VXLAN) com o objetivo de aumentar a fiabilidade e a disponbilidade do serviço de acesso à Internet aos passageiros. A solução TMVC pretende assim tirar partido dos diferentes operadores móveis disponiveis ao longo da linha e é capaz de suportar mecanismos de *handover* e balanceamento de cargas entre as várias interfaces de rede do comboio para a infraestrutura.

A solução TMVC foi avaliada experimentalmente. Apesar da implementação e avaliação terem sido desenvolvidas num ambiente virtualizado, os resultados obtidos revelaram que uma arquitetura SDN pode ser utilizada para reconfigurar eficientemente uma rede *overlay* e para criar mecanismos que permitam distribuir tráfego por múltiplas ligações entre o comboio e a infraestrutura.

Acknowledgments

First of all, i would like to express my gratitude to my supervisors Helder Martins Fontes and Rui Lopes Campos for their outstanding support and for always being available. Their vision and their motivation were of great importance for the realization of this dissertation.

I would like also to thank to all the Cisco Academy team, especially to Bruno Silva, Daniel Valente and to my colleague and personal friend Bruno Mauricio, for all the time we spent together working and discussing the best approach to complete successfully the dissertation.

Last but not the least, i would like to refer the support of my family and friends, especially from my grandfather for passing me all of his personal values and for being an eternal example for me.

Agostinho Filipe Maia

Contents

List of Figures

List of Tables

Abbreviations

API	Application Programming Interface
ARP	Address Resolution Protocol
BGP	Border Gateway Protocol
CLI	Command Line Interface
CPU	Central Processing Unit
DC	Data Centers
FIFO	First In First Out
HO	Handover
ICMP	Internet Control Message Protocol
IP	Internet Protocol
ISP	Internet Service Providers
LACP	Link Aggregation Control Protocol
LAN	Local Area Network
LB	Load Balancing
LC	Least Connections
MLD	Multicast Listener Discovery
MPLS	Multi-Protocol Label Switching
MTU	Maximum Transfer Unit
NAT	Network Address Translation
NIC	Network Interface Card
NS	Namespace
NVGRE	Network Virtualization Using Generic Encapsulation Routing
NVP	Network Virtualization Protocol
ODL	OpenDaylight
OF	Open Flow
ONF	Open Networking Foundation
OVS	OpenvSwitch
OVSDB	OpenvSwitch Database Management Protocol
POF	Protocol Oblivious Forwarding
QoS	Quality of Service
RR	Round Robin
SCTP	Stream Control Transmission Protocol
SDN	Software Defined Networking
SoA	State of the Art
SSH	Secure Shell
STP	Spanning Tree Protocol
SVI	Switch Virtual Interface
TCP	Transmission Control Protocol
TMVC	Transparent Multi-homed Vehicular Communications
TNI	Tenant Network Identifier
UDP	User Datagram Protocol

V2I	Vehicle-to-Infrastructure
VLAN	Virtual Local Area Network
VNI	VXLAN Network Identifier
VM	Virtual Machine
VPN	Virtual Private Networks
VTEP	Virtual Tunnel Endpoint
VXLAN	Virtual Extensible Local Area Network
WAN	Wide Area Network
Wi-Fi	Wireless Fidelity
XMPP	Extensible Messaging and Presence Protocol

Chapter 1

Introduction

1.1 Context

In recent years we witnessed a revolution in the mobile networks area. Nowadays, users have grown accustomed to be able to access all services and applications anytime, everywhere. One particular scenario where this happens is in public transportation such as trains and buses, where people commonly spend their time using smartphones, whether for working or leisure purposes. In this sense, there has been an increasing offer of on-board Internet services via Wireless Fidelity (Wi-Fi), providing passengers a way to overcome limitations such as finite mobile data plans, lack of cellular interfaces (e.g., laptops and tablets) and cellular signal attenuation by the cabin of the vehicle.

A study made by BWCS consultancy revealed that the number of European citizens that will use Wi-Fi in trains will be multiplied by five until 2028, reaching 5300 million connections per year [1]. In a study done in the United Kingdom, the same company also reported that 72% of the passengers on work travel would be willing to switch the airplane by train as their preferred transport with the guarantee of a functional Wi-Fi service [2].

Currently, implemented solutions typically consider only one cellular operator to access the infrastructure or more than one cellular operator but through a single gateway/mobile router. For example, in Portugal, there is a Wi-Fi solution in use in two categories of long-distance trains (*Alfa Pendular* and *Inter Cidades*). This solution allows connections to multiple operators but only as a failover mechanism. If the actual connection to the primary mobile operator fails, a different connection to an alternative operator is used. Nevertheless, when using this strategy other problems arise. For instance, when switching to the alternative connection or when the user switches carriages, the active user connections are broken due to the different Wide Area Network (WAN) Internet Protocol (IP) address used.

1.2 Problem Definition and Motivation

Using a single cellular operator or a single mobile router to access the infrastructure creates a single point of failure. If the cellular operator in use does not have coverage in a specific zone, Internet access fails and the solution becomes an unreliable system. On the other hand, the connection to multiple operators as a failover mechanism does not take advantage of combining multiple links simultaneously, which can lead to Quality of Service (QoS) improvements.

This has brought up the interest in developing a system based on Virtual Extensible Local Area Network (VXLAN) that allows to encapsulate traffic from legacy user terminals using the standard IP network stack, taking advantage of a central unity that controls the operation of the overlay network system and each respective network hardware on-board the trains. This concept of central unit controlling the network is motivated by the increasing use of the Software Defined Networking (SDN) paradigm in nowadays networks. The centralized SDN controller, with an holistic view of the overlay network, can then perform actions such as handover (HO) between the different interfaces, as the user moves along the train with minimal connection loss to the Internet, and selection of multiple available links for simultaneous usage between the train and the infrastructure to optimize the resilience and aggregated bandwidth of the overlay network.

The design and implementation of the referred system is the main focus of this dissertation.

1.3 Objectives

The goal of this dissertation is to develop a communications system based on the SDN paradigm, optimizing the usage of the available network access links between the train and the infrastructure. Taking that into consideration, the following objectives are considered:

- Design a Layer 2 overlay network based on VXLAN to be established between each subsystem (carriage and Internet gateway) and the central unity (SDN controller) that permits user movement between carriages and consequently user terminals HO between different interfaces with minimal connection loss to the Internet.
- Implement an algorithm in the SDN controller that allows the creation of load balancing (LB) rules in each SDN agent to improve the QoS provided by the VXLAN-based overlay network.
- Develop a prototype that implements the new referred communications system. This system consists of two main modules: the SDN agents running on each carriage with connection to the Internet gateway and the SDN controller supervising and creating rules that manipulate the overlay network behavior.
- Test and validate the solution experimentally, emulating each network element making part of the architecture such as the centralized SDN orchestrator, train carriages, the user terminals, an Internet Gateway and multiple Vehicle-to-infrastructure (V2I) connections representing different network operators.

1.4 Contributions

The two main original contributions of this dissertation are:

- An SDN-based VXLAN solution, named TMVC, allowing transparent multi-interface vehicular Internet access for legacy user terminals and being capable of performing seamless HO between different interfaces with minimal connection loss to the Internet.
- An SDN-based LB mechanism compatible with the previously mentioned SDN-based overlay network, capable of choosing a combination of links/interfaces that connect each carriage to the Internet, optimizing the utilization of the available network resources and improving QoS.

1.5 Document Structure

This document is structured as follows. Chapter 2 presents the State of the Art (SoA) technologies which are relevant for this work, including the SDN paradigm, the most used SDN controllers in the market, the OpenFlow (OF) concept and its evolution, the solutions used to implement overlay networks with a brief reference to how they are used in Data Centers (DC), as well as the related work. Chapter 3 presents an overview of the system model and a detailed explanation of the RYU SDN framework, as well as the design and implementation of the TMVC solution. Chapter 4 focus on the experimental evaluation of the proposed solution, analyzing the amount of time that a system takes to react to specific network reconfiguration scenarios and how the traffic is distributed among multiple links. Finally, Chapter 5 presents the conclusions of this dissertation, as well as the known limitations and suggestions that can be implemented in future developments.

Chapter 2

State of the Art

This chapter covers the SoA analysis on the existing technologies and the solutions that may be used to solve the problem defined in Chapter 1. It is structured in the following four sections:

- Software Defined Networking: Its concept, its architecture, functions provided by an SDN controller and a comparison between the most used controllers in the market. Besides that, there is also a detailed approach to the OF protocol and its evolution as well a explanation related to the operation of a OF switch.
- Open vSwitch (OVS): An introduction to the technology, its architecture, networking scenarios where it is used and a precise explanation about OF group tables and how are they used to implement multipath algorithms in an virtualized system.
- Overlay Networks: Its concept, the most used technologies in the market with references to implementations used in DC and a comparison between the referred technologies;
- Related work: Similar work that has been published, identifying the main shortcoming and a comparison between the work done on the present project and the work done by the community.

2.1 Software Defined Networking

2.1.1 Concept

Networks have been growing in size and requirements and the way traditional IP networks are designed is complex, which difficults their reconfiguration in case of any change or fault. This is how SDN revolutioned the industry by creating flexibility and programmability in the network. Current networks are vertically integrated which means the control plane (responsible for deciding how to handle traffic) and the data plane (responsible for forwarding the traffic according to decisions taken by the control plane) are packed together inside the networking devices. This reduces flexibility and might decrease the evolution possibilities of networking infrastructure.

The SDN concept [3] consists in the separation between control and data plane in the networking domain. This separation turns switches into simple forwarding devices and the control logic operation is implemented in a logic centralized controller, the SDN controller. The control plane contains Layer 2 and Layer 3 route forwarding mechanisms, such as routing protocol neighbor tables and topology tables, IPv4 and IPv6 routing tables, Spanning Tree Protocol (STP), and the Address Resolution Protocol (ARP) table. Information sent to the control plane is processed by the Central Processing Unit (CPU).

The date plane is typically the switch fabric connecting the various network ports on a device. The data plane of each device is used to forward traffic flows. Routers and switches use information from the control plane to forward incoming traffic through the appropriate egress interface. Information in the data plane is typically processed by a special data plane processor without the CPU getting involved.

The architecture comparison between a traditional and SDN network can be visualized in Figure 2.1.

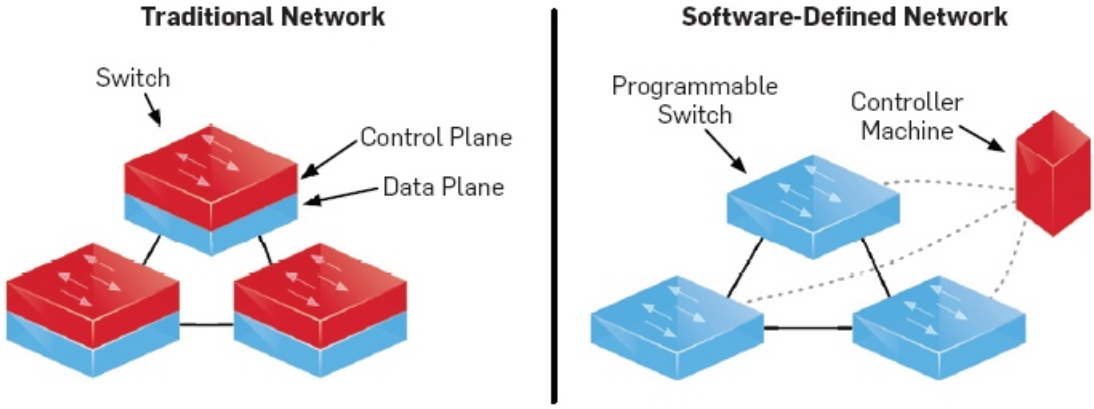

Figure 2.1: Traditional network vs SDN network [3].

2.1.2 SDN Architecture

The SDN architecture [3] [4] is constituted by three layers and four interfaces. The *data layer*, *control layer* and *application layer*. These layers communicate using the referred interfaces. The four key interfaces are: the *Northbound interface*, the *Southbound interface*, the *Westbound interface* and the *Eastbound interface*.

The data layer assimilates resources that are directly connected with customer traffic but also supports the resources needed to support virtualization, connectivity, availability and security. It is composed of multiple network equipments, which form the underlying network to forward network traffic. It has also the responsibility to handle data packets based on instructions given by the controller. Actions performed on this layer are forwarding, modifying and dropping the packet.

The control layer has the responsibility to make decisions on how packets should be forwarded. It is where is located the logic of the system. Functionalities such as topology discovery and maintenance, packet route selection and path failover mechanism are part of the control layer. In the next section three main SDN controllers will be explored, which are part of this layer.

The application layer contains one or more application which can have exclusive control of a set of resources exposed by the SDN controllers. It is an open area for developing innovative applications by using network information such as network topology, network statistics and network state.

Regarding the interfaces to communicate among the referred layers:

- **Northbound interface**: As stated before, SDN allows the exchange of information with applications running on top of the network. This information is exchanged using the Northbound Application Programming Interface (API). The Northbound API is not standardized, which means the form and frequency of the communication depends on the target application and the network.

- **Southbound interface**: It is the interface between the control and data planes. It provides programmatic control of all forwarding operations, capabilities advertisement, statistics reporting and event notification. Most used Southbound interfaces for SDN are OF, OVS Database Management Protocol (OVSDB) and Protocol Oblivious Forwarding (POF).

- **Westbound interface**: It allows the exchange of network state information to influence network routing decisions of each controller. For the information exchange, routing protocols like Border Gateway Protocol (BGP) can be an option.

- **Eastbound interface**: It allows the communication with control planes of non-SDN domains such as Multi-Protocol Label Switching (MPLS). Normally a translation module between SDN and the legacy technology is required.

  The representation of these interfaces in the SDN architecture can be visualized in Figure 2.2.

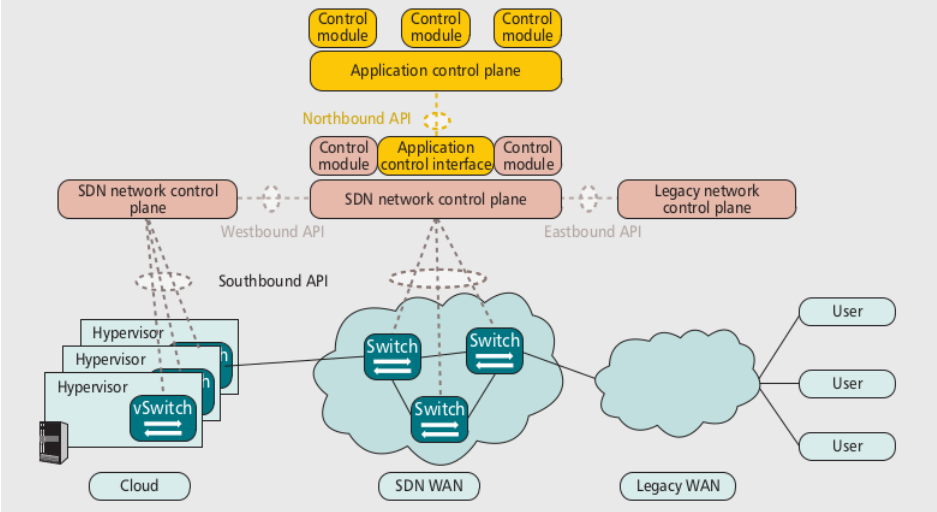

Figure 2.2: SDN architecture [4].

2.1.3 SDN Controller

An SDN controller is the entity that defines the data flows between the centralized control plane and the data plane on routers and switches.

Before travelling through the network, each flow must first get permission from the SDN controller, which verifies if the communication is permissible according to established network rules. If the controller allows the flow, it processes a route for the flow to take on and adds an entry related to the respective flow in each of the switches along the network path.

All complex functions are performed by the controller and it populates the flow tables. Three main SDN controllers implementations used in the market are the following: Ryu, Floodlight and OpenDayligh (ODL).

- Ryu controller is open source and under the Apache 2.0 license. It is written in Python and it is supported and deployed by NTT cloud data centers. Main source code can be found on GitHub, supported by Open Ryu community [5].
- The Floodlight Open SDN Controller is an enterprise-class, Apache-licensed, Java-based OF Controller. It is supported by a community of developers including a number of engineers from Big Switch Networks [6].
- ODL is an open source project supported by IBM, Cisco, Juniper, VMWare and several other major networking vendors. ODL is an SDN controller platform implemented in Java [7].

All these three controllers can use OF to interact with the forwarding plane (switches and routers) to modify how the network will handle traffic flows. In terms of virtualization, the three can also use Mininet emulator and OpenvSwitch (OVS) [5]. The comparison between the three controllers can be seen in Table 2.1.

	Ryu	Floodlight	OpenDaylight
Programming Language	Python	Java	Java
Northbound API	REST	REST, Java RPC, Quantum	REST, RESTCONF, XMPP
Southbound API	OpenFlow 1.0-1.5	OpenFlow 1.0-1.3	OpenFlow 1.0-1.3
Interface	CLI	CLI, WebUI	CLI, WebUI
Multithreading	Yes	Yes	Yes
Modularity	Medium	Medium	High
TLS support	Yes	Yes	Yes
Platform support	Linux, MacOS	Windows, Mac, Linux	Linux
Documentation	Good	Good	Good

Table 2.1: SDN controller's feature comparison table.

2.1.4 Open Flow

2.1.4.1 Introduction and Evolution

OF is currently the most used SDN standard and is composed by many specifications. It provides direct access and control over the data plane of virtual and physical network equipment such as routers and switches. Therefore, OF establishes itself as a communication protocol that allows network administrators to define network packet forwarding using software.

It was proposed by Stanford University and by the California University in Berkley, although the standardization process has been handled by the Open Networking Foundation (ONF). OF has been prototyped on Ethernet switches, IP routers, Wi-Fi access points, and WiMAX base stations by a variety of vendors [8], meaning that the OF should be seen as being compatible with legacy equipment and can add new functionality and services to the network. Currently OF establishes itself in version 1.5.

A summary of the evolution of OF, detailing the most relevant features introduced in each version, is presented below:

- OF version 1.0 [9] was released in December 2009 and the protocol had multiple number of changes since its inception [10]. This version supports a lookup into a single flow table and only supports Ethernet and IP protocols, not covering neither MPLS and ECMP.

- Version 1.1 [11] was released in February 2011 and introduced the table chaining concept, which allowed a packet to be modified or updated as well as going to a different table than the one where the packet was received. This version also introduced the concept of group tables, explored in detail in Section 2.2.4 and added MPLS label and MPLS traffic class to match fields.

  Released in December 2011, in OF version 1.2 [12] introduced IPv6 support (matching fields include IPv6 source address, destination address, Internet Control Message Protocol (ICMP) v6 code) and extensible matches (in previous versions, the order and length of match fields was fixed).

- Version 1.3 [13] was released in June 2012 and included support for overlay tunnels, a separate flow entry for table miss actions, per-connection event filtering (which provides a better filtering of connections to multiple controllers) and an extension of IPv6 headers. Besides that, this version includes the meter field, a switch element used to measure and control the rate of packets going in and out of the OF device.

- Released in October 2013, version 1.4 [14] introduced the concept of bundle, which are a group of instructions sent from the controller that are either all executed or none is. Instructions may be sent to a group of switches and then applied at approximately same time. This version also introduced flow table monitoring used for synchronization in a multi-controller system, used to notify the SDN controller if a set of flow table entries is modified by another controller in .

- The last version, 1.5 [15], was released in March 2015 and introduced Egress tables, where actions to be done when exiting through a port such as encapsulate or decapsulate a packet are present. Also introduced Transmission Control Protocol (TCP) Flags Matching, where parameters such as Syn, Ack and Fin might be used to detect the beginning and end of a TCP established connection.

In the context of the present dissertation, OF features such as the support for overlay tunnels, extension of IPv6 headers and group tables are usefull to design the intended solution.

2.1.4.2 OF Switch

An OF switch can have numerous flow tables, a group table and an OF channel. Each flow table is composed by flow entries and can communicate with the controller. The group table can configure the flow entries. OF switches are connected between each other using OF ports [16]. One example of an OF switch is OVS, detailed in Section 2.2.

The OF model can be visualized in Figure 2.3.

At the beginning, the data path of the OF routing devices has an empty routing table. This table is composed by several fields. Also, in this table are contained packet fields such as the destination of the different ports used and an action field in which is present the code for different inputs, such as packet forwarding or reception. When a new packet is received and has no match in the data flow table (this action is named the *table miss packet*), either can be dropped or can be forward to the controller, which handles it and makes a decision (whether to drop it or to add a new entry in the data flow table, creating then a mechanism to deal with similar packets that might be received in the future) [16].

There are two types of OF compliant switches: OF-only and OF-hybrid. The first type supports only OF operations and all packets are processed by the OF pipeline and can not be processed otherwise. OF-hybrid switches support both OF and standard Ethernet switching such as L2 Ethernet switching, Virtual Local Area Network (VLAN) isolation, L3 routing and QoS processing [13].

