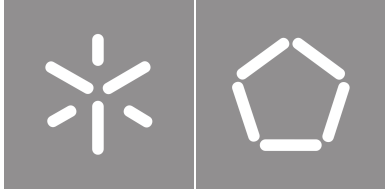
Universidade do Minho
Escola de Engenharia

Luís Carlos Sousa Magalhães

SDN support on V2X devices

January, 2025



Universidade do Minho
Escola de Engenharia

Luís Carlos Sousa Magalhães

SDN support on V2X devices

Master Thesis
Master in Informatics Engineering

Work developed under the supervision of:
António Costa
Maria João Nicolau

January, 2025

Resumo

Suporte de SDN em dispositivos V2X

Nas últimas três décadas, a procura de integração da conectividade à Internet no domínio das redes veiculares, normalmente designadas por Vehicular ad hoc Network (VANET), tem vindo a aumentar. Software Defined Networking (SDN) representa um novo paradigma de rede que promete numerosos benefícios, e cuja concretização pode potencialmente melhorar as VANETs. Esta linha de pensamento levou ao aparecimento da Software Defined Vehicular Network (SDVN), sendo um dos principais obstáculos à sua adoção a falta de plataformas experimentais baseadas em hardware real. Este problema representa um desafio significativo para a investigação em SDVN, uma vez que é altamente complexo e exige uma compreensão abrangente dos domínios SDN e VANET. Além disso, existe uma escassez de investigação que examine a integração da Programming Protocol-independent Packet Processors (P4) e da segunda geração de SDN com as VANETs. Dada a natureza inovadora deste desafio e a complexidade excecional destes dois domínios, a sua resolução constitui um objetivo altamente interessante e ambicioso.

Este documento tem por objetivo colmatar a lacuna existente neste domínio de estudo, apresentando o desenvolvimento e a implementação de um On Board Unit (OBU)/Intelligent Transport System (ITS) vehicle subsystem que cumpra as especificações de SDN. O protótipo proposto baseia-se em hardware acessível e facilmente disponível e utiliza software de código aberto sempre que possível. Além disso, este documento apresenta uma nova perspetiva sobre o futuro do domínio SDVN, com base numa análise global abrangente. Estas conclusões apresentam uma potencial trajetória para a evolução da tecnologia SDVN. Este artigo apresenta o desenvolvimento de um protótipo constituído por um PC Engines apu3d4 que opera o switch P4 Behavioral Model version 2 (BMv2), comunicando a frequências de 5,9 GHz através da norma 802.11p. Esta solução estabelece as bases necessárias para o desenvolvimento de um ambiente SDVN completo no qual podem ser efectuados testes.

Palavras-chave: Redes vehiculares, Redes definidas por software, Intelligent Transport System, Redes vehiculares definidas por software, European Telecommunications Standards Institute, On Board Unit, ITS vehicle subsystem, 802.11p, Open virtual Switch, Behavioral Model version 2, Programming Protocol-independent Packet Processors

Abstract

SDN support on V2X devices

Over the past three decades, there has been a surge in demand for the integration of internet connectivity within the domain of vehicular networks, commonly referred to as VANETs. SDN represents a novel networking paradigm that promises numerous benefits, the realization of which could potentially enhance VANETs. This line of thinking has led to the emergence of SDVN, with one of the primary obstacles to its adoption being the lack of experimental platforms utilizing real hardware. This problem represents a significant challenge for SDVN research, as it is highly intricate and necessitates a comprehensive grasp of both SDN and VANET domains. Moreover, there is a dearth of research examining the integration of P4 and the second generation of SDN with VANETs. Given the novel nature of this endeavor and the exceptional complexity of these domains, solving this challenge presents a highly intriguing, yet difficult, objective.

This paper aims to address the existing gap in this field of study by presenting the development and implementation of an OBU/ITS vehicle subsystem that complies fully with the specifications of SDN. The proposed prototype is based on affordable and readily available hardware and utilizes open-source software wherever feasible. Additionally, this paper provides a new perspective on the future of the SDVN field, based on a comprehensive global analysis. These findings present a potential trajectory for the evolution of SDVN technology. This paper presents the development of a prototype comprising a PC Engines apu3d4 that operates the BMv2 P4 switch, communicating at 5.9GHz frequencies via the 802.11p standard. This solution establishes the necessary framework for the assembly of a comprehensive SDVN environment in which tests may be conducted.