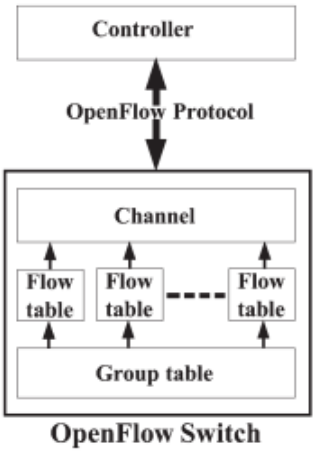

Figure 2.3: OF model [16].

The OF pipeline processing defines how packets communicate with flow tables. The minimum number of flow tables is one and in this case the pipeline processing is simplified. In Figure 2.4, it is possible to visualize the procedure that a packet follows since it enters the pipeline until the packet leaves it. Flow tables are sequentially numbered, starting at 0 and the pipeline processing always begins on that table.

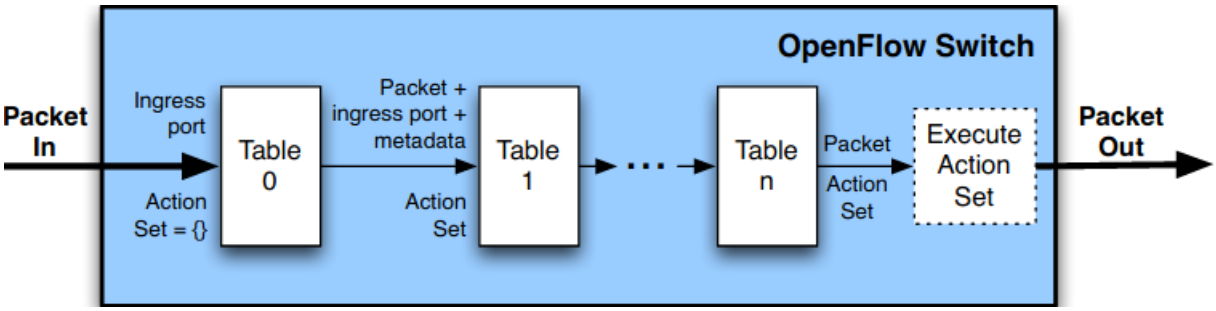

Figure 2.4: Pipeline process [13].

2.2 Open vSwitch

2.2.1 Introduction

Released in 2009, OVS is an open source implementation of a distributed virtual multilayer switch. Written in C, it enables network automation through programmable extensions and it supports standard management interfaces and protocols such as NetFlow, Link Aggregation Control Protocol (LACP) and 802.1D. In the academic field and data center networks, OVS has been extensively used as an OF switch. It can be installed on a physical server to connect all of the Virtual Machines (VM) running on the server with a real network. It can also run on a standalone Linux system, allowing the Linux system and its network interface cards (NIC) to act as a hardware switch. Several

switch vendors have also deployed OVS as an agent on their own hardware switches to support the OF protocol. Currently, OVS establishes itself in the 2.14.0 version, released in August 2020 [17].

2.2.2 Architecture

OVS is a software application that is divided in two main components: The user and the kernel space. According to their features, user-space modules can be divided in three groups: The management tool, the database service and the core component. The management tools include *ovs-vsctl*, *ovs-ofctl*, *ovs-dpctl*, *ovs-appetl* and *ovsdb-clients*. These tools allow network admnistrators to handle OVS directly, retrieving information such as port statistics, flow table modifications and other types of database operations. The second group, the database service, is manipulated by the *ovsdb-server* in order to store switch-level settings. It also comes with a JSON-RPC interface for interacting with *ovs-vswitchd* [18], a daemon that manages and controls any number of OVS switches on a local machine. The third category, the core section, where the *ovs-vswitchd* is included, is responsible for flow table lookups and packet processing. Being in the core group, the *ovs-vswitchd* is the most important component in the user space. It modifies the database using a predefined interface, maintains flow tables, parses the OF protocol and exchanges network information with the OF controller [19].

The kernel space is composed only by the OVS kernel module. Using multiple netlink sockets, this module manages one or more datapaths and exchanges information with *ovs-vswitchd*. In kernel space, a datapath is a software implementation of an OF switch. As referred previously, a datapath can use the ports on a hardware NIC as its switch ports. It overrides the default receive handler function in the kernel for these ports with its own handler function. As so, instead of the original system protocol stack, packets arriving at these ports are now managed by the OVS kernel module [19].

The OVS architecture can be visualized in Figure 2.5.

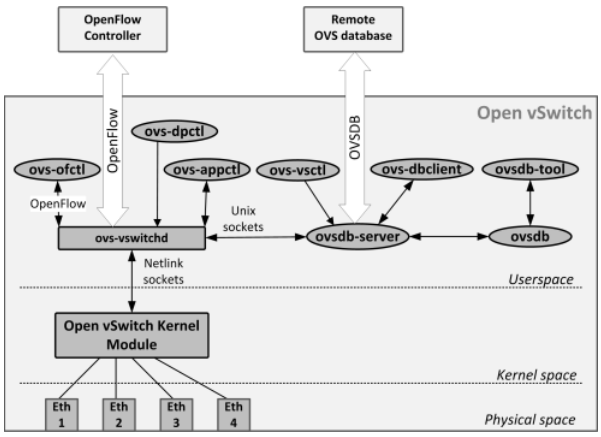

Figure 2.5: OVS Architecture [20].

2.2.3 OVS Configuration Scenarios

After understanding the OVS concept and its architecture, it is important to explore some OVS configurations [21]. To do so, some scenarios will be exposed and consequently, their traffic patterns will be explored and analyzed in what follows.

2.2.3.1 Scenario 1: Basic OVS Configuration

In this scenario, traffic from a guest domain travels across the OVS bridge to which it is connected. In Figure 2.6, a guest domain attached to *ovsbr0* communicates over *enp0s3* physical interface and a guest domain attached to *ovsbr1* communicates over *enp0s8* physical interface. On the other hand, traffic from a Linux host can use the native TCP/IP stack and any configured interfaces instead of communicating over any of the configured OVS bridges. Therefore, if *enp0s9* is configured and operational, all traffic to/from the Linux host itself will travel through this interface.

It is also important to refer that a physical Ethernet device that is part of an OVS bridge should not have an IP address. If one does, then that IP address will not be functional. To restore layer 3 functionalities, an IP address can be configured inside each OVS internal interface.

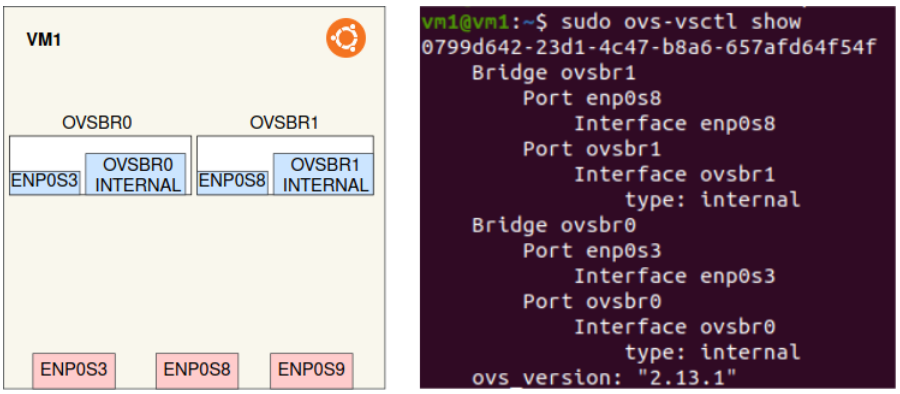

Figure 2.6: Basic OVS configuration.

2.2.3.2 Scenario 2: OVS Bonding

Before exploring this scenario, it is necessary to refer that network bonding is an extremely used process in the networking environment. Network bonding is the aggregation of multiple links into a single link in order to increase throughput and achieve redundancy in case one of links fails. Network administrators must guarantee the best network efficiency as possible and one of the most used methods to do so is LB between multiple physical/logical interfaces. However, the use of multiple physical/logical network connections might lead to broadcast storms and to solve that problem, STP is used. The problem is that STP blocks specific ports, reducing the efficiency of the system. In this scenario, the use of LACP solves the referred issue because this protocol allows devices to automatically configure and use logical link aggregates made up of several physical links. In Figure 2.7, a bond was created associating two physical interfaces *enp0s3* and *enp0s8*

into the logical port *bond0*. However, traffic from guest domain will still travel across *ovsbr0* (it is the only configured bridge) and traffic from the Linux host will still use available interfaces defined by the host TCP/IP stack.

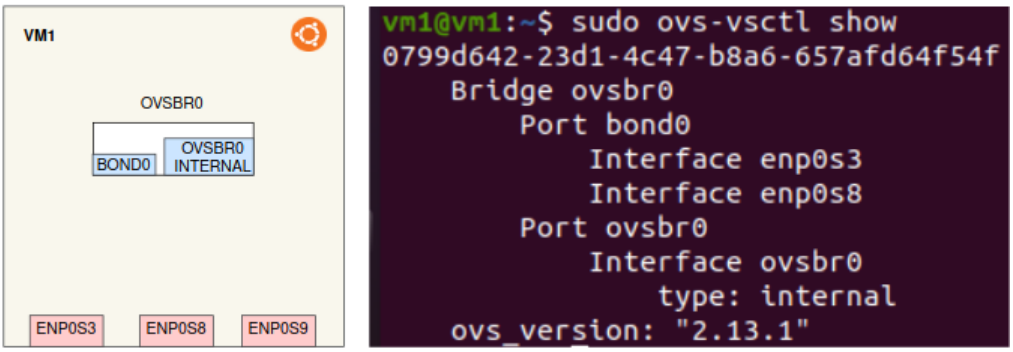

Figure 2.7: OVS bonding.

2.2.3.3 Scenario 3: Integration Bridge

This configuration is often used in Nicira Network Virtualization Protocol (NVP) and OpenStack and it is mainly used when setting up overlay networks. It is called integration bridge because the bridge has no physical interfaces attached. In Figure 2.8, guest domains attached to *intbr* can only use the VXLAN tunnel. This tunnel is created and maintained by the OVS process and it uses the host's TCP/IP stack and IP routing tables to communicate across the network. As an example, the interface *enp0s9* is on the 192.168.1.0/24 network and VXLAN traffic travels using that interface. On the other hand, traffic from guest domains attached to *ovsbr0* remains unchanged and will go through one of the physical interfaces according with the bonding configuration. Traffic from the Linux host also remains unchanged and will go according to the host TCP/IP stack.

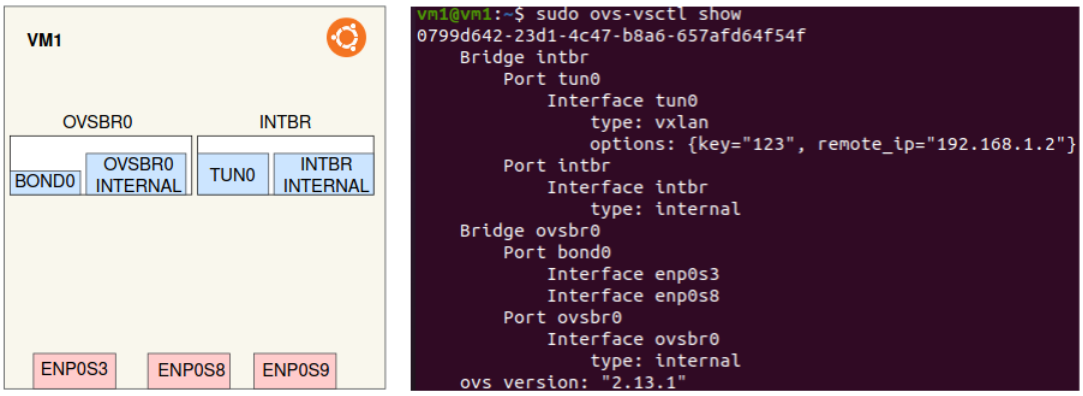

Figure 2.8: Integrated bridge.

2.2.3.4 Scenario 4: OVS Host Management

This scenario is used when it is necessary to setup a management interface to control the OVS. Its purpose is the same as the Cisco Switch Virtual Interface (SVI). An IP address has been assigned to the *ovsbr0* internal interface inside bridge *ovsbr0* but the OVS scheme is the same as presented in Figure 2.8. OVS internal interfaces can represent physical interfaces to the Linux host and therefore an IP address can be assigned and used for communication. As in the previous examples, traffic from guest domains attached to *ovsbr0* will go through the physical interfaces in the bond. The main difference is on the traffic from the Linux host and from the guest domains attached to the *intbr*. The only interface that the Linux host has is the *ovsbr0* internal interface and since this interface is attached to *ovsbr0*, all traffic will travel using the physical interfaces in the bond. Traffic from guest domains attached to *intbr* will also be directed through *ovsbr0*, since this is the interface that the Linux host uses, impacting the way traffic is directed.

2.2.4 OVS Multipath

As referred previously, when a bond is created, protocols such as LACP bundle physical ports into a single logical channel, increasing bandwidth and redundancy. However, using the SDN paradigm, instead of creating a bond and applying a layer 2 protocol (in the underlay network) to obtain a LB system, a multipath approach can be used in the overlay network, selecting specific interfaces through which traffic travels. A multipath system can be implemented using OF group tables. In Section 2.1.4.2, OF group tables have been referred but their purpose was not explored in depth. An OF group is an abstraction that permits to handle more complex and specialized packet operations that cannot be performed through a flow table entry. Each group receives packets as input and performs any OF actions on these packets but a group is not capable of performing any OF instructions, so it has not the ability to send packets to other flow tables or meters. Therefore, it is expected that packets have been matched before to entry to a group, since groups do not support matching on packets, being used only as mechanisms to perform advanced actions [22]. As represented in Figure 2.9, a group is composed by a separate list of actions and each action is referred as an OF bucket.

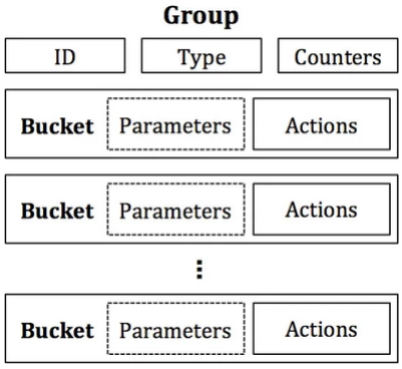

Figure 2.9: OF group structure [22].

There are four group types, the ALL group, the SELECT group, the FAST-FAILOVER group and the INDIRECT group.

The ALL group is considered to be the simplest as it will take any packet received as input and duplicate it so that each bucket present on the bucket list will work on it independently. Different actions can be in each bucket, allowing different operations to be performed on different copies of the packet. A representation of the ALL group type can be visualized in Figure 2.10.

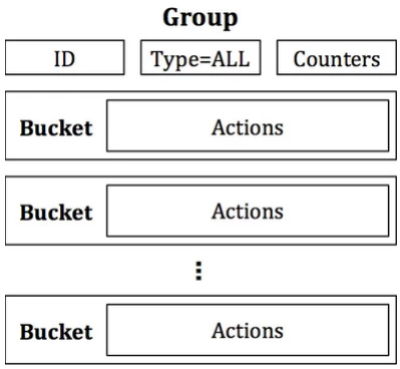

Figure 2.10: OF ALL group type [22].

The SELECT group is mainly designed for LB applications since each bucket in this group type has an defined weight and each packet entering the group is sent to a single bucket. The algorithm responsible for the bucket selection and packet distribution is dependent on the switch implementation, which means it can be manipulated by an OF controller. However, the algorithm can utilize the weight parameter assigned to each interface to balance traffic between links.

OVS 2.3 and earlier versions used the destination Ethernet address to choose a bucket in a select group. Since version 2.4 and later, OVS hashes a set of fields in the packet (source and destination Ethernet address, VLAN ID, Ethernet type, IPv4/v6 source and destination address and protocol, and for TCP and Stream Control Transmission Protocol (SCTP) only, the source and destination ports), mapping then the hash value to a bucket. This means that packets whose hashed fields are the same will always go to the same bucket (unless the hash includes fields that vary within a traffic flow, such as TCP flags or if a bucket has a watch port or group whose liveness changes during tests). Therefore, statistics regarding a small number of flows may still lead to an uneven distribution and inconsistent results [22] [23].

A representation of the SELECT group type can be visualized in Figure 2.11.

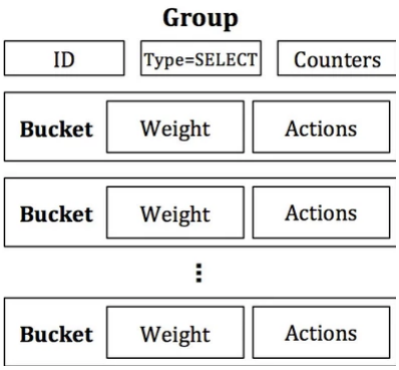

Figure 2.11: OF SELECT group type [22].

The FAST-FAILOVER group is mostly used to detect and react to port failures. As the other group types, this group has a list of buckets but each bucket has a watch port and/or a watch group as a special parameter. Both are used to monitor the status of the port/group, respectively. Only one bucket is used at a time and the bucket will not be changed unless there is a state transition (i.e, up to down). Despite there is no certainty of how long it will take to select a new bucket in a failure scenario, the transition time depends on search time to find a watch port/group that is up and available. One benefit from using this group type is that it is almost always faster than using the control plane to manage the port down and inserting a new flow. Link failure detetion and recovery occurs entirely in the data plane when using FAST-FAILOVER group type [22].

A representation of the FAST-FAILOVER group type can be visualized in Figure 2.12.

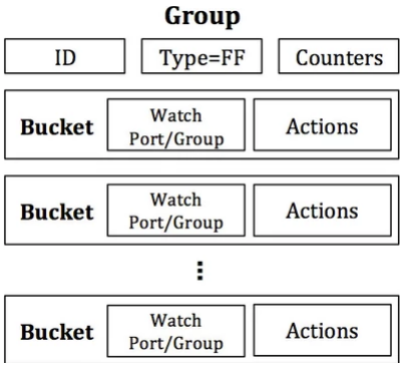

Figure 2.12: OF FF group type [22].

The INDIRECT group type is composed by a single bucket where all packets received by the group are sent. Its purpose is to cover a common set of actions used by many flows. If a set of flows match on different packet headers but share a set of actions, these flows can send packets to the INDIRECT group type, eliminating situations where a list of common actions for each flow are duplicated. In summary, it is used to simplify OF deployment by reducing flow table memory consumption [22].

A representation of the INDIRECT group type can be visualized in figure 2.13.

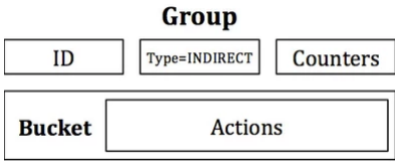

Figure 2.13: OF INDIRECT group type [22].

2.3 Overlay Networks

2.3.1 Concept

An overlay network is a computer network which is built on top of another network. It is a software method used to create layers of abstraction that allow to run multiple virtualized network layers on the top of the physical network. They are used to increase security, robustness and to provide new services for mobile users. One key feature about overlay networks is that they can be deployed on end hosts without the direct involvement of Internet Service Providers (ISP), not requiring new equipment or modifications to existing protocols [24].

Overlay Networking is related to many different protocols and standards such as IP, Virtual Private Networks (VPN) and IP multicast. As SDN development arises, two main technologies have been explored: VXLAN and Network Virtualization Using Generic Routing Encapsulation (NVGRE).

2.3.2 VXLAN

2.3.2.1 Necessity

As defined on RFC 7348 [25], VXLAN is an overlay technology designed to provide Layer 2 and Layer 3 connectivity over a common IP network. VXLAN accomplishes this by tunneling L2 frames into IP packets. More precisely, the VXLAN header, along the original Ethernet frame are encapsulated into a User Datagram Protocol (UDP) segment. The encapsulation provided by the VXLAN standard uses an 8-byte header, composed by 24-bit VXLAN Network Identifier (VNI) and multiple reserved bits. The 24-bit address space allows scaling virtual networks beyond the 4096 available with 802.1Q up to 16.7 million possible virtual networks. The VXLAN packet structure is represented in Figure 2.14.

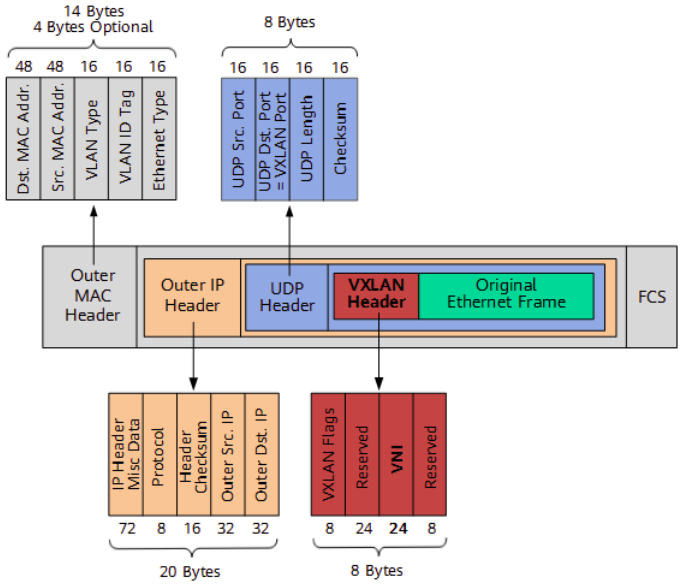

Figure 2.14: VXLAN packet structure [26].

When implementing network infrastructures, whether it is physical or virtual, one of the biggest problems is the depletion of IP addresses, which leads the search and reuse of available IP addresses on cancelled clients and on network devices that are no longer in use. Besides that, it is worth mention the lack of scalability in VLAN segmentation, especially in huge service providers networks and data centers. In order to solve this scalability issue, VXLAN technology is used as an alternative. On the other hand, VXLAN also proposes the creation of a virtual layer 2 network, which inserts in the system a new level of abstraction. More precisely, this creates the possibility of having two VMs located into two different geographically located data centers that are integrated into the same virtual VXLAN network [27].

The representation of the VXLAN abstraction is represented in Figure 2.15.

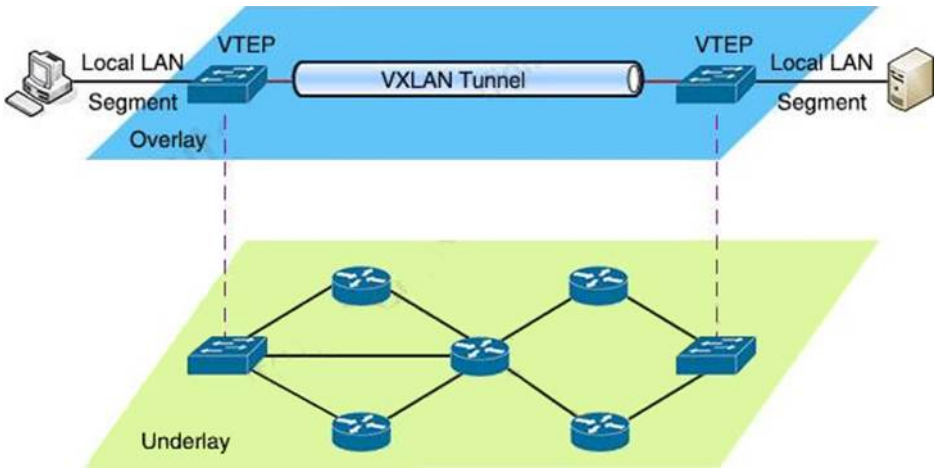

Figure 2.15: VXLAN overlay and underlay network [28].

2.3.2.2 How it Works

When understanding how VXLAN works, there are some key elements in the system that must be referred. VXLAN uses a Virtual Tunnel Endpoint (VTEP) device to perform encapsulation and de-capsulation. A VTEP can be an hardware or software (hypervisor) element in charge of instantiating the VXLAN tunnel. Each VTEP has two interfaces: one on the local LAN segment to support local endpoint communication (overlay interface) and the other interface for the transport IP network (underlay interface). A VTEP discovers other remote VTEPs for its VXLAN segments and learns about remote MAC address for VXLAN mappings through the underlay interface [27].

As it is possible to see in Figure 2.16, when VTEP1 receives an Ethernet frame from VM1 address to VM3, it uses the VNI (VMs with different VNIs cannot communicate) and the destination MAC to lookup in its forwarding table for the VTEP to send the packet to. Then, VTEP1 adds a VXLAN header that contains the VNI associated with the Ethernet frame, encapsulates the frame into a UDP packet and routes the packet to VTEP2 over the Layer 3 underlay network. VTEP2 decapsulates the original Ethernet frame and forwards it to VM3. VM1 and VM3 are completely unaware of the VXLAN tunnel and the Layer 3 underlay network between them [29].

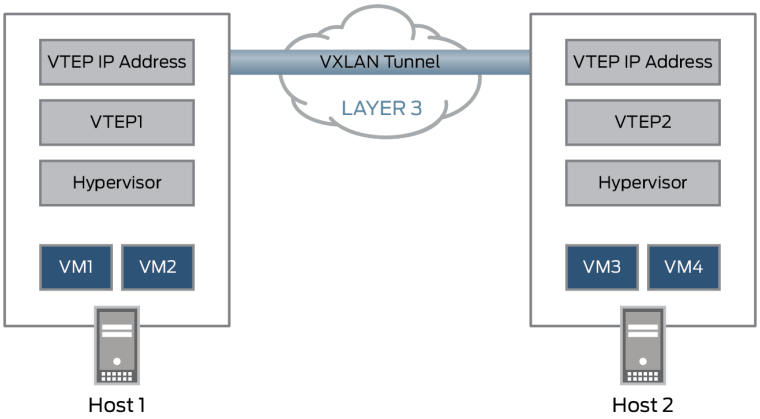

Figure 2.16: VXLAN tunnel example [29].

2.3.2.3 VM Migration in Data Centers and Analogy with the Intended Solution

One of the main issues that system administrators come across when working on DC is the migration of running VMs from one physical server to another with zero downtime, continuous service availability and full transaction integrity. This problem has some similarities to one of the problems presented in this dissertation. As a user moves along the train, the intention is that there is no connection loss.

To solve this problem, overlay networks technologies like VXLAN are used. As stated before, VXLAN technology can be used to establish a Layer 2 virtual network over any networks with reachable routes to implement Layer 2 interconnection. Packets encapsulated by VXLAN are sent by VMs over a VXLAN tunnel. VMs at the two ends of a VXLAN tunnel do not need to know the physical structure of the underlay network. As so VMs using IP addresses in the same network segment are in a Layer 2 domain logically, even if they are on different physical Layer 2 networks [30].

2.3.3 NVGRE

Defined on RFC 7637 [31], NVGRE was introduced by Microsoft, Intel, HP and Dell. It is identical to VXLAN with the main difference that the L2 Ethernet frame is encapsulated in a GRE/IP header instead of a UDP/IP header. GRE header uses a 24bit virtual subnet ID, called Tenant Network Identifier (TNI) and such as VXLAN it stands for 16 million unique virtual segments. [32].

As well in VXLAN, all NVGRE interfaces have assigned a unicast and multicast IP address. If the interface does not know where to send the packet, it uses the NVGRE multicast group as preferred address [33].

In Figure 2.17, it is possible to see the NVGRE packet format.

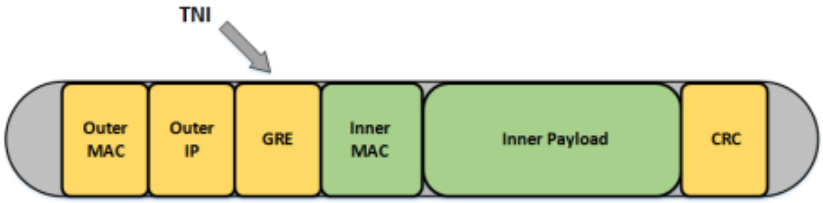

Figure 2.17: NVGRE packet format [33].

In Table 2.2, it is possible to see a comparison between VXLAN and NVGRE.

Data Plane	Encapsulation headers	Encapsulation overhead	Payload	Strengths	Weaknesses
VXLAN	Ethernet-IP-UDP-VXLAN	50 bytes	Ethernet	Widely deployed Multi-vendor support	Encapsulates only Ethernet frame payload
NVGRE	Ethernet-IP-GRE	42 bytes	Ethernet	Based on GRE	Lack of TCP/UDP headers make routing support challenging

Table 2.2: Comparison of the two main used overlay protocols.

2.4 Related Work

Over the past years some studies have been done regarding the intersection between SDN, OVS and overlay networks applications.

Fernando Ruano [34] created a virtual overlay network using VXLAN tunnels between three OVS managed by a RYU SDN controller. SDN based VXLAN is a concept that introduces SDN as the control plane and VXLAN as the data plane. The network topology which consisted of 3 servers was created to represent a DC. In each server there were 4 virtual containers, each one associated to a network with different VNIs. The study allowed to understand that the controller avoids manual flow configuration for each virtual switch port, improving the system efficiency. However, this study did not test how an overlay network behaves when users try to access the Internet and how it reacts if users moved from one server to another.

Another combination of SDN and VXLAN has been done in [35] but in this article the fundamental objective was to study an SDN based VXLAN architecture in Cloud Computing Networks. VXLAN uses IP multicasting techniques to increase overlay capacity by encapsulating the original message with headers at one VTEP and sending the duplications out to other VTEPs to jointly decide the destination host. This multicasting technique necessitates a significant amount of extra signalling traffic. A new architecture was proposed to enhance the multicast capability and to simplify VM migration. The authors argue that the SDN architecture proposed reduced IP multicasting extra signalling, reinforced no interruption of the system during VM migration and simplified the configuration for LB.

Despite not referring to overlay networks, the study presented in [36] explores an SDN based framework to ensure that multiple streams go over multiple paths in the network. The authors argue that data streams are forwarded through a single path even when are multiple paths available in the network, not benefiting from all the available bandwidth. To overtake this limitation, the Floodlight SDN controller computes disjoint paths from a given network topology, creating routing rules that map each stream to a distinct path. These rules are then received by a RESTful application through the use of Floodlight's Northbound API. The study showed improvements in terms of QoS, more precisely in terms of delay and throughput experienced by users.

In [37] the authors used the Floodlight SDN controller and implemented a fat-tree topology in the Mininet emulator to evaluate traditional LB algorithms. Metrics such as the latency and throughput of the network were compared using Least Connections (LC), Round Robin (RR) and Weighted RR algorithms. The study revealed that the average response time is higher when using RR and Weighted RR, when compared with LC, as expected. The same result was achieved when testing the throughput metric, being highest in the LC algorithm when compared to the other two chosen techniques.

In summary, four SDN based implementations were exposed to evaluate the SDN paradigm in different applications. The first work is a Ryu-based VXLAN application (as well as the TMVC solution) and it aimed to prove that an SDN architecture can be used to avoid manual flow configuration for each OVS. The second work presented also a Ryu-based VXLAN architecture but the main purpose is to reduce extra signalling in the VXLAN network and to simplify VM migration. The third work is a Floodlight application designed to ensure that different streams generated by multi-stream transport protocols follow multiple paths in the network and those paths can be visualized in a RESTful API. The last work was developed to evaluate throughput and latency metrics in an SDN architecture when using traditional LB algorithms.

To the best of our knowledge, the solutions presented in the SoA do not address the objectives defined for this dissertation, bringing up an opportunity to develop specific contributions regarding SDN-based overlay networks for supporting vehicle-to-infrastructure communications.

Chapter 3

The TMVC Solution

This chapter explains the solution designed to solve the problem presented in Section 1.2. It is structured in four main topics:

- Overview: Brief explanation of the TMVC solution, including the explanation of the rationale to use of the SoA technologies and how they were integrated into TMVC.
- RYU SDN Controller: Explanation of how the RYU SDN framework operates and detail of the events that it generates.
- Testbed Architecture: Detailed approach to the TMVC testbed architecture developed to test and evaluate the solution.
- RYU Application Implementation: Each sub system of the TMVC solution is explored and explained.

3.1 Overview

As stated before, the main objective of this dissertation is to develop an SDN-based VXLAN network that allows encapsulation of the original traffic from passengers, providing the Internet access through an Internet gateway that maintains the same WAN IP regardless of the interface used in the vehicle. One key feature of this solution is that the end-user can move between carriages with minimal connection loss and without the need of Wi-Fi access reconfiguration.

After some research and due to the fact that it is actually widely deployed in ISP nowadays, the technology chosen to implement the overlay network was VXLAN, since it permits that two machines using IP addresses in the same network segment are in a Layer 2 domain logically, even if they are on different physical Layer 2 networks, as long it exists Layer 3 connectivity between both machines.

The decisions taken by the SDN controller have as main goal to make use of all available resources in the network, providing LB between the different interfaces. In this work, the protocol used to perform the communication between the controller and the SDN agents is OF (explained in Section 2.1.4). In order to implement the gateway/SDN agent present in each carriage, OVS was chosen due to the fact that it supports a number of features that allow a network control system to respond and adapt as the environment changes (OVS supports OVSDB, which supports remote triggers that are used in the TMVC solution to create VXLAN interfaces in each carriage). Besides that, OVS supports thousands of simultaneous overlay tunnels implementations such as GRE or VXLAN.

The controller implementation used in this dissertation is the RYU framework because it is developed in Python and it has a lot of documentation available. RYU framework is event-oriented and those events will be explored in Section 3.2.2.

The system model consists of three fundamental components:

SDN agents: All traffic generated by the train passengers inside each carriage and destined to the Internet is forwarded to the OVS (SDN agent), which in turn encapsulates it and sends it to the SDN controller that must take decisions and install flows regarding the best link or set of links through which the traffic travels.

SDN controller: The main role of this module is to take decisions regarding the traffic that comes from the SDN agents and vice-versa. RYU SDN controller listens on TCP port 6633 and it reacts to events that describe reception of OF messages from each OVS.

Overlay Network: Each OVS acts like a VXLAN VTEP and it has a set of virtual VXLAN interfaces configured. Three of these interfaces are used to represent ISP links and the other one is used to connect one carriage to another. The link tunnel between carriages is usefull in a situation where the user, despite being connected to one OVS, uses an OVS located on other carriage to access the Internet.

The system model is represented in Figure 3.1. It is assumed that all carriages have connection to multiple ISP (representing the underlay network) and that each connected client integrates the overlay network and therefore connected clients stay in the same Local Area Network (LAN).

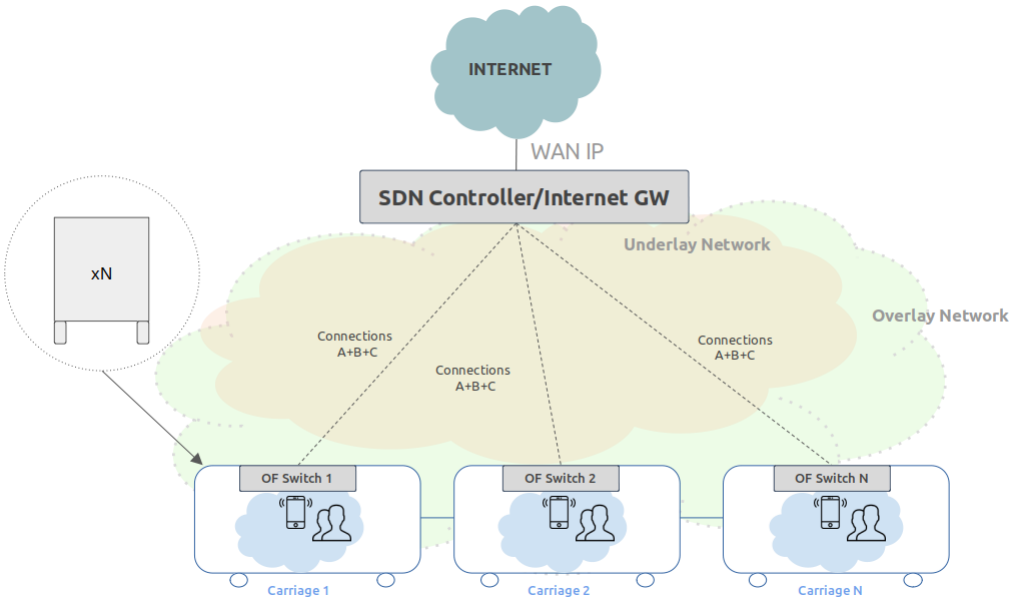

Figure 3.1: System model of the TMVC Solution.

3.2 Ryu SDN Controller

3.2.1 Application Programming Model

Ryu SDN controller has already been introduced in Section 2.1.3. However, the way that Ryu framework manages packets and events has not been explored yet.

Ryu applications are single threaded entities that incorporate a variety of features. Each Ryu application has an event receive queue which implements a First In First Out (FIFO). Besides that, each Ryu application has an event processing thread. When the event processing thread is called, an event handler is executed and it is inside that handler that software can be developed to perform specific functions in the system. No more events for the Ryu application can be processed when an event handler is blocked [38]. The programming model used for Ryu applications is represented in Figure 3.2.

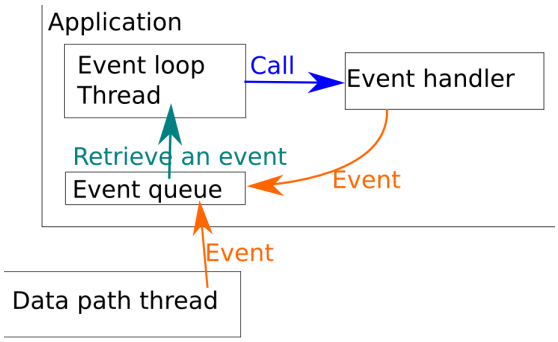

Figure 3.2: RYU application programming model [38].

3.2.2 RYU Events

Ryu reacts to events which are generated either by the core part of Ryu or Ryu applications. The *ryu.controller.handler.set ev cls* decorator allows a Ryu program to register its interest for a specific type of event. RYU has a module [39] used specifically to handle the reception of OF messages from connected switches such as an OVS. By convention, the name of the event class of OFP is in the *ryu.controller.ofp_event.EventOFP* type and the process type is defined next. As an example, *EventOFPPacketIn* is used for packet-in process type and *EventOFPStateChange* is used for state change.

When declaring an event handler, it is necessary to specify a negotiation phase (or a list of them) [40] for which events should be generated for this handler. The negotiation phases are the following:

- **HANDSHAKE_DISPATCHER:** Sending and waiting for hello message.
- **CONFIG_DISPATCHER:** Version negotiated and sent features-request message.
- **MAIN_DISPATCHER:** Switch-features message received and sent set-config message.
- **DEAD_DISPATCHER:** Disconnect from the peer. Or disconnecting due to some unrecoverable errors.

In the TMVC solution, five event types have been used: *EventOFPStateChange*, *EventOFPSwitchFeatures*, *EventOFPPacketIn*, *EventOFPPortStatus* and *EventOFPPortDescStatsReply*.

The **EventOFPStateChange** is an event class used for negotiation phase change notification. In the example in Section 3.3, the event has two negotiation phases to notify the connection or disconnection of a datapath (such as OVS).

```
@set_ev_cls(ofp_event.EventOFPStateChange,[MAIN_DISPATCHER, DEAD_DISPATCHER])
```

Figure 3.3: Event *OFPStateChange*.

The **EventOFPSwitchFeatures** is an event class used by the controller and the datapath to exchange information about the datapath features such as the ID, maximum number of packets buffered at once and number of tables supported by the datapath. In Figure 3.4, the event is in the *CONFIG_DISPATCHER* phase negotiation as explained above.

```
@set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
```

Figure 3.4: Event *OFPSwitchFeatures*.

The **EventOFPPacketIn** is an event class called when the controller receives an OF packet_in message. A scenario where this happens is when a datapath do not have the flows installed to perform specific actions and it is necessary to contact the SDN controller. In Figure 3.5, the event is in the *MAIN_DISPATCHER* phase negotiation because the datapath and the controller have already exchanged all the necessary information.

```
@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
```

Figure 3.5: Event OFPPacketIn.

The **EventOFPPortStatus** is an event class called when there is any update in an OF port in the datapath. This change could be a port added, deleted or modified to the OF datapath. Just like the *OFPPacketIn* event, it is called in the *MAIN_DISPATCHER* negotiation phase, as it is possible to visualize in Figure 3.6.

```
@set_ev_cls(ofp_event.EventOFPPortStatus, MAIN_DISPATCHER)
```

Figure 3.6: Event *OFPPortStatus*.

The **EventOFPPortDescStatsReply** is an event class called when there is a reply message to a description statistics request. Statistics involve parameters such as OF ports number, MAC address of a specific OF port, port speed, port name and port configuration.

```
@set_ev_cls(ofp_event.EventOFPPortDescStatsReply, MAIN_DISPATCHER)
```

Figure 3.7: Event *PortDescStatsReply*.

The RYU framework provides a group of libraries intended for networking applications. Examples are the packet library, the OVS library, OF-Config library, NETCONF library and the XFLOW library. In the TMVC solution, only the packet and OVS libraries are used. The **packet** library is used to parse and build multiple protocol packets such as Ethernet and ARP. The **OVS** library is used to connect to an OVS by setting up the IP address of the OVS instance. It is possible then to add or modify VXLAN ports inside the referred instance.