Keywords: Vehicular ad hoc Network, Software Defined Networking, Intelligent Transport System, SDVN, European Telecommunications Standards Institute, On Board Unit, ITS vehicle subsystem, 802.11p, Open virtual Switch, Behavioral Model version 2, Programming Protocol-independent Packet Processors

Contents

List of Figures

List of Tables

Acronyms

3GPP 3rd Generation Partnership Project *(pp. 19, 20)*
5GAA 5G Automotive Association *(pp. 19, 20)*

API Application Programming Interface *(pp. 31, 36, 38–40, 43–49, 52, 55, 57, 64, 65, 84, 96)*
AU Autonomous Unit *(pp. 8–10, 13)*

BIOS Basic Input/Output System *(pp. 44, 63)*
BMv2 Behavioral Model version 2 *(pp. ii, iii, viii, ix, 64, 66, 71, 72, 84, 86–89, 91–94, 96, 98)*
BSS Basic Service Set *(pp. xii, 17)*
BTP Basic Transport Protocol *(p. 21)*

C-V2X Cellular V2X *(pp. 19, 20, 28, 55, 63)*
C2C-CC Car-2-Car Communication Consortium *(pp. vii, 6, 8–14)*
CAM Cooperative Awareness Message *(pp. vii, 22, 23, 57)*
CEN European Committee for Standardization *(p. 22)*
CPU Central processing unit *(pp. 41, 56, 62, 65, 66, 68)*

DENM Decentralized Environmental Notification Message *(pp. vii, 22, 23)*
DRAM Dynamic Random-access memory *(pp. 56, 65, 68)*
DRAM Static Random-access memory *(pp. 41, 57)*

ECC Electronic Communications Committee *(pp. 17, 18)*
ECU Electronic Control Unit *(pp. 10, 11)*
ETSI European Telecommunications Standards Institute *(pp. ii, iii, 6, 8–10, 12, 13, 17, 18, 20–23, 26, 27, 56–59, 63, 85, 96, 97, 99)*
EU European Union *(pp. vii, 6, 16–18, 24, 28, 29, 50, 51)*

FCC Federal Communications Commission *(pp. vii, 17, 18)*

GN6 GeoNetworking-IPv6 *(p. 21)*
GPS Global Positioning System *(pp. 5, 11, 57, 58)*

IBSS Independent Basic Service Set *(p. 17)*
IEEE Institute of Electrical and Electronics Engineers *(pp. 16, 17, 19, 57)*
IP Internet Protocol *(pp. ix, xii, 9, 28, 80, 81, 86, 91, 93)*
IPv6 Internet Protocol version 6 *(pp. xii, 21, 28)*
ISO International Organization for Standardization *(p. 22)*
ITS Intelligent Transport System *(pp. ii, iii, vii, ix, xii, 6, 9–20, 22–24, 26, 27, 29, 51, 53–60, 65, 66, 71, 96–98)*
ITS-S ITS station *(pp. 9, 10, 96)*
ITSC Architecture of Communications in ITS *(pp. vii, 13, 24, 26)*

LTE Long-Term Evolution *(pp. 19, 20, 57, 58, 63, 66, 69)*

MANET Mobile ad hoc Network *(pp. 5–7, 20)*
mPCIe Mini PCI Express *(pp. 56–58, 65, 66, 68)*
mSATA Mini-Serial Advanced Technology Attachment *(pp. 65, 66, 68)*

NIC Network Interface Controller *(pp. xiv, 62)*
NOS Network Operating System *(pp. 46, 48, 49)*
NPU Network Processing Unit *(pp. 41–43)*