3.3 Testbed Architecture

3.3.1 Design

The testbed developed to implement and evaluate the TMVC solution is represented in Figure 3.8. It has five main components, each one hosted inside a VM. Each VM is running Ubuntu 20.04 LTS operating system.

The connection between all VMs in the system was performed using the Internal Networking mode in Oracle Virtual Box. An internal network is identified only by its name (internal network ID). If there are more than one active virtual network card with the same internal network ID, Oracle Virtual Box support drive will automatically wire the cards and act as network switch. Oracle Virtual Box support driver emulates a complete Ethernet switch, supporting broadcast, multicast frames as well as promiscuous mode.

Carriage 1: Hosted in VM1, it has six NICs. ETH0 connects to the RYU controller (hosted in VM5) using the CONTROL internal network. Several flow rules instantiated in each OVS and other OF messages travel through this link. ETH1, ETH2 and ETH3 represent the ISP links. Each of this NICs inside VM1 is connected to a NIC interface inside VM3. This ISP links can be divided in two paradigms: The underlay paradigm and the overlay paradigm. More precisely, ETH1-VM1 is connected to the ETH0-VM3 using the VODAFONE1 internal network (using the Internal Networking mode), which represents the underlay network/Layer 3 connectivity between VMs. However, TUN0-OVS1 virtual interface is connected to TUN0-OVS3 and represents the overlay network. The same process was applied to links ALTICE1 and NOS1.

ETH4 represents the link between carriages and the process is very similar to what has been explained before. This link was created to predict a situation where all ISP links used to access the Internet in a carriage are down and the traffic must be redirected to another carriage with Internet access.

ETH5 represents the NIC that is used to connect to the client that intends to access the Internet, therefore it is used as an OVS interface and it is connected to the ETH0-VM4 interface using the CLIENTA network.

Carriage 2: Hosted in VM2, the design is very similar to what was done for Carriage 1. ETH0 is also used as the control interface, receiving networking rules from the RYU SDN controller to be applied in OVS2. ETH1, ETH2 and ETH3 are also connected to NICs hosted in the VM4 and it represent VODAFONE2, ALTICE2 and NOS2 links, respectively.

ETH4 represents the link between carriages and ETH5 is used to establish a connection with the client hosted in VM4 using the CLIENTB network.

User terminal: It represents the terminal used to access to the Internet. Since the present dissertation has been executed in a fully virtualized environment, it was necessary to design a scenario to test HO between two carriages. Therefore, VM4 with two NICs was used and a network bridge was created as a logical combination of the referred interfaces. This architecture allowed to connect the bridge to one OVS (carriage), access the Internet and then shutdown the connection, connect the bridge to another OVS and test if the system allowed a minimal interruption regarding Internet access.

Internet OVS: Hosted in VM3, it is the end point of the ISP Links of each carriage. It is also the gateway to the Internet. It has a NIC (ETH6) belonging to the CONTROL internal network responsible for receiving OF rules. Besides the ISP NIC, VM3 has two more interfaces. ETH7 is the interface used to perform Internet Access. It is configured in Network Address Translation (NAT) Network mode in Oracle Virtual Box. As explained in Section 2.2.3.1, to configure an IP address into an OVS, the OVS internal interface should be used. This interface is used as the default gateway of the overlay network and all the traffic coming from the VXLAN tunnels that is destined to the Internet is redirected to the ETH7 interface. VETH0 is an interface used mainly for debugging purposes. It is connected to a Network Namespace (NS). Network namespaces isolate networking system resources such as networking devices, IPv4/IPv6 protocol stacks, IP routing tables, firewall rules and port numbers. Since NAT interfaces might cause some problems due to possible external access list configurations, it is usefull to create a client connected to the ISP switch and to test connectivity between the user terminal and the ISP OVS.

RYU SDN Controller: Hosted in VM5, it is the software application created to control all the system. It is the responsible for adding and removing flows, creating VXLAN tunnels, detect and control HO between carriage 1 and carriage 2, performing LB between the VXLAN tunnels (also referred as ISP links) and to redirect traffic from carriage 1 to carriage 2 or vice-versa in cases where one of the carriages has no Internet access. It has only one NIC, also inserted in the CONTROL internal network and it listen on TCP port 6633 for OVS connections.

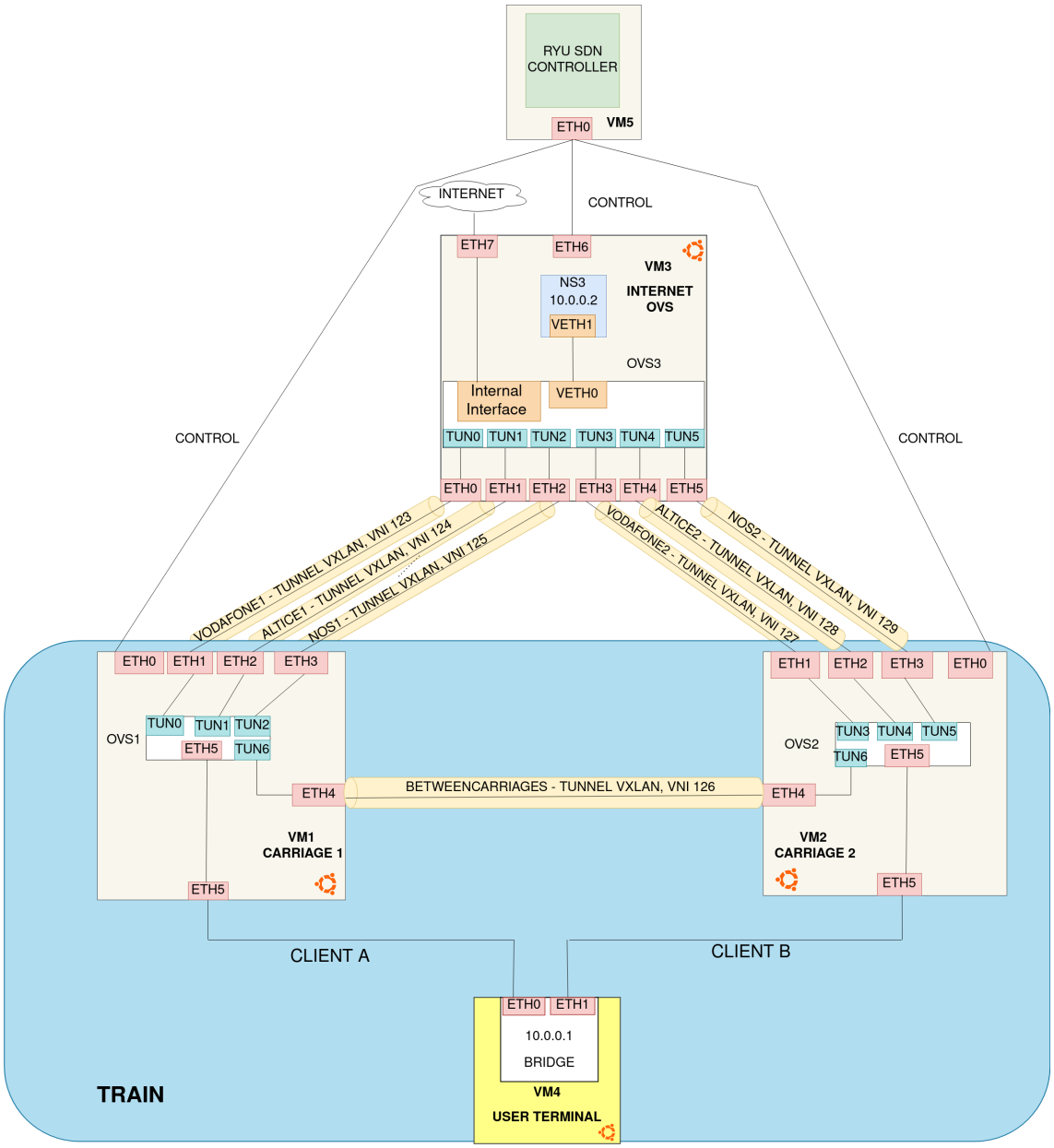

Figure 3.8: Testbed Architecture used for the TMVC implementation.

3.3.2 Bridge and Network Namespaces Configuration

As referred previously, to represent a situation where a client performs HO between two carriages, a bridge with two NICs was created. To do so, the package *bridge-utils* was installed inside VM4 Linux operating system. A very simple bash script was developed to create a bridge that logically joined *enp0s3* and *enp0s8* NIC into bridge *br0*. The *br0* bridge was configured with the IP address 10.0.0.1/24. In Figure 3.9 it is possible to see the bridge *br0* configuration. For the sake of simplicity, in Figure 3.8 enp0s3 is represented as *ETH0* and enp0s8 is represented as *ETH1*.

```
cliente@client:~$ sudo brctl show
bridge name     bridge id               STP enabled     interfaces
br0             8000.0800274f27f3       no              enp0s3
                                                        enp0s8
```

Figure 3.9: Bridge *br0* configuration.

One important detail to refer is the Maximum Transfer Unit (MTU) used. The default is 1500 bytes and when trying to sent a TCP flow to NS3 through the VXLAN tunnels, the attempt was unsuccessful.

This can be explained due to the overlay protocol (VXLAN) overhead. This overhead has the value of 50 bytes, as can be seen in Figure 2.14. Specific MTU configuration is necessary for VXLAN to function as expected. Therefore, *br0* MTU was set to 1450 bytes and the TCP flow problem was solved. In Figure 3.10 it is possible to see the MTU and IP address of bridge *br0*.

```
cliente@client:~$ sudo ifconfig br0
br0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1450
        inet 10.0.0.1  netmask 255.255.255.0  broadcast 10.0.0.255
        inet6 fe80::a00:27ff:fe4f:27f3  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:4f:27:f3  txqueuelen 1000  (Ethernet)
        RX packets 59236  bytes 4012334 (4.0 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 57699  bytes 1129451033 (1.1 GB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Figure 3.10: Bridge *br0* network configurations.

A network namespace (NS) was created inside VM3 with the purpose of testing connectivity between the user terminal and the Internet OVS. More precisely, a veth device pair is a virtual ethernet cable and it was developed using the command present in Figure 3.11.

```
vm3@vm3:~$ sudo ip link add name veth1 type veth peer name veth0
```

Figure 3.11: Veth pair creation.

After creating the veth pair, the network namespace should be created and one endpoint of the veth pair should be assigned to the NS3 and the other endpoint should be assigned as an OVS interface. In Figure 3.12, it is possible to visualize that *veth0* has been assigned to *ovsbr3* and in Figure 3.13 it is possible to see network configurations of the network namespace "isp". Despite being hosted inside VM3 and as referred previously, the "isp" network namespace has its own routing tables and TCP/IP stack properties.

The development of the network namespaces was also performed with a bash script.

```
vm3@vm3:~$ sudo ovs-vsctl show
83831813-853e-49ea-8682-01ee8ee08a42
    Manager "ptcp:6640"
    Bridge ovsbr3
        Controller "tcp:172.16.50.39:6633"
        fail_mode: secure
        Port ovsbr3
            Interface ovsbr3
                type: internal
        Port veth0
            Interface veth0
    ovs_version: "2.13.1"
```

Figure 3.12: Debugging command ovs-vsctl show output.

```
vm3@vm3:~$ sudo ip netns exec isp ifconfig
lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

veth1: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1450
        inet 10.0.0.2  netmask 255.255.255.0  broadcast 10.0.0.255
        ether d6:38:09:41:72:eb  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Figure 3.13: Network namespace "isp" properties.

3.3.3 OVS and RYU Configuration

This section is usefull to understand how the RYU framework can be used, how OVS bridges are created and how each OVS connects to the OVSDB manager and to the RYU SDN controller. The first procedure was to install the RYU framework [41] inside VM5. That can be done directly using *pip3* packet management system or from the source code, located in GitHub.

Therefore, is was necessary to install the *openvswitch-switch* package present in the Linux kernel. It was installed inside VM1, VM2 and VM3. After installing each OVS, a bridge was created, as can be visualized in Figure 3.14.

```
vm1@vm1:~$ sudo ovs-vsctl add-br ovsbr1
```

Figure 3.14: Bridge ovsbr1 creation inside VM1.

After the bridge creation, OF version, the SDN controller IP address, the controller mode and the set-manager mode should be defined. In the TMVC solution, OF version used was OF 1.3 due to the fact that this OF version contains all the necessary features to implement the intended

solution, such as support for overlay tunnels, extension for IPv6 headers, the table miss flow entry and a lot of documentation available.

The RYU SDN controller has two controller modes. The standalone mode and the secure mode. In the first, the OVS will take responsibility for forwarding the packets if the controller fails. In the second, only the controller is responsible for forwarding packets, and if the controller turns down state, all packets are dropped. The mode chosen for this work is the secure mode, so that the SDN controller can take full control of the system.

In order to connect with OVSDB to create VXLAN tunnels, the set manager mode must be enabled. OVSDB listens in port 6640. In Figure 3.15 it is possible to visualize the referred configurations.

```
vm3@vm3:~$ sudo ovs-vsctl set bridge ovsbr3 protocols=OpenFlow13
vm3@vm3:~$ sudo ovs-vsctl set-controller ovsbr3 tcp:172.16.50.39:6633
vm3@vm3:~$ sudo ovs-vsctl set-fail-mode ovsbr3 secure
vm3@vm3:~$ sudo ovs-vsctl set-manager ptcp:6640
```

Figure 3.15: OVS configurations used to connect to RYU SDN controller.

3.4 RYU Application Implementation

In this section, the RYU application developed to implement the system described in the TMVC solution is detailed and explored. It is divided in five parts:

1. How the SDN controller detects an OVS connecting to it.
2. How the VXLAN tunnels are created using the SDN controller.
3. How the ARP process is handled in order to avoid broadcast storms/loops between the multiple VXLAN tunnels.
4. How LB has been developed to distribute the traffic accordingly to specific weights assigned to each VXLAN tunnel.
5. How HO is performed between carriages with minimal Internet connection interruption and how the traffic from one carriage destined to the Internet is directed to through another one in cases where all the links from the first carriage go down.

3.4.1 OVS Entering in the Network

It is fundamental that the SDN controller detects that an OVS has connected or has disconnected from it. To do that, the event *OFPStateChange* is used with two negotiation phases envolved, the *MAIN_DISPATCHER* phase used for OVS connection and the *DEAD_DISPATCHER* phase used for OVS disconnection.

A dictionary was created to identify each OVS ID. This ID has exactly 16 characters and it is described as the datapath ID and can be visualized using the command present in Figure 3.16.

```
vm3@vm3:~$ sudo ovs-vsctl get Bridge ovsbr3 datapath-id
"000086a7a0f0234b"
```

Figure 3.16: Internet OVS ID.

The same ID can be retrieved by the RYU SDN controller application developed and it can be visualized in Figure 3.17.

```
EVENT ofp_event->overlaysolution EventOFPStateChange
openvswitch entered in the network with dpid: 000086a7a0f0234b
```

Figure 3.17: Internet OVS ID in RYU SDN App.

The flowchart that represents this procedure can be visualized in Figure 3.18.

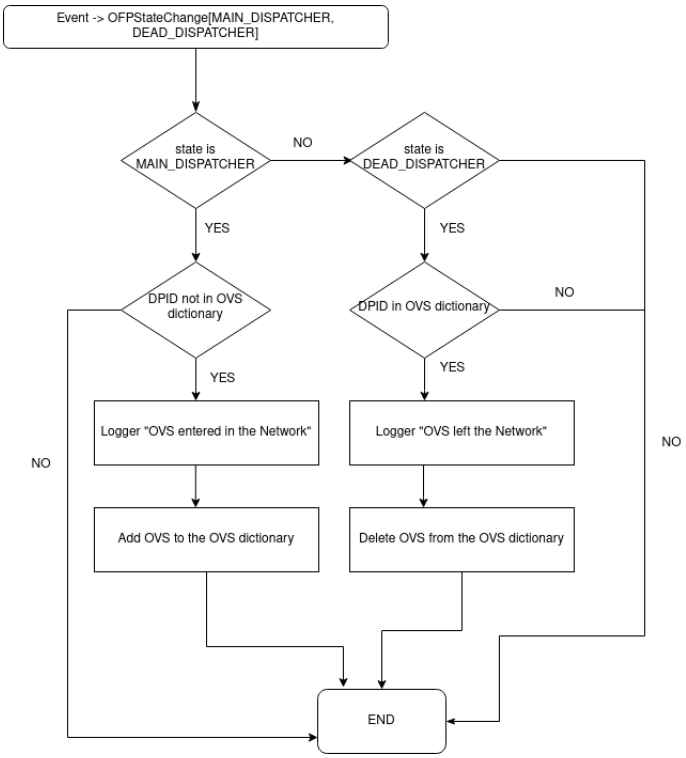

Figure 3.18: Flowchart that represents the entry of an OVS in the network.

3.4.2 VXLAN Tunnels Creation

To establish the overlay network, it is necessary to create VXLAN interfaces in each OVS. To do so, OVSDB Manager library was used, since this library permits to interact with devices speaking the OVSDB protocol and therefore the controller can perform remote management of the devices, as intended. RYU SDN framework contains the ovs library that permit to create VXLAN interfaces, defining the underlay remote IP address, the VNI and the OF port number. Therefore, three libraries were imported, as can be visualized in Figure 3.19. This functionality was developed based on a prototype developed by Aki Tuomi, available in GitHub [42].

```
from ryu.services.protocols.ovsdb import api as ovsdb
from ryu.services.protocols.ovsdb import event as ovsdb_event
from ryu.lib.ovs import bridge as ovs_bridge
```

Figure 3.19: OVSDB and OVS libraries.

The two main functions used in this section are functions *_get_ovs_bridge* and the *_add_vxlan_port*. This functions are executed also in the *OFPStateChange* event.

The first one is used to establish a TCP connection with each OVS. To do so, it uses the CONF instance [43], developed by the RYU SDN community. The flowchart of the referred function can be visualized in Figure 3.20.

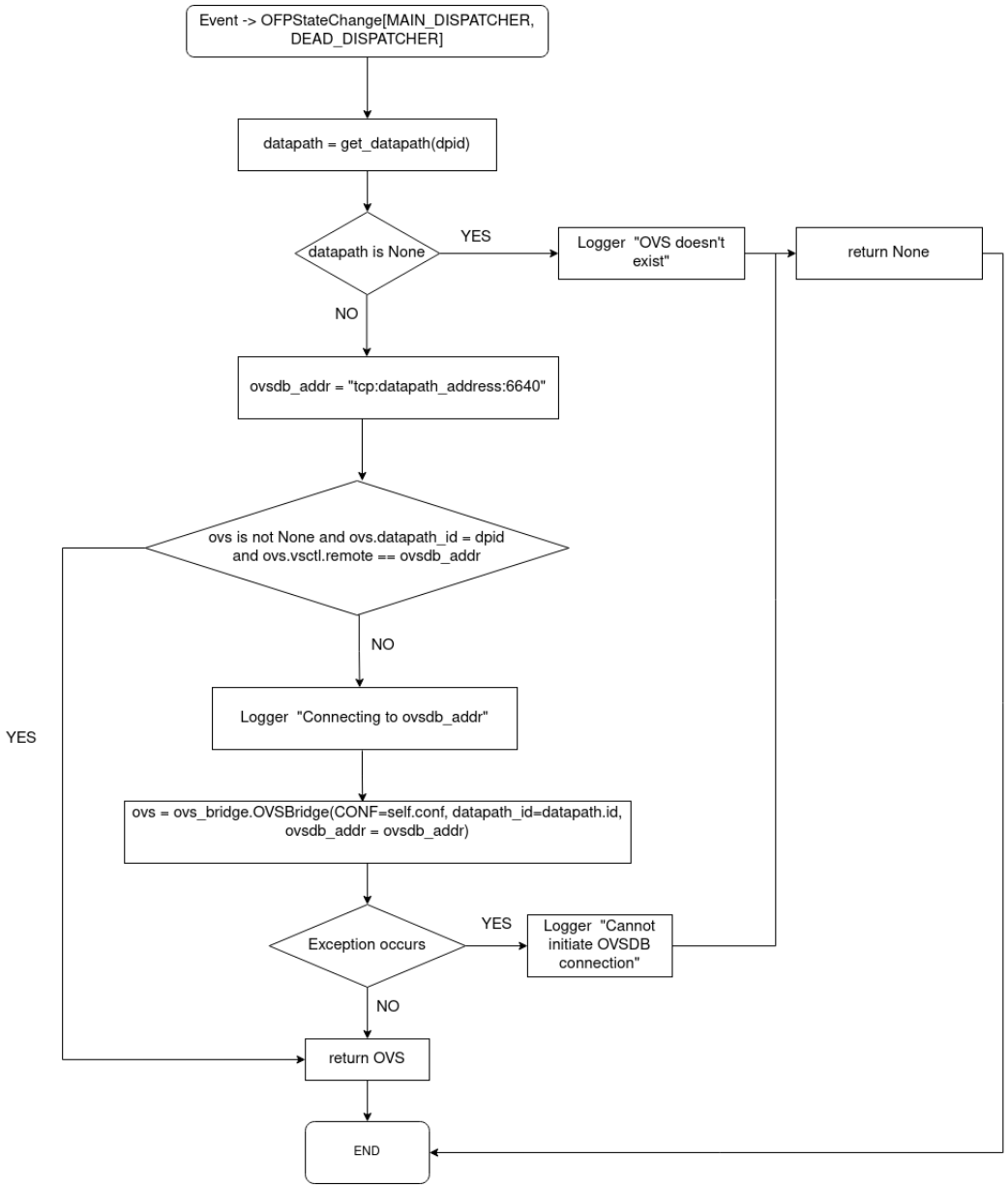

Figure 3.20: Function _get_ovs_bridge function flowchart.