OBU On Board Unit *(pp. ii, iii, vii, 2, 3, 8, 9, 11, 13, 56–59, 62, 66, 71, 98, 99)*
OCB Out of Context BSS *(pp. 17, 19)*
ODL Open Day Light *(pp. 48, 65)*
ONF Open Networking Foundation *(pp. 30, 39, 45, 85)*
ONL Open Network Linux *(p. 63)*
ONOS Open Network Operating System *(pp. viii, ix, 48, 65, 66, 71–73, 78, 80, 83–86, 89–91, 95, 96, 98)*
OS Operating System *(pp. xii, xiii, 44, 46, 56, 63, 66, 70)*
OSI Open System Interconnection *(pp. 9, 15, 20, 22, 53, 96)*
OvS Open virtual Switch *(pp. ii, iii, viii, 64, 66, 71–82, 84, 98)*

P4 Programming Protocol-independent Packet Processors *(pp. ii, iii, 32, 39–41, 43, 45, 51, 59, 64, 71, 84–86, 91, 96–99)*

PC Personal Computer *(pp. 40, 41, 71)*

PCI Peripheral Component Interconnect *(pp. xii, 62)*

PDU Packet Data Unit *(pp. 22, 23)*

PISA Protocol Independent Switching Architecture *(p. 43)*

POX Pythonic Network Operating System *(p. 48)*

PSA Portable Switch Architecture *(p. 43)*

QoS Quality of Service *(pp. 8, 55)*

RAM Random-access memory *(pp. xi, 62, 65, 68)*

RSU Roadside Unit *(pp. vii, 8, 9, 12, 13, 57, 58)*

SATA Serial Advanced Technology Attachment *(pp. xii, 56)*

SD Secure Digital *(pp. 66, 68)*

SD-VANET Software Defined Vehicular ad hoc Network *(p. 50)*

SDN Software Defined Networking *(pp. ii, iii, vii–ix, 1–3, 28, 30–41, 44–66, 71–75, 77, 80–82, 85, 86, 89, 94, 96–99)*

SDR Software Defined Radio *(pp. 64, 99)*

SDVN Software Defined Vehicular Network *(pp. ii, iii, viii, 1–4, 50–54, 56–60, 65, 98)*

SIM Subscriber Identity Module *(pp. 66, 68)*

SNMP Simple Network Management Protocol *(p. 45)*

SONiC Software for Open Networking in the Cloud *(p. 63)*

SSD Solid State Drive *(pp. 56, 57, 66, 68, 69)*

TNA Tofino Native Architecture *(p. 43)*

UDP User Datagram Protocol *(p. 21)*

US United States of America *(pp. vii, 6, 16–18, 21)*

V2I Vehicle to Infrastructure communication *(pp. 14, 17)*

V2V Vehicle to Vehicle communication *(pp. 14, 17)*

V2X Vehicle to Everything communication *(pp. xi, 14, 16, 20)*

VANET Vehicular ad hoc Network *(pp. ii, iii, vii, xiii, 1–3, 5–9, 11, 12, 14–17, 19–21, 23–25, 27, 28, 50–53, 55–58, 62, 65, 71, 72, 84)*

VM Virtual Machine *(pp. 72, 73, 86)*

VRF Virtual Routing and Forwarding *(pp. viii, ix, 72, 73, 76, 77, 80, 81, 83, 86, 88, 93, 94)*

WAVE Wireless Access in Vehicular Environments *(pp. 16, 17)*

WNIC Wireless Network Interface Controller *(pp. 56, 58)*

1 Introduction

This chapter serves as an introduction to this thesis, providing an overview of its context, motivation, objectives, and methodology.

1.1 Contextualization

The Internet is universally acknowledged as one of the most significant technological advancements in human history, with its widespread adoption and the numerous benefits it has brought about having irrevocably transformed the manner in which societies in the 21st century live and communicate.

While this impact is indisputable, the exponential growth observed over the past two decades has revealed significant flaws in Internet design. Indeed, in its current state, the Internet is not fully capable of satisfying the demands of the modern digital landscape.

One such issue pertinent to this thesis arises from the fact that our society is becoming increasingly mobile. With the advent of widespread automobile use, researchers and engineers were quick to recognize the vast potential advantages of enabling these vehicles to communicate with one another. Such developments have the potential to enhance a number of aspects of modern transportation, from road security to the quality of life for drivers. Given the direct impact of this technology on driver wellbeing, it is crucial that rigorous real-world testing be conducted to guarantee the safety of the general public.

For the past three decades, universities, companies, and governments have been engaged in collaborative efforts to develop the technology known as VANETs, whose objective is to enable vehicles to communicate directly with each other on the road. This process has made enormous progress in adapting the Internet to vehicular environments. Nevertheless, these solutions retain some of the aforementioned shortcomings inherent in Internet designs.

SDN, a new networking paradigm, promises to deliver increased innovation, reduced cost, simplified management, and improved performance. It thus promises to address the limitations of the Internet and facilitate further growth. In light of this potential, researchers have recently focused their attention on this novel and promising field, treating it as a potential solution to many of the challenges currently faced in the current Internet landscape. Notably, this has occurred in the domain of VANET, which has led to the creation of the SDVN field. The potential for SDN to enhance the capabilities of this technology

has prompted recent studies to apply SDN to vehicular networks with the goal of developing optimized solutions for the future.

1.2 Motivation

In accordance with the indispensable role that real-world testing plays in the ultimate success of VANET, it is of the utmost importance to conduct comprehensive testing in realistic settings to ensure the successful deployment of SDVN. The majority of SDVN projects are designed for use in the real world, yet they are predominantly subjected to simulation-based testing, which compromises the veracity of the results. This emphasizes the pressing need for the availability of real hardware testing platforms to achieve more accurate and reliable results in this critically important area.

The development of a functional hardware testing platform utilizing free and open-source software components for SDVN projects represents a substantial advancement in the evolution of both the SDN and VANET fields. This will demonstrate the value of SDN beyond the data center, establishing it as a dominant force in the field while also fostering improvements in VANET.

1.3 Objectives

This thesis presents a proposal for the design and implementation of a functional SDVN device. In essence, the core objective is to develop a device that can be classified as both a SDN and a VANET device in a single entity.

As a further clarification, the device will be designed and assembled to fulfill the role of an OBU and thus be able to communicate in the vehicular ad hoc space. The device must also conform to the design principles of SDN.

This document will undertake a detailed and thorough analysis of the SDN and VANET fields in order to define the characteristics of this device. Nevertheless, it is possible to establish a preliminary set of guidelines to which the device must adhere even before such an evaluation is conducted. The following two criteria serve as the fundamental basis for the evaluation of the devices in question. First and foremost, the device must be affordable. Secondly, the software components utilized must be open-source, wherever feasible.

1.4 Methodology

The methodology employed in this thesis was designed with the objective of assembling the necessary theoretical basis for an assessment of the viability of the SDVN technology as a whole. The selected methodologies are explained and justified in the context of the research objectives.

The dissertation will commence with a comprehensive analysis of the SDVN field. In order to achieve the proposed objective of developing an SDN-compatible OBU, it is essential to conduct such research.

While a review of the existing software in the SDVN field is undoubtedly important, the goal of the research step of this document is to extensively analyze the SDN and VANET fields.

In regard to SDN, this research aims to provide a rationale for its integration with VANET. It is therefore essential to present a comprehensive analysis of the underlying motivations, advantages, and fundamental principles that shape this technology.

Conversely, research on VANET is primarily concerned with defining the characteristics that distinguish this field of study from other ad hoc technologies, thereby establishing a comprehensive understanding of this field. Moreover, it is of critical importance to gain insight into the current state of this technology and to identify the remaining steps required for its widespread adoption. In light of this, this study will primarily focus on investigating the advancements in the field within the European context.

The research component of this document will be concluded with a review of existing literature, the objective of which is to identify any studies that share similar motivations to this document.

The practical component of this dissertation will begin with a period of rigorous experimentation and familiarization with existing software in the field. Subsequently, a new SDN OBU will be designed and assembled, utilizing upgraded hardware and adhering to the requisite standards for SDN compliance.

To conclude, a concrete testing scenario will be defined and subsequently executed with the intention of illustrating and validating the aforementioned achievements.

1.5 Main contributions

The primary contributions of this thesis are as follows:

- A comprehensive and insightful analysis of the regulations governing VANETs.
- A detailed examination of SDN and its underlying motivations and benefits.
- A well-founded and realistic assessment of the potential implications of SDN for future developments.
- The creation of a functional OBU prototype.

These achievements are significant because our approach to the theoretical foundations of SDVN represents a novel and crucial contribution to the field, which is essential for the advancement and future of this technology.