The second function is used to create a VXLAN port inside an OVS. It uses the function *add_vxlan_port* [44], also developed by the RYU SDN community. This function receives as parameters the port name, the remote underlay IP address of the peer, the VXLAN VNI and the OF port number. The flowchart of this function can be visualized in Figure 3.21.

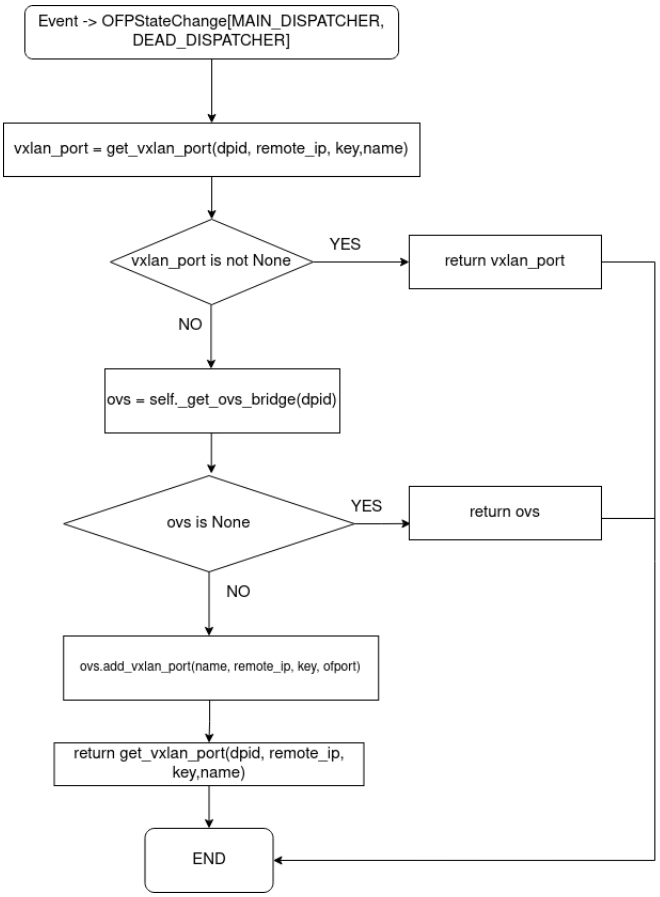

Figure 3.21: Function *_add_vxlan_port* function flowchart.

3.4.3 ARP Process

One of the major problems when developing multipath solutions in order to perform LB is the occurrence of loops in the network that might lead to broadcast storms and consequently to a network breakdown. In those situations, it is very usual to use the STP. However, STP process blocks specific ports and therefore LB cannot be executed.

For most TCP/IP network traffic, the first packet sent from a host is an ARP packet. The Ethernet destination of an ARP request is a broadcast address. Therefore, when testing the system architecture with each OVS in standalone mode (not blocking packets when the controller does not have intelligence developed for a specific kind of input), the ARP request sent by the User Terminal causes a broadcast storm and the overlay network fails immediately.

It is important to refer that this debugging mode (testing the overlay network in standalone mode) was very usefull during this work development, since it permitted to study the network behaviour and what type of intelligence should be developed to prevent specific situations.

After understanding the problem, a solution was designed in the RYU application to prevent broadcast storms, by using specific functions that guaranteed that the ARP request would go for only one of the four VXLAN interfaces available. Besides that, it was also created a mechanism

that permitted that the overlay network to continue working with minimal interruption in cases where the interface through which the ARP request travelled to go down. The solution developed was based in a prototype designed "Multipath forwarding" created by Li Cheng, available on GitHub [45].

The function *arp_forwarding* is used to verify if the ARP process has already been resolved. More precisely, if the output port for a specific Ethernet destination address has been resolved, a flow is installed on each OVS. If not, a flood message is sent all over the overlay network. The function *mac_learning* is a boolean function used to detect possible loop situations in the network. If the source MAC address of the User Terminal is associated with different in ports of an OVS, it returns False. Otherwise, it returns True and it associates the referred source MAC address with the OVS in port.

The flowchart representing both functions can be visualized in Figure 3.22.

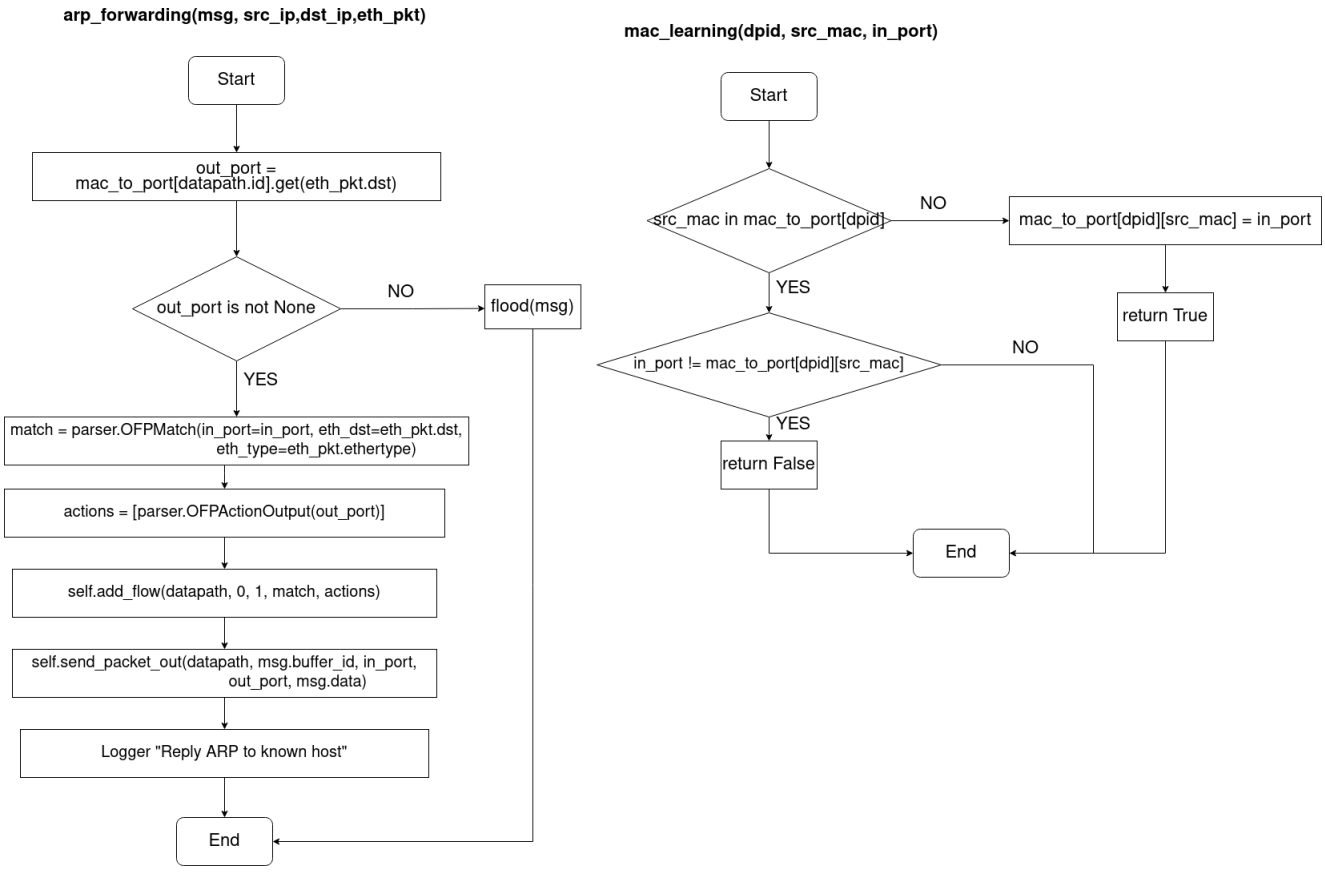

Figure 3.22: Functions *arp_forwarding* and *mac_learning* flowcharts.

The ARP process is accomplished in the *OFPPacketIn* event. As already referred in the present document, when an OVS does not have a flow installed to a specific situation, it contacts the controller and generates an *OFPPacketIn* Event. This is caused by the table-miss flow entry. The table-miss flow entry wildcards all match fields (all fields omitted) and has the lowest priority 0.

In Figure 3.23 it is possible to see the interaction between the ARP process and the *OFPPacketIn* event.

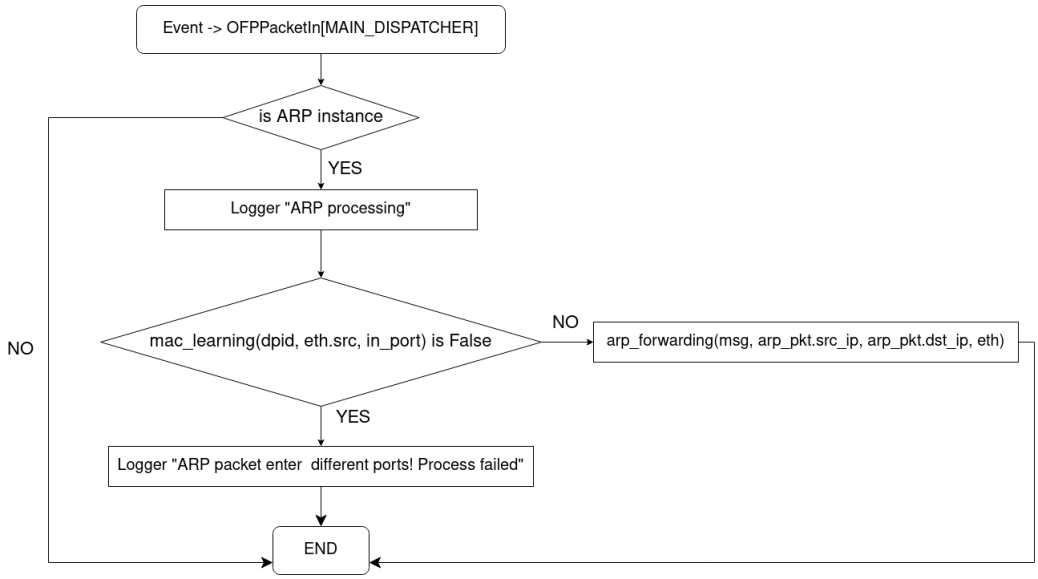

Figure 3.23: Event *OFPPacketIn* handling ARP process flowchart.

Finally, it was necessary to develop a mechanism that permitted that the overlay network continue functional when the interface through which the ARP request travels went from up to down state. To force an OVS interface to not receive or forward traffic, the management tool *ovs-ofctl* was used.

To detect that an OVS interface switched state, the event *OFPPortStatus* was used and when an interface stops receiving and forwarding traffic, the OVS flows associated with it are deleted, such as the OVS ID is deleted from the *mac_to_port* dictionary, making then possible to the system to accept new flows from different OVS ports.

The flowchart of this procedure can be visualized in Figure 3.24.

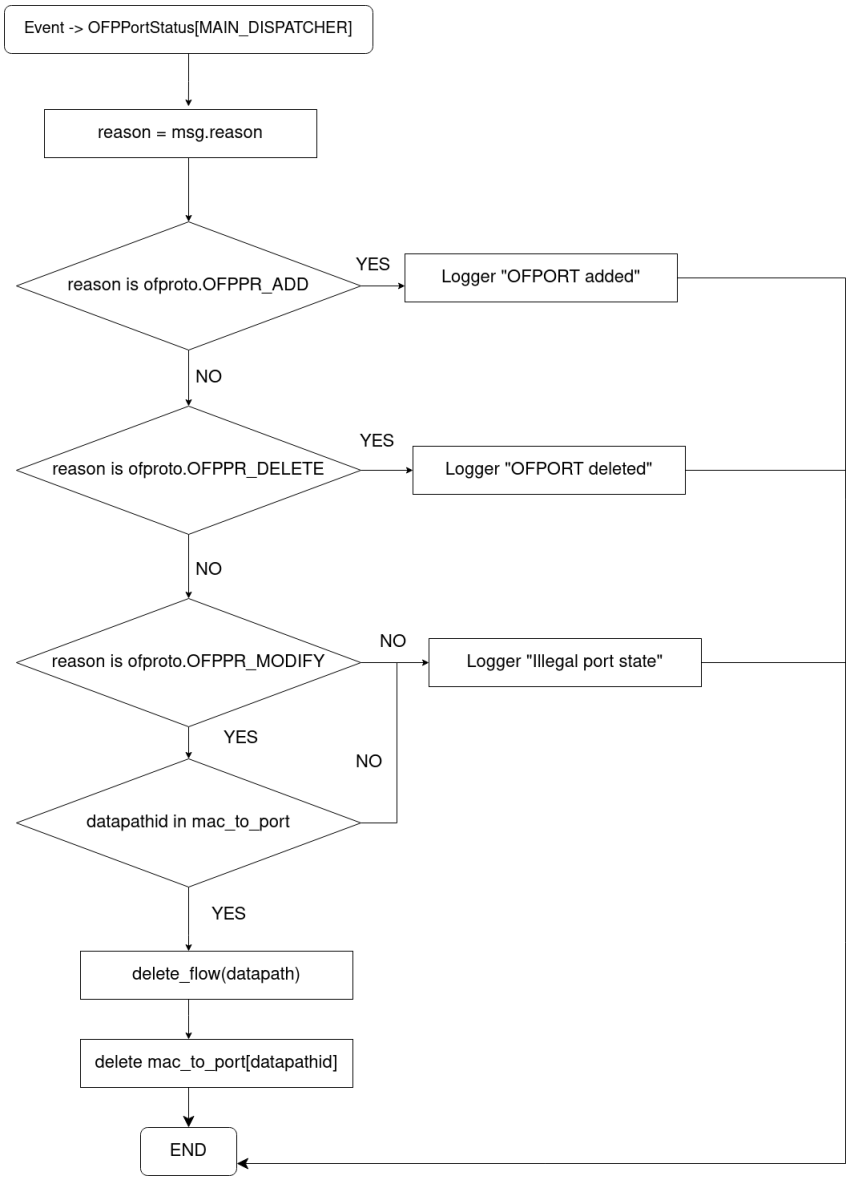

Figure 3.24: *OFPPortStatus* handler flowchart.

3.4.4 Load Balancing between VXLAN Tunnels

One key feature of the TMVC solution is the possibility of using multiple links, increasing the efficiency of the network. Multipath routing has been made easier, with a more programmable approach to computer networks with the rise of the SDN paradigm and the OF protocol. As explained in Section 2.2.4, the SELECT group table is the most used approach to develop multipath applications.

More precisely, buckets were created in a group action and a weight has been assigned to each bucket and each bucket is associated with a VXLAN port. Traffic can then be balanced between the three ISP/VXLAN links and bandwidth can be increased.

Despite looking a Weighted RR approach, it is not. Understanding how an OVS divides flows

among buckets in OF SELECT group type was one of the hardest tasks during the dissertation implementation development.

However, as explained in the OVS documentation [46], since version 2.4, OVS hashes the source and destination Ethernet address, VLAN ID, Ethernet type, IPv4/v6 source and destination address and protocol, and for TCP and SCTP only, the source and destination ports. In previous versions, OVS used only the destinaton Ethernet address to choose a bucket in a select group type.

Since in the TMVC solution, version 2.13.1 is the OVS version used, to select a bucket, OVS hashes flow data with the bucket ID and multiplies by the bucket weight to obtain a "score" and then selects the bucket with highest score. This means that the bucket selection is highly dependent on the headers hash and the bucket weight may not affect absolutely the bucket selection.

It is also important to refer that packets whose hashed fields are the same will always go to the same bucket. This means that if a single traffic flow is used to validate the balance algorithm, only one bucket will receive traffic. Besides that, hashing is probabilistic and as so, testing with a small number of flows may still lead to an uneven distribution.

Function *loadbalancing_weights* was created to define the VXLAN/OF ports through which each bucket is associated and also to define parameters such as each bucket weight, the watch port, the watch group and the group id. The weights are relative weights, meaning that if the weights were assigned in a 1/1/1 proportion for the three ISP links, a proportion of 2/2/2 would have exactly the same effect, since the OVS normalizes it.

The watch port and watch group are not mandatory when using SELECT group type. In fact, it is a parameter developed for the Fast-Failover group type but it is usefull since it permits another bucket to be chosen when a OF port switches state from up to down.

After the ARP process (explained in Section 3.4.3) is concluded, it was necessary to create intelligence to deal with IPv4 packets. When an IPv4 packet is received in the OVS port attached to the User Terminal, the function *loadbalancing_weights* is executed and a flow is added to the OVS. This flow is the flow with higher priority, since it is intended that the LB rules are always executed first. In Figure 3.25, it is possible to visualize the flowchart representing the LB rules being added to the OVS present in carriage 1, when an *EventOFPPacketIn* is received.

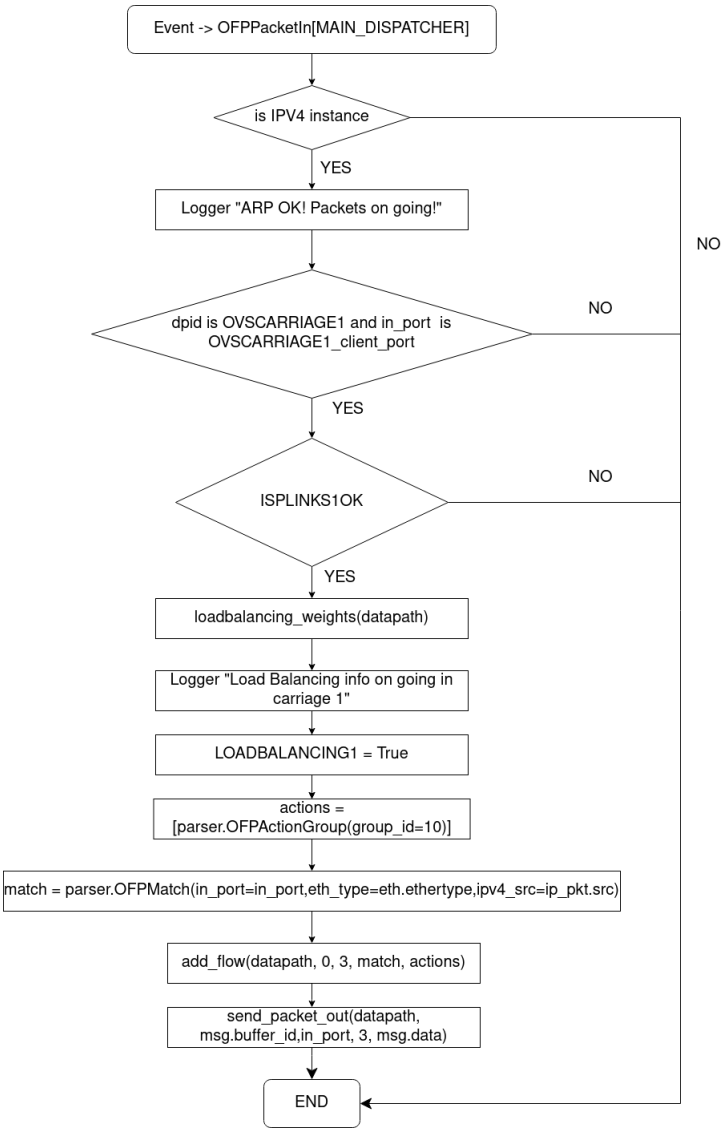

Figure 3.25: OVS Carriage 1 LB process.

3.4.5 HO Between Carriages

Another objective of the present dissertation consisted in the possibility of user movement between carriages with minimal Internet connection interruption. Since the TMVC solution was developed in a full virtualized environment due to the Covid-19 pandemic, it was necessary to create a scenario to test the HO from OVS1 to OVS2. As explained in Section 3.3, a network bridge with two interfaces was used and each one of them was connected to an OVS located in a different carriage. Using the bridge with only one of the two interfaces connected at a time, the HO scenario was successfully created and tested.

The first use case raised was how the controller could detect that an User Terminal interface turned down state and another turned up state, since the User Terminal interfaces were not directly attached to the OVS. Therefore, as almost in every steps along the present work, the first attempt

was to put all the OVSs in standalone mode and use *tcpdump* packet analyzer to visualize possible network events when the HO (from carriage 1 to carriage or vice-versa) was executed.

An event was observed, a Multicast Listener Discovery (MLD), a component of the IPv6 suite, detailed in RFC 2710 [47]. MLD is used by IPv6 routers for discovering multicast listeners on a directly attached link. More precisely, it was detected an MLDv2 report with a IPv6 link local multicast destination address (FF02::16). This type of message is an all mldv6 router. This message is mapped to 33:33:00:00:00:16, which is the Ethernet destination used by the SDN controller to detect an HO between carriages.

In Figure 3.26, it is possible to visualize the referred message after using *tcpdump* to listen port ETH5 of VM/Carriage 2.

```
08:04:05.713168 IP6 :: > ff02::16: HBH ICMP6, multicast listener report v2, 4 group record(s)
```

Figure 3.26: MLDv2 report message in carriage 2.

After understanding the event, it was necessary to create the intelligence that permitted the HO between carriages without Internet connection interruption, in the RYU Application. The most important condition in this procedure is the direction of the flows. More precisely, if the User Terminal accesses the Internet using OVS1, the Internet OVS (located in VM3) associates the MAC address of the User Terminal with OVS1. When the HO event occurs, it is necessary to remove previous flows not just from OVS1 but also from the Internet OVS flows, permitting the User Terminal to use the OVS2. The process is similar to the ARP process when the interface through which the ARP request travels goes down.

The *EventOFPPacketIn* is again used but in this case an IPv6 instance was explored. The IPv6 is another feature present in the packet library of the SDN RYU framework.

The flowchart of the HO process can be visualized in Figure 3.27.

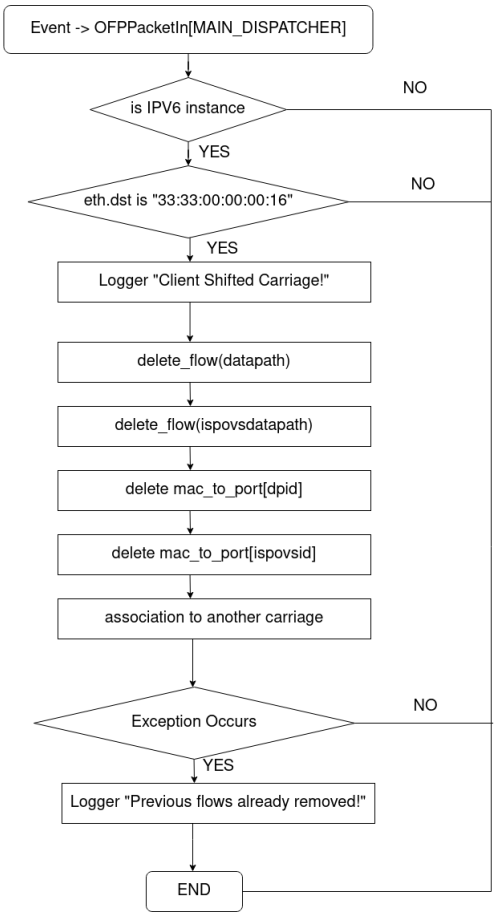

Figure 3.27: HO process flowchart.

3.4.6 Traffic Redirection

The last feature of the RYU application is the possibility of accessing to the Internet in a carriage using the ISP links of another carriage, in cases where all ISP links of the first are in down state.

To do so, it was created a VXLAN tunnel named "BETWEENCARRIAGES", through which the overlay network traffic travels when needed. The first attempt to create this functionality was to detect that a VXLAN link is not receiving neither forwarding traffic. When that happens in the present dissertation, it was defined that the specific link is in down state. That was achieved using the event *OFPPortDescStatsReply* that occurs when there is and *OFPPortDescStatsRequest*. It was also necessary to detect if LB rules had already been applied to a specific OVS to prevent OF Group errors that may occur when rules are duplicated.

Such as in the ARP process, it was created a bash script that created a scenario where all ISP links stop receiving and forwarding traffic, using again the ovs-ofctl management tool. When a link is in down state, a *OFPPortDescStatsRequest* is generated.

The *OFPPortDescStatsReply* generates a message with OVS port statistics, such as OF ports number, MAC address of a specific port, ports speed, (ports name and ports configuration). The port configuration represents the state of the port, more precisely, it consists of a value that varies

based on if the port is only forwarding traffic or if it is forwarding and receiving traffic. This value is a bitmap of port configuration flags.

In Figure 3.28 it is possible to visualized how OFP *OFPPortDescStatsRequest* and *OFPPortDescStatsReply* interact with each other. In this flowchart it is represented the situation where ISP links from carriage 1 go down state. The same process was applied for carriage 2.

Figure 3.28: OFP *OFPPortDescStatsRequest* and *OFPPortDescStatsReply* interaction flowchart.

When the controller detects that all ISP links are down, it installs a flow in the referred OVS so that traffic can go out through other carriage using the BETWEENCARRIAGES VXLAN tunnel. If load balance rules had not been applied yet, it also applies those rules in the OVS with ISP links available.

In Figure 3.29 it is possible to see the continuation of the process represented in Figure 3.28 (also for when all ISP links go DOWN state in carriage 1). When all ISP links are down in a

carriage, it is necessary to remove OVS previous OVS flows associated with those links and then a new flow can be created, redirecting the traffic to the VXLAN tunnel created between carriages.

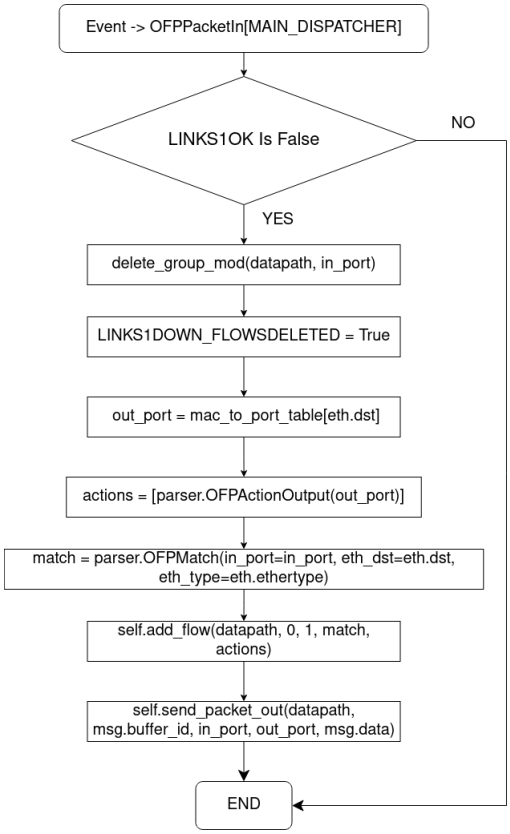

Figure 3.29: Traffic rediretion flowchart.

In summary, in this chapter the TMVC solution was detailed and each component of the system was explored. Firstly, it was presented an overview of the system model to justify the interaction between each component of the solution. After that, the RYU SDN framework and its application programming model were explained in order to provide a faster comprehension of the RYU application implemented. In the third place, the testbed architecture developed to test and evaluate the TMVC solution was explored in depth. The last presented component of the system was the RYU application implementation where a flowchart of each functionality of the TMVC is exposed.

Chapter 4

Evaluation of the TMVC Solution

This chapter is focused on the evaluation of the TMVC solution. It includes the results of multiple tests designed to assess the correct operation of the solution and it is divided in two components:

- **Internet connectivity tests**: Based on ICMP packets directed to Google's public DNS server (8.8.8.8), dynamic changes in the conditions of the network were performed to test the TMVC functionalities and related reconfiguration times to re-establish Internet connectivity.
- **LB tests**: Based on the traffic generator Iperf3, the distribution of network flows through each ISP link has been tested to validate the proper operation of the implemented LB system, part of the TMVC solution.

4.1 Experimental Setup

This section presents the experimental setup and serves as a configuration manual of the testbed used. The testing scenario is the same shown in Figure 3.8. However, a more detailed setup can be visualized in Figure 4.1, where all the underlay networks configured to represent the ISP links, the control links (through which the OF rules from the Ryu SDN controller are transmitted to the OVSs) and the link that interconnects carriages 1 and 2 are depicted. Furthermore, the overlay LAN which the train passengers connect to is also represented and configured in the 10.0.0.0/24 subnet. The internal interface of the Internet OVS (located in VM3) is configured with the IP 10.0.0.254/24 and it serves as the default gateway of the overlay network.

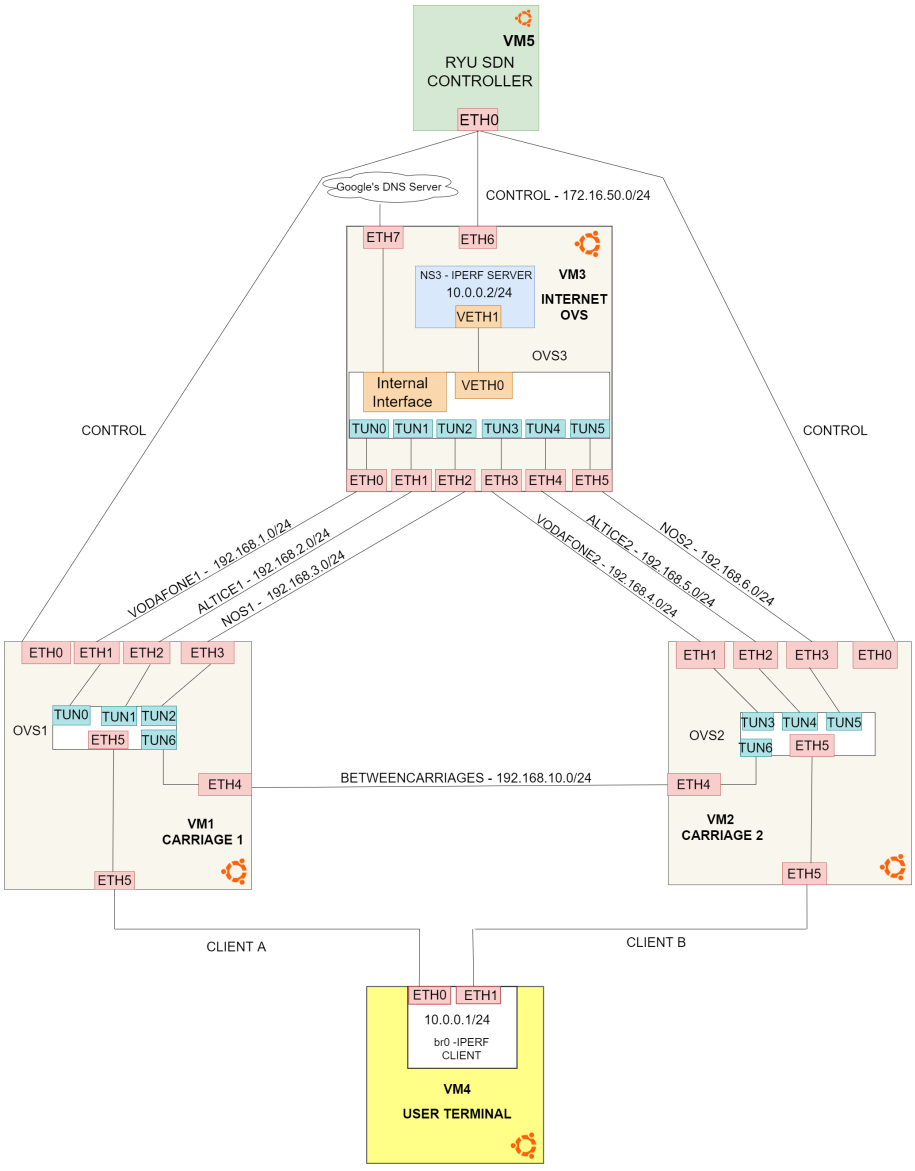

Figure 4.1: Experimental scenario setup.

All the underlay networks are configured using the virtual LAN segments instantiated by the "Internal Network" functionality of Oracle Virtual Box. In Figure 4.2, it is possible to visualize the Vodafone1 underlay connection between carriage 1 (VM1) and VM3, where the Internet OVS is located.

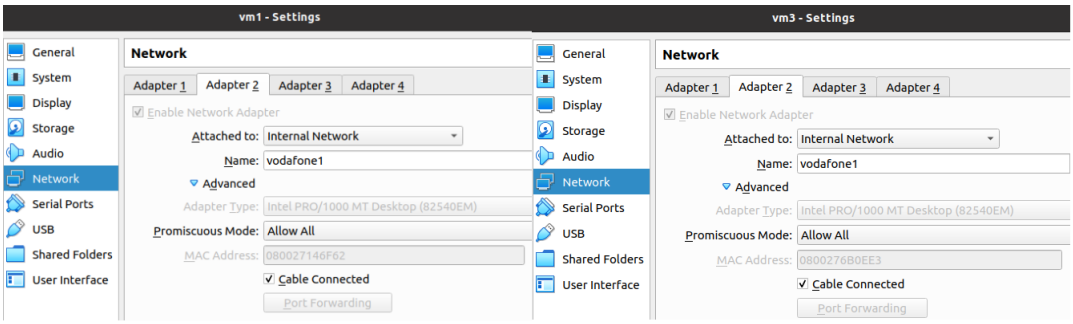

Figure 4.2: Internal network created to represent Vodafone1 ISP link.

In order to achieve the Internet, Linux *iptables* were configured to route packets coming from the internal interface *ovsbr3* (represented as *Internal Interface* in Figure 3.8) to the NAT interface *enp0s19* (represented as *ETH7* in the same figure) and vice-versa. The last command line is used to change the source IP address on packets going out to the Internet. The iptables rules configured can be visualized in Figure 4.3.

```
echo 1 > /proc/sys/net/ipv4/ip_forward
iptables -A FORWARD -i ovsbr3 -o enp0s19 -j ACCEPT
iptables -A FORWARD -i enp0s19 -o ovsbr3 -m state --state ESTABLISHED,RELATED \
-j ACCEPT
iptables -t nat -A POSTROUTING -o enp0s19 -j MASQUERADE
```

Figure 4.3: Configuration of iptables inside VM3 to allow Internet access.

In order to test the LB system developed in the present work, Iperf3 was used, which is a tool created to test network speeds, throughput and other network measurements. It generates TCP or UDP flows between a client and a server. In the present test scenario, bridge *br0* is the client and the NS3 located inside VM3 is the server.

4.2 Internet Connectivity Tests

The objective was to test how the TMVC solution handles different scenarios that can happen in a real environment, which can potentially affect the connectivity between a passenger's terminal and the Internet, such as user movement between carriages or changes in the state of V2I links. For that, we created 5 different scenarios, presented below, to evaluate the amount of time that the system needs to react until the connectivity to the Internet is re-established from the perspective of a passengers' terminal. For each scenario a single bash script is executed in the host machine, which then remotely accesses each of the VMs via Secure Shell (SSH) to configure and execute the necessary commands to setup the scenario conditions and to run the experiments.

More precisely, a ping command is performed from the bridge *br0* to Google's primary DNS server (8.8.8.8), representing an Internet flow while the TMVC reacts to the following connectivity scenarios simulated:

- HO from carriage 1 to carriage 2 and vice-versa.
- The ISP link through which the ARP packet travels turns down state, forcing the system to react and change the OVS flow rules.
- All the ISP links are in down state (including the BETWEENCARRIAGES VXLAN tunnel) and one of the ISP links turns up state.
- All the ISP links are in down state in one carriage and the traffic achieves the Internet through the other carriage (using the BETWEENCARRIAGES VXLAN tunnel).
- After the previous referred situation (where the BETWEENCARRIAGES VXLAN tunnel is used), the three ISP links recover to up state and the traffic follows its original path.

4.2.1 HO Between Carriages

The main purpose of this test was to understand the amount of time that the SDN controller takes to recognize and react to the HO from the carriage 1 to carriage 2 and vice-versa.

More precisely, bridge *br0* has two interfaces and each one is connected to one carriage (interface *enp0s3* is connected to carriage 1 and interface *enp0s8* is connected to carriage 2). Again, when comparing with the scenario represented in Figure 3.8, *enp0s3* is *ETH0* and *enp0s8* is *ETH1*. Bridge *br0* is connected to only one carriage at a time.

A timestamp was inserted in each Ryu *OFPPacketIn* event to retrieve measurements.

Regarding the HO from carriage 1 to carriage 2, the time measured is the time since the Ryu controler detects that the link between *br0* and carriage 1 (VM1) is down and the time that the Ryu controller detects that the ICMP packet achieves the Internet through carriage 2 (VM2) ISP links. A graphical demonstration of the HO from carriage 1 to carriage 2 is presented in Figure 4.4. In this process it was necessary to remove previous flows (from VM1) in the OVS Internet and to install LB flows in VM2.

Twenty HO tests were executed and an average of 0.78 ± 0.08 seconds was achieved. This is the value obtained for a confidence interval of 90%. Hereafter, we use this notation to refer to the confidence interval of 90%.

The same process was executed from carriage 2 to carriage 1 and an average of 0.83 ± 0.24 seconds was achieved. This value is very similar to the previous scenario (HO from carriage 1 to carriage 2) because the process is identical.

Based on the objectives established in the beginning of this work, the system reacts to an HO between two carriages with a delay under 1 second, while being transparent from the passenger's terminal point-of-view (no network configuration is needed and its WAN IP remains the same), as intended.

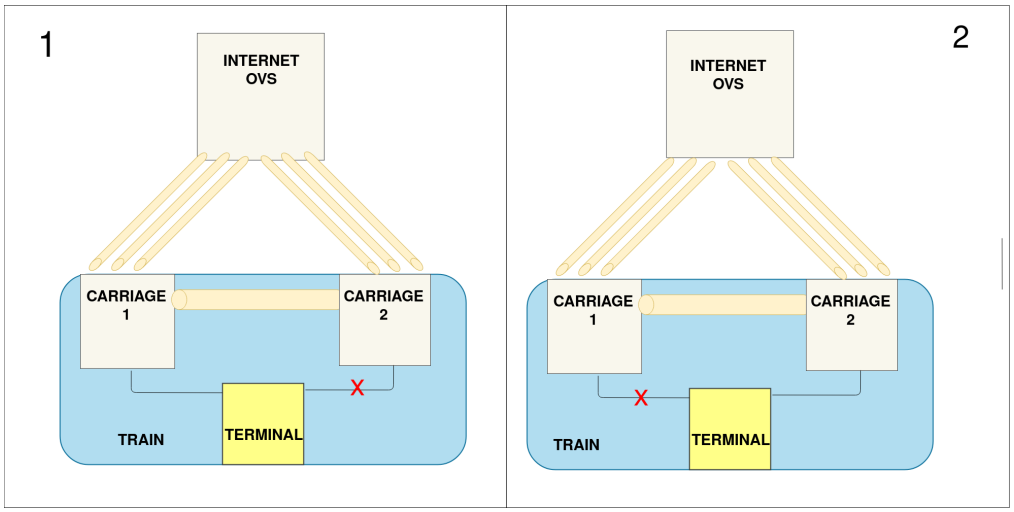

Figure 4.4: Graphical representation of the scenario of the HO from carriage 1 to carriage 2.

4.2.2 ISP Link Down

This test is used to determine the amount of time that the TMVC solution takes to react when the ISP link through which the ARP packet travels turns into down state, forcing the system to remove the previous flows and adding a new one, as represented in Figure 4.5.

When performing a ping command destined to Google's DNS servers, bridge *br0* needs to register the MAC address of the default gateway (the internal interface of the Internet OVS) in its ARP table. When generating the ARP request (and consequently a broadcast packet), the SDN controller creates a rule so that the ARP packet follows always through the same link, eliminating any broadcast storm possibility. This link is variable but is always the first interface to receive the broadcast packet. The logic implemented in the RYU SDN controller restricts the ARP packet to go through more than one interface.

The time measured is the time since the SDN controller detects that the ISP link through which the ARP packet travels is in down state and the time that the SDN controller reacts and detects

that the Internet connection is recovered and the ARP flow has been reinstalled through another VXLAN ISP interface.

In a first set of tests, an average of 14.03 ± 4.61 seconds was achieved, as the result of twenty runs. Since each run presented very different time results (in some runs the delay was 1 second while in the following run the delay was almost 30 seconds), some investigation was needed to understand this behavior. After some debugging, it was concluded that the system needed a trigger to detect that an interface stopped receiving/forwarding traffic and that trigger is an ARP packet. Therefore, when clearing the ARP cache of the user terminal, the system reacted immediately because the consequence of deleting the ARP cache is the generation of a new ARP request.

More precisely, a bash script was created (in the host machine) that turns the link through which the ARP packet travels into down state and after that, it deletes the ARP cache from the user terminal. After twenty runs, an average delay of 0.69 ± 0.22 seconds was achieved.

This solution should be improved since the system should never affect the client's terminal; this is left for future work. However, with this approach, the average delay is under 1 second as well.

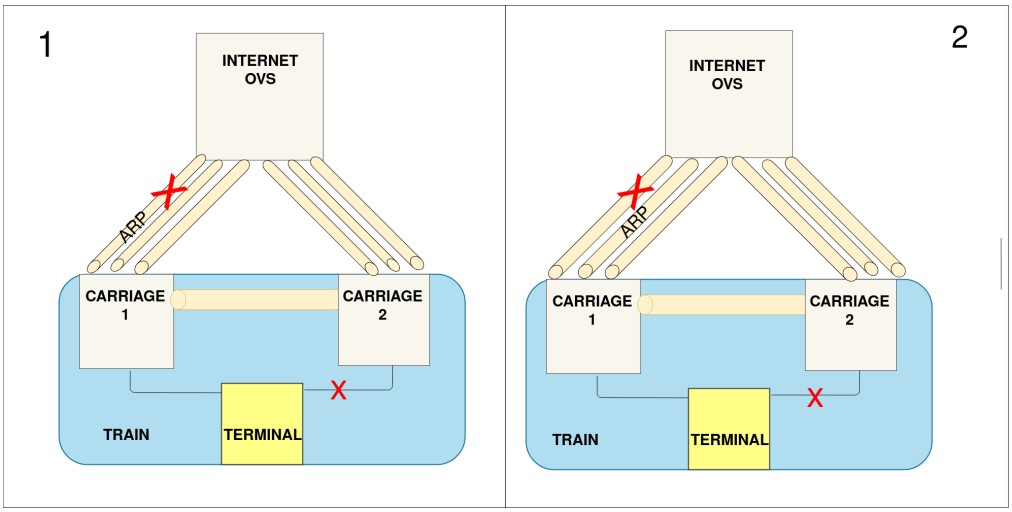

Figure 4.5: Graphical representation of the scenario with the ARP link down.

4.2.3 One ISP Link Up

This test aimed at checking how the system reacts in a situation where all the links are in down state (including the BETWEENCARRIAGES VXLAN tunnel) and one of the ISP links turns into up state, as can be visualized in Figure 4.6. This is an extremely unlikely situation to happen in a real environment but it is usefull to understand how the TMVC solution overtakes such process.

Just like in all the performed Internet connectivity tests, a timestamp was added to each Ryu *OFPPacketIn* event. The time measured in this test is the time since the SDN controller detects that one of the ISP links is in up state and the time the SDN controller detects that the ICMP packet reached Google's DNS server successfully.

The test was created using a bash script that turned one of the three ISP links into up state (assuming obviously that the three ISP links were originally in the down state). In the first approach, an average of 15.96 ± 4.18 seconds was achieved, being the result of twenty runs.

In the second approach, when one of the three ISP links turns into up state and afterwards the ARP cache of the user terminal (bridge *br0*) is cleared, the average delay achieved has the value of 0.03 ± 0.01 seconds, after a total of twenty runs. This shows the system is capable of reacting almost instantaneously.

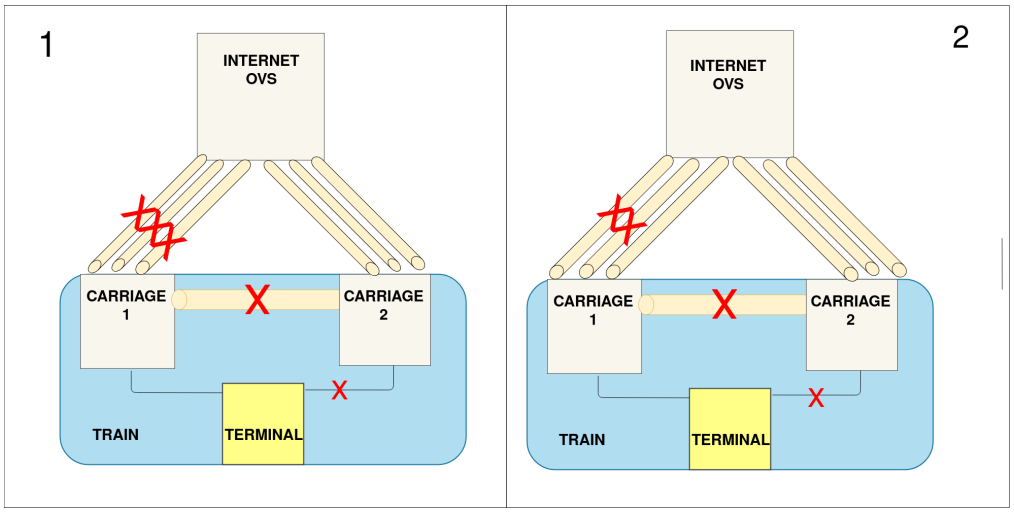

Figure 4.6: Graphical representation of the scenario with one ISP link up.

4.2.4 All ISP Links Down

This test aimed to evaluate the system in a scenario where all the three ISP links in one carriage turn into down state, forcing the SDN controller to install a flow that redirects all the traffic through the BETWEENCARRIAGES VXLAN tunnel, as can be seen in Figure 4.7.

The time measured is the time since the SDN controller detects that all the ISP links of one carriage turn into down state and the time that the SDN controller detects that the ICMP packet reached Google's DNS server sucessfully.

It is important to refer that in a situation where all the ISP links in carriage 1 turn into down state and the OVS present in carriage 2 (OVS2) has no load balance (OF tables) configured, there is no impact on the system behavior and the load balance rules for the OVS2 are still configured.

Each OF flow rule has a priority and that is relevant in this test and in the next one. The traffic redirection through the BETWEENCARRIAGES VXLAN tunnel has lower priority than the traffic direction through the ISP links. This is important because when one or the three ISP links turns again into up state, the system should send again the traffic through the ISP link.

In the first approach, an average of 12.92 ± 5.39 seconds was achieved, as the result of twenty runs.

In the second approach, when the three ISP links turns into down state and afterwards the ARP cache of the user terminal (bridge *br0*) is cleared, the average delay achieved has the value of 0.09± 0.08 seconds, after a total of twenty runs.

This is an important test since it evaluates the amount of the time that the system takes to react and the effective use of the tunnel created to send traffic through another carriage. The results show the average delay is low, confirming that the system is capable of reconfiguring almost instantaneously.

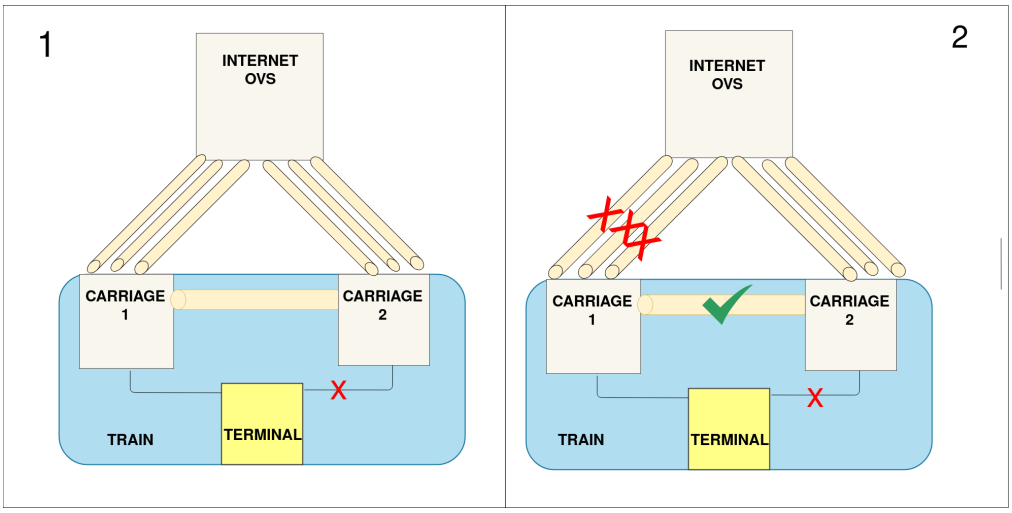

Figure 4.7: Graphical representation of the scenario with all ISP links down and traffic going through BETWEENCARRIAGES link.

4.2.5 All ISP Links Recovery

This test can be considered a continuation of the previous one since it checks the amount of time the system to reconfigure in a situation when all the ISP links turn into down state, the traffic is redirected through the BETWEENCARRIAGES VXLAN tunnel and all the three ISP links recover to UP state, as it can be visualized in Figure 4.8.

It measures the time since the SDN controller detects that the ISP links turn into up state and that the SDN controller detects the ICMP message has achieved its destination (it is the same as all previous tests).

In the first approach, an average of 26.46 ± 4.11 seconds was achieved, as the result of twenty runs.

In the second approach, when the three ISP links turns into up state and afterwards the ARP cache of the user terminal (bridge *br0*) is cleared, the average delay achieved has the value of 1.05± 0.46 seconds, after a total of twenty runs.

In Table 4.1 it is possible to visualize the average delay with a 90% confidence interval for each of the tests previously presented (with and without the user terminal ARP cache cleared).

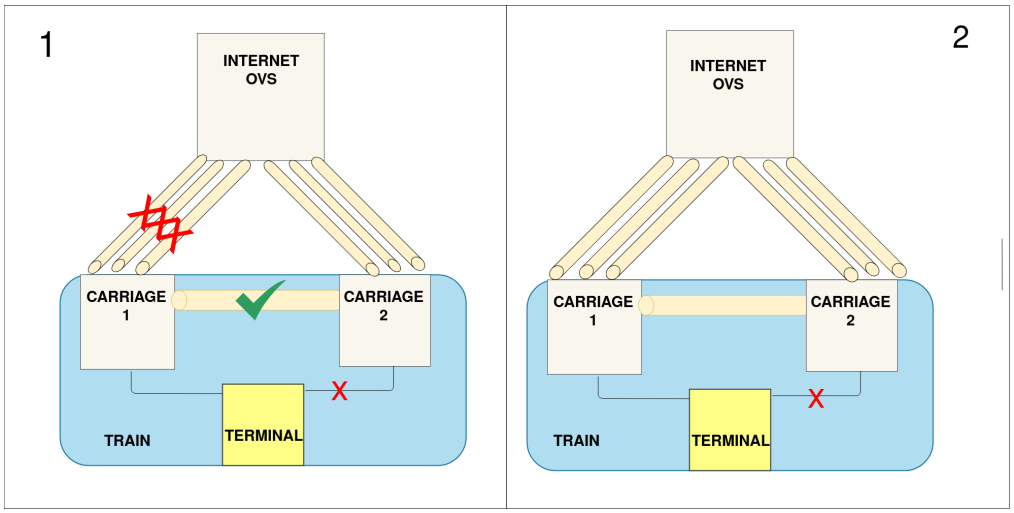

Figure 4.8: Graphical representation of the scenario with the recovery of all ISP links from carriage 1.

Scenario	Average Delay (s)	Cache Cleared Average Delay (s)
HO C1 → C2	0.78 ± 0.08	0.78 ± 0.08
HO C2 → C1	0.83 ± 0.24	0.83 ± 0.24
ISP Link Down	14.03 ± 4.61	0.69 ± 0.22
One ISP Link Up	15.96 ± 4.18	0.03 ± 0.01
All ISP Links Down	12.92 ± 5.39	0.09 ± 0.08
All ISP Links Recovery	26.46 ± 4.11	1.05 ± 0.46

Table 4.1: Average delay for Internet connectivity re-establishment with and without clearing user's terminal ARP cache.

4.3 Load Balancing Tests

In this section it is validated how the TMVC solution distributes traffic between the three ISP VXLAN links. As previously explained, through the use of OF group tables in SELECT mode, different weights can be assigned to each OVS interface. The tool used to generate TCP traffic is Iperf3. The Iperf3 client is the bridge *br0* and the Iperf3 server is the NS3 located in VM3.

More precisely, the following scenarios are tested:

- **Equal Weight Distribution**: VXLAN links Vodafone1, NOS1 and Altice1 will be assigned the same weight and three TCP flows will be launched with one second of difference between each. The three TCP flows are used to simulate three clients.
- **Unequal Weight Distribution**: VXLAN links Vodafone1, NOS1 and Altice1 will be assigned different weights and two different set of weights will be tested using a different number of flows.

4.3.1 Equal Weight Distribution Between ISP Links

The main purpose of this test is to confirm whether the traffic is equally distributed when the weights assigned to each link are the same. A weight of 1/1/1 was assigned to each link. When the Iperf3 flow is generated by the bridge *br0*, an OF group table is installed (can be in carriage 1 or carriage 2) by the RYU SDN controller and each bucket (that corresponds to a VXLAN interface) is assigned the referred weights.

In Figure 4.9 it can be visualized a bash script (executed by the user terminal, the bridge *br0*) that generates three TCP flows with a bitrate of 10 Mbit/s each. A delay of 1 second between each Iperf3 request is applied in order to not have perfectly overlapping throughput curves when later analysing the throughput of each ISP interface over the duration of the experiments. traffic distribution. Each Iperf3 TCP flow has a duration of 30 seconds.

```
#!/bin/bash

iperf3 -c 10.0.0.2 -b 10M -t 30 -p 2000  &
sleep 1
iperf3 -c 10.0.0.2 -b 10M -t 30 -p 2001 &
sleep 1
iperf3 -c 10.0.0.2 -b 10M -t 30 -p 2002
```

Figure 4.9: Bash script to generate three TCP flows with a second of delay between each.

In Figure 4.10 it is possible to visualize the amount of bytes that each VXLAN interface of carriage 1 transmits per second. 10 Mbit corresponds to 1.25 MBytes and that is the value that each interface transmits in almost every second, confirming the correct system behavior. There are some throughput peaks of approximately 2 MBytes that can be related with the Iperf3 traffic generator because these peaks occur at the same time for the three interfaces. Since this implementation is fully virtualized, the option of saturing the links is not viable because the solution is limited to the performance of the CPU used in the host machine.

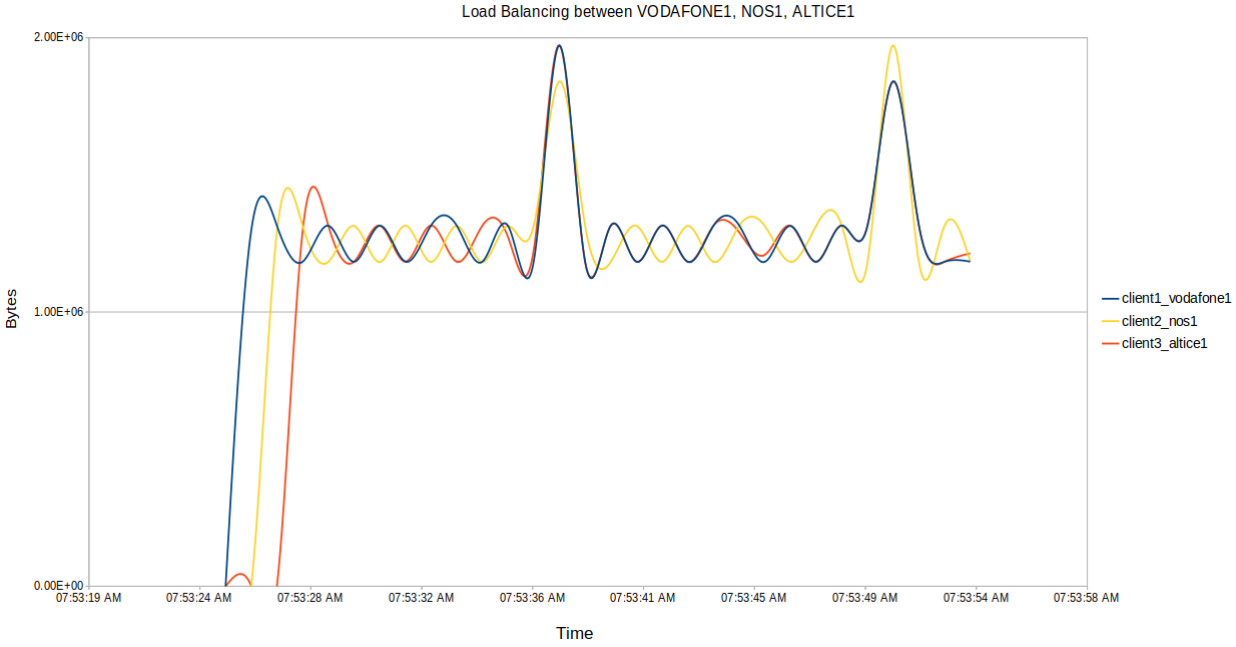

Figure 4.10: Throughput achieved for each VXLAN interface in an equal weight system.

4.3.2 Unequal Weight Distribution Between ISP Links

This is the second test of the LB system, part of the TMVC solution, with the purpose of verifying how the system distributes network flows when the interface weights are different. A distribution of 1/1/0 was assigned to VXLAN links Vodafone1, NOS1 and Altice1, respectively. This means that no traffic should travel through Altice1 link.

As can be visualized in Figure 4.11 and similarly with the previous test, a bash script was created to generate three TCP flows and the third flow starts with a delay of 10 seconds when comparing with the first.

Since one of the links does not receive or forward traffic and there are three flows, the traffic is duplicated in one of the active links.

```
#!/bin/bash

iperf3 -c 10.0.0.2 -b 10M -t 30 -p 2000  &
sleep 1
iperf3 -c 10.0.0.2 -b 10M -t 30 -p 2001 &
sleep 10
iperf3 -c 10.0.0.2 -b 10M -t 30 -p 2002
```

Figure 4.11: Bash script that generates three TCP flows with a second of delay between the first two flows, and 10 seconds of delay between the second and third flow.

Analysing Figure 4.12 it is possible to conclude that after 10 seconds from the moment when the VXLAN link NOS1 started receiving traffic, the same link receives the third TCP flow, duplicating amount of traffic received. After 30 seconds (when the second TCP stream finishes), the throughput measured in NOS1 link reduces since only the third flow is still on going.

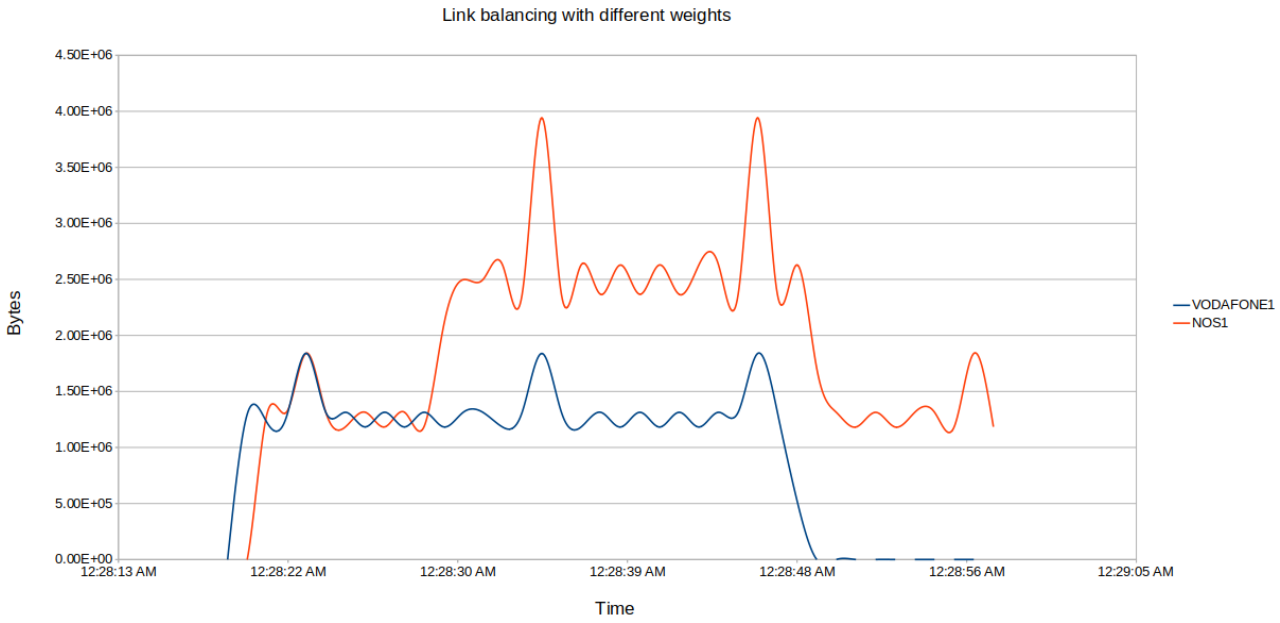

Figure 4.12: Throughput achieved for each VXLAN interface in an unequal weight system.

The third and last test of the present chapter is used to understand how the system divides network flows per each ISP link when the weights assigned to each link are significantly different. The more heterogeneous the related weights are, more flows are needed to achieve a more precise division of the traffic.

A weight of 75 was assigned to Vodafone1 link, 25 to NOS1 link and 0 to Altice1 link. In Figure 4.13, it is possible to see the Iperf3 client command executed from VM4 to generate 4 TCP flows in parallel with the previous bitrate established of 10 Mbit/s.

```
cliente@client:~$ sudo iperf3 -c 10.0.0.2 -b 10M -P 4
```

Figure 4.13: Iperf3 client side request with 4 parallel TCP flows.

In Figure 4.14 it is possible to visualize the throughput of each link (achieved with the refereed weight assignment distribution). However, since the weights are more heterogeneously distributed, it is necessary to confirm that the system can fairly distribute traffic when only 4 TCP flows were sent. As intended, the throughput value of link Vodafone1 is the triple of the throughput value of link NOS1. Since each flow is executed with the same bitrate, it can be concluded that 3 TCP flows travel through link Vodafone1 and the other TCP flow travel through link NOS1. In Figure 4.15 it is possible to see the output of command *ovs-ofctl -Oopenflow13 dump-ports ovsbr1* that shows the amount of traffic transmitted by each OF ports of the OVS present in carriage 1. Port 3 (Vodafone1) has transmitted to the Internet OVS a total of 37.90 MBytes and port4 (NOS1) has transmitted a total of 12.64 MBytes. This corresponds to a proportion of 75,00% for Vodafone1 link and 24,99% for NOS1 link, as intended.

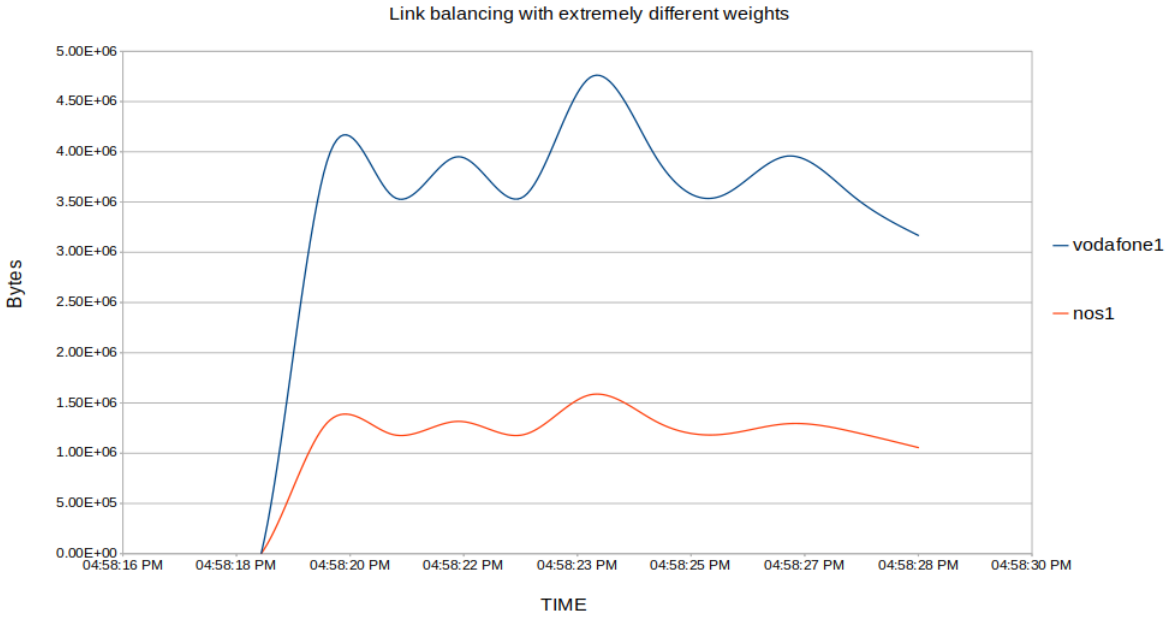

Figure 4.14: Throughput achieved for each VXLAN interface considering interfaces with more heterogeneous weights, with a ratio of 3:1:0.

```
port  4: rx pkts=2005, bytes=133414, drop=?, errs=?, frame=?, over=?, crc=?
         tx pkts=1106, bytes=12635311, drop=?, errs=?, coll=?
         duration=70.602s
port  3: rx pkts=1, bytes=60, drop=?, errs=?, frame=?, over=?, crc=?
         tx pkts=3265, bytes=37903265, drop=?, errs=?, coll=?
         duration=82.158s
```

Figure 4.15: Output of the command *ovs-ofctl -Oopenflow13 dump-ports ovsbr1*.

4.4 Discussion and Future Work

The evaluation of the TMVC solution allowed to test the overall performance of the designed system. The context that motivates this dissertation (Vehicular networks in general and on-board Internet for train passengers in particular) originates a group of use cases or different connectivity scenarios, such as a link turning into down state, forcing the network to reconfigure itself immediately.

On the other hand and as reported in the SoA, nowadays the SDN paradigm is emerging and it is advantageous to understand how the paradigm would adapt to different networking scenarios.

Therefore, five network scenarios where tested to validate the TMVC solution considering the fast (and transparent) re-establishment of Internet connectivity. The LB feature of the TMVC solution was also evaluated. Different weights were assigned to each ISP link and the system revealed to work properly since the network flows were correctly distributed between VXLAN tunnels. Another conclusion is that the more different the weights between VXLAN interfaces are, the higher the number of flows needed to achieve a proper flow distribution. Finally, the use

of OF group tables to perform traffic distribution in an SDN architecture affirmed as a very reliable option and can be used in a real environment.

As future work, it is proposed that the system does not need the ARP cache from the client side to be cleared because accessing or changing the user terminals in any way is not an option when developing a final solution. Also, the experiments that were performed were based on a fully virtualized environment, due to the limitations imposed by the COVID-19 pandemic. It would be interesting to evaluate the TMVC solution in a real vehicular networking scenario in the future. Nevertheless, after clearing the ARP cache from the client side, the results in terms of average delay were very satisfactory and almost all of the experiments resulted in an under 1 second average delay value, which confirms the SDN paradigm as a valid option when developing network architectures for overlay networking solutions, namely when considering transparent multi-homed vehicular communications.

Also, as future work, the LB tool could be dynamic and weights could vary according to predictive network QoS measurements (a train follows always the same rail and some network conditions can be predicted). Besides that, if a flow travels through a link and that link turns to down state, that flow is not recovered and it is necessary to generate a new one so that the new OVS flow rules perform their actions.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

The main goal of this dissertation was to develop a solution to overcome the limitations associated with Internet access in vehicular environments, namely on-board trains. Current solutions implemented in Portugal, such as for *Alfa Pendular* and *Intercidades*, use only one ISP at a time to access the Internet and may support the use of another ISP as a failover mechanism only. If the primary operator fails, then another one can be activated if configured. Besides that, when a train passenger switches carriages, the active connections established are broken due to the different WAN IP address used.

In order to solve the problem, the TMVC solution has been developed, proposing a network architecture that implements an overlay network for vehicular communications based on the SDN paradigm. The TMVC solution considers a central unity (the SDN Controller), which has holistic view of the network and manages all the network resources available in each train by assigning each of the clients network flows to the Internet gateways installed on-board of each carriage. The SDN controller is responsible for creating VXLAN tunnels to implement the overlay network and for defining the LB rules so that network flows can be divided among those tunnels. The TMVC solution provides an abstraction for multi-homed vehicular communications because train passengers access Internet resources using the overlay network despite the fact there are multiple (ISP underlay links) completely unknown to the user. The TMVC solution is capable of handling a set of possible network scenarios that could break Internet connectivity by quickly reconfiguring the network to re-establish Internet access. For example, if a user moves between two carriages, the system is capable of reacting immediately, performing seamless HO with minimal (under 1 second) Internet connectivity interruption. Furthermore, the system is capable of reacting in other scenarios such as when all the ISP VXLAN tunnels of one carriage turn into down state by redirecting the traffic through another carriage, reaching the Internet through the carriage which has ISP links available.

Regarding the implementation challenges, the first one was the network architecture design. Initially, the stipulated design for creating the overlay network assumed the existence of only

one VXLAN tunnel with three ISP underlay links. Using that network architecture, LB between underlay links could be achieve by creating a network bond between them and consequently to apply a known aggregation protocol such as LACP. However, if the physical ports of the system were aggregated to achieve network QoS improvements, the SDN controller application would lose its relevance. Since the SDN controller communicates with each OVS (that is the gateway present in each carriage) using the OF protocol, the controller can only affect and manage OF ports. Therefore, three OF ports were created in each OVS and the configured type for those ports is VXLAN.

The second challenge was to find a way of detecting an HO from one carriage to another. More precisely, creating a mechanism that detected that the link that connects the User Terminal to one carriage turned down state and that the link that connects the User Terminal to the other carriage turned up state. One option was to turn manually down the OF port present in each carriage. However, that is not correct, since the OF port should be always up in a real system. The responsible for the disconnection and consequently HO should always be the User Terminal. After a lot of debugging regarding each OVS in standalone mode, an MLD IPv6 event was detected and it was used to overtake the referred challenge.

The third implementation challenge was the amount of traffic that each OF VXLAN interface transmitted after the OF group tables installed in each OVS by the SDN application. When first testing the LB feature, a simple Iperf3 client was executed to generate a TCP flow from the User Terminal to the NS3 located in VM3. Despite having an equal weight distribution for the three OF VXLAN interfaces, traffic was not evenly distributed. After reading the OVS documentation, it was concluded that the OVS 2.4 version and later used the TCP header and other fields to calculate the hash that defined the bucket (in the OF group defined, each OF VXLAN port is a bucket) through which the traffic should travel. After understanding the information present in the documentation and that the LB works per network flow and not per packet, more than one TCP flow was generated in parallel and the amount of network flows that each OF VXLAN port transmitted were according to the weights established in the controller application, as intended.

After the implementation, groups of tests were defined and executed to evaluate individually each feature of the TMVC solution. The tests related to the Internet access allowed us to conclude that despite some identified problems that can be solved in future developments, the average delay the system incurs to recover from underlay network topology changes is below 1 second. One of the biggest issues to consider when applying an SDN architecture is the delay that an SDN system might cause due to the presence of a central element that manages a group of agents. The average delay obtained in all the situations tested confirm that an SDN architecture is reliable and it explains why it is an emerging paradigm nowadays.

Although there are still several improvements that can be made in the TMVC solution, the main goals established initially for this work were achieved which resulted in two main original contributions: 1) An SDN-based VXLAN solution, named TMVC, allowing transparent multi-interface vehicular Internet access for legacy user terminals and being capable of performing seamless HO between different interfaces with minimal connection loss to the Internet; 2) an SDN-

based LB mechanism compatible with the an SDN-based overlay network, capable of choosing a combination of VXLAN interfaces that connect each carriage to the Internet, optimizing the utilization of the available network resources and improving QoS.

5.2 Known Limitations and Future Work

As previously referred, despite the fact that the present dissertation had achieved the proposed objectives, there are some known limitations and there is still room for functional and performance improvements.

The first limitation is the fact that the TMVC solution does not take advantage of the predictive route followed by a vehicular system. Although some unexpected network scenarios might occur, the rail followed by a train is always the same and the rail network conditions can be predicted so that the SDN controller can implement mechanisms that increase the QoS of the system. For example, if the ISP link had no coverage in a specific area of the rail, the SDN controller could then create new OF rules in each carriage so that all the resources were moved to ISP links 2 and 3. In its current version, TMVC only considers reactive mechanisms. In future versions, it would be useful to include predictive intelligence and proactive mechanisms.

The second known limitation is related to the average delay obtained when the system needs to re-establish Internet connectivity in the situations exposed in Chapter 4. When the ISP through which an ARP packet travels turns into down state, the system needs a new ARP request to be executed in order to create new flow rules and consequently recover. The mechanism created to force a new ARP request was to clear the user terminal ARP cache. In a real system, that is a limitation since by design the system should not affect the user terminal. In the future, this mechanism should be improved. A mechanism that does not need a new ARP request to trigger the configuration of new OF rules should be created.

The third limitation is the fact that the HO between carriages is detected by an IPv6 MLD event and it is not guaranteed that all terminals connecting to the overlay network have the IPv6 technology embedded. In a real system, this mechanism could be detected using others tools. As an example, a tool that measures the level of received power between the terminal and the access point could be used as input for the creation of the rules at the SDN controller.

The fourth limitation is related to the TCP flows. When a TCP flow is sent through the overlay network and the link through which that flow turns into down state, the flow is never recovered by the system and it is necessary to generate a new TCP flow to reach the destination. This limitation can be restrictive in a situation where a train passenger accesses the Internet through a specific link and that link turns into down state. In that scenario, the user has to execute a new request to reach the destination. A mechanism should be created in the future to redirect a specific flow through other ISP link available.

References

[1] Claus Hetting. Use of wi-fi on european trains set to surge, April 2015. Available from https://wifinowglobal.com/news-and-blog/use-of-wi-fi-on-trains-set-to-skyrocket-500-by-2028-consultancy-says/.

[2] Shoaib Khan Ahmed Qureshi Karim, Sajjad and Imran Daud. Turning need into demand for wi-fi broadband internet access on trains. *International Journal of Computer Applications*, September 2012.

[3] Jitisha Patel Arpita Prajapati, Achyut Sakadasariya. Software defined network: Future of networking. *International Conference on Inventive Systems and Control (ICISC)*, page 1, June 2018.

[4] Tobias Hoßfeld Phuoc Tran-Gia Michael Jarschel, Thomas Zinner and Wolfgang Kellerer. Interfaces, attributes, and use cases: A compass for sdn. *IEEE Communications Magazine*, page 2, June 2014.

[5] Bhargavi Goswami Saleh Asadollahi and Mohammed Sameer. Ryucontroller's scalability experiment on software defined networks. *IEEE International Conference on Current Trends in Advanced Computing (ICCTAC)*, pages 2–3, June 2018.

[6] Naresh Kumar Nagwani Ravindra Kumar Chouhan, Mithilesh Atulkar. Performance comparison of ryu and floodlight controllers in different sdn topologies. *1st International Conference on Advanced Technologies in Intelligent Control, Environment, Computing Communication Engineering (ICATIECE)*, page 2, March 2019.

[7] Adnan Iqbal Zuhran Khan Khattak, Muhammad Awais. Performance evaluation of openday-light sdn controller. *20th IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, page 2, Dec 2014.

[8] Nick McKeown Saurav Das, Guru Parulkar. Unifying packet and circuit switched networks. *IEEE Globecom Workshops*, pages 1–6, Nov 2009.

[9] Open Networking Foundation. Openflow switch specification - version 1.0, Dec 2009. Available from https://opennetworking.org/wp-content/uploads/2013/04/openflow-spec-v1.0.0.pdf.

[10] Raj Jain. Introduction to openflow, 2015. Available from https://www.cse.wustl.edu/~jain/cse570-15/ftp/m_15ofl.pdf.

[11] Open Networking Foundation. Openflow switch specification - version 1.1, Feb 2011. Available from https://opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.1.0.pdf.

[12] Open Networking Foundation. Openflow switch specification - version 1.2, Dec 2011. Available from https://opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.2.pdf.

[13] Open Networking Foundation. Openflow switch specification - version 1.3, June 2012. Available from https://opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf.

[14] Open Networking Foundation. Openflow switch specification - version 1.4. https://opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.4.0.pdf.

[15] Open Networking Foundation. Openflow switch specification - version 1.5, March 2015. Available from https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf.

[16] Qi Hao Fei Hu and Ke Bao. A survey on software-defined network and openflow: From concept to implementation. *IEEE Communications Surveys Tutorials (Volume: 16, Issue: 4, Fourthquarter*, page 3, May 2014.

[17] The Linux Foundation. Production quality, multilayer open virtual switch, 2016. Available from http://www.openvswitch.org/.

[18] The Linux Foundation. Open vswitch manual - ovs-switchd(8). Available from http://www.openvswitch.org/support/dist-docs/ovs-vswitchd.8.html.

[19] Shie-Yuan Wang Hung-Wei Chiu. Boosting the openflow control-plane message exchange performance of openvswitch. *IEEE ICC 2015 - Next Generation Networking Symposium*, page 2, June 2015.

[20] Eduardo Jacob Marivi Higuero Alaitz Mendiola, Jasone Astorga. A survey on the contributions of software-defined networking to traffic engineering. *IEEE Communications Surveys Tutorials*, page 13, May 2017.

[21] Scott Lowe. Examining open vswitch traffic patterns, 2013. Available from https://blog.scottlowe.org/2013/05/15/examining-open-vswitch-traffic-patterns/#scenario-3-the-isolated-bridge.

[22] Project Floodlight. How to work with fast-failover openflow groups, 2018. Available from https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/7995427/How+to+Work+with+Fast-Failover+OpenFlow+Groups.

[23] The Linux Foundation. Using openflow. Available from https://docs.openvswitch.org/en/latest/faq/openflow/.

[24] Pascal Bouvry Jianguo Ding, Ilangko Balasingham. Management of overlay networks: A survey. *IEEE Third International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies)*, page 1, Oct 2009.

[25] Internet Engineering Task Force. Rfc 7348: Virtual extensible local area network (vxlan): A framework for overlaying virtualized layer 2 networks over layer 3 networks, August 2014. Available from https://tools.ietf.org/html/rfc7348.

[26] ProgrammerSought. The relationship and concepts between data center sdn network, vxlan, and virtualization. Available from https://www.programmersought.com/article/93765120839/.

[27] Gustavo D. Salazar Edison F. Naranjo. Underlay and overlay networks: The approach to solve addressing and segmentation problems in the new networking era. *IEEE Second Ecuador Technical Chapters Meeting (ETCM))*, page 2, Oct 2017.

[28] Alexander La rosa. Vxlans: Applying virtualization to networks., April 2019. Available from https://pandorafms.com/blog/vxlan-network/.

[29] Juniper Networks. What is vxlan? Available from https://www.juniper.net/us/en/products-services/what-is/vxlan/.

[30] Huawei. Smooth vm migration using vxlan, December 2020. Available from https://support.huawei.com/enterprise/en/doc/EDOC1000178188/df509648/smooth-vm-migration-using-vxlan.

[31] Internet Engineering Task Force. Rfc 7347 nvgre: Network virtualization using generic routing encapsulation, September 2015. Available from https://tools.ietf.org/html/rfc7347.

[32] Olufemi Komolafe. Ip multicast in virtualized data centers:challenges and opportunities. *IFIP/IEEE International Symposium on Integrated Network Management*, page 4, May 2017.

[33] Peter Palúch Jakub Hrabovský-Jozef Papán Marek Moravčík, Pavel Segeč. Clouds in educational process. *13th International Conference on Emerging eLearning Technologies and Applications (ICETA)*, page 3, November 2015.

[34] Fernando Lama Ruano. Creation of a virtual overlay network with sdn and vxlan. Technical report, Universitat Politècnica de Catalunya, June 2017.

[35] Ronpeng Li Zhifeng Zhao, Feng Hong. Sdn based vxlan optimization in cloud computing networks. page 2, October 2017.

[36] Roch Glitho Edmundo Madeira Pedro Rezende, Somayeh Kianpisheh. An sdn-based framework for routing multi-streams transport traffic over multipath networks. *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, page 1 and 6, May 2019.

[37] Pushpa C. Fancy, M. Pushpalatha. Experimentation of traditional load balancing algorithms in software defined network. *International Journal of Recent Technology and Engineering (IJRTE)*, pages 4–5, June 2019.

[38] RYU SDN Community. Ryu application api. Available from https://ryu.readthedocs.io/en/latest/ryu_app_api.html.

[39] RYU SDN Community. ryu documentation - release 4.34. Available from https://readthedocs.org/projects/ryu/downloads/pdf/latest/.

[40] RYU SDN Community. ryu documentation - release 4.34. Available from https://readthedocs.org/projects/ryu/downloads/pdf/latest/.

[41] RYU SDN Community. Getting started. Available from https://ryu.readthedocs.io/en/latest/getting_started.html.

[42] Aki Tuomi. ryu-auto-vxlan. Available from https://github.com/cmouse/ryu-auto-vxlan/blob/master/fabric.py.

[43] RYU SDN Community. ryu documentation - release 4.34. Available from https://readthedocs.org/projects/ryu/downloads/pdf/latest/.

[44] RYU SDN Community. ryu documentation - release 4.34. Available from https://readthedocs.org/projects/ryu/downloads/pdf/latest/.

[45] Li Cheng. Multipath. Available from https://github.com/muzixing/ryu/blob/master/ryu/app/multipath/multipath.py.

[46] Open Networking Foundation. Using openflow. Available from https://docs.openvswitch.org/en/latest/faq/openflow/.

[47] W. Fenner ATT Research-B. Haberman S. Deering, Cisco Systems. Rfc 2710. Available from https://datatracker.ietf.org/doc/html/rfc2710.