1.6 Document structure

This paper is divided into eight other main chapters. Chapter 2 aims to provide a broad context about vehicular networks, in order to understand the main challenges in this field. In chapter 3, a very detailed overview of SDN is made, with a higher emphasis on the data plane, to show the power and potential of this technology. Chapter 4 serves to address what are the advantages and difficulties of using SDN in

a vehicular context, while also presenting the SDVN architecture. In Chapter 5, studies in this field that share a similar objective to the present study are reviewed and analyzed. Chapter 6 presents a critical analysis of the future of SDVN. Chapter 7 initiates the implementation phase of this thesis by delineating the design of the prototype in layers and concluding with a description of the hardware that will be utilized. In Chapter 8, the pertinent software and hardware are evaluated, resulting in the completion of the solution prototype. Chapter 9 concludes this document and outlines future work.

2

Vehicular ad hoc Networks

The automobile was one of the most revolutionary inventions in human history, for most cities around the world are dependent on cars for the transportation of goods and people and, therefore, the prevalence of vehicles in our daily lives is greater than ever before. Even though it is impossible to predict if future vehicle usage will continue to be as dominant as today, it is reasonable to assume that this technology will never disappear completely. Even today, in cities dominated by alternative modes of transportation, the use of motorized vehicles is still present and, in some cases, a necessity.

It has been a long-time goal for car manufacturers and researchers to bring internet connectivity to automobiles, as the benefits are enormous. The Internet is an extremely powerful technology, not only for the countless services it provides but also for the ability of two endpoints to communicate almost instantly. Moreover, recent improvements to wireless communication technologies not only make this goal more realistic but also even more attractive, as integrating network interfaces with Global Positioning System (GPS) receivers and sensors only increases the potential of allowing cars to communicate with each other [1].

The efforts to bring connectivity to automobiles resulted in the emergence of the technology named VANET, which is an adaptation of the Mobile ad hoc Network (MANET) technology, specialized for automobiles [2] [3]. In contrast to wired networks, where all devices are connected to the infrastructure at all times to communicate with each other, this technology allows devices to transmit data directly to other devices inside its range, as illustrated in Figure 1. As a result, networks are generated spontaneously between nodes that are in range of each other, which makes topologies unstable and requires each node in the network to act as both a transmitter and a receiver.

As the Internet was not designed with mobility in mind, most protocols and standards were not developed to support the high-speed scenarios of VANET. A tremendous amount of work and research efforts has already been conducted in this area, amounting to decades of research, development, and standardization. However, the development of VANET technology began without any major centralized entity to standardize its implementation worldwide, resulting in various architectures emerging simultaneously all around the globe. It is common for different regions of the world to develop different standards for infrastructure-related technologies and these solutions tend to converge as they seek to solve the same problems in the most optimal way.

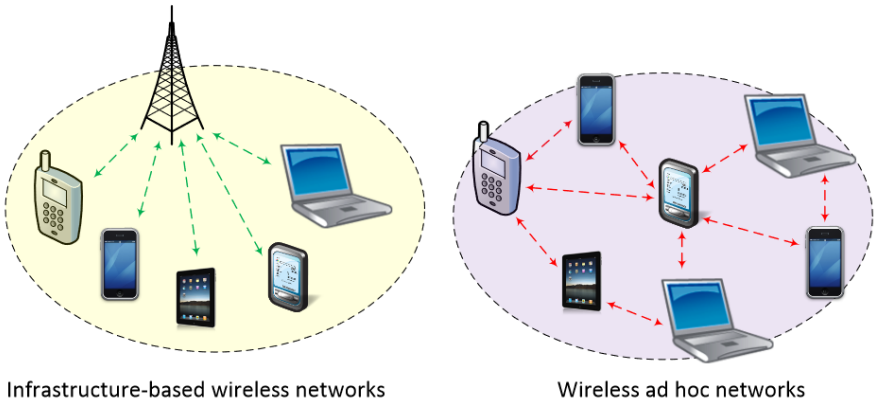


Figure 1: Comparison between infrastructure networks [4]

The leaders in global research on VANETs are considered to be the US, Japan, and the EU. These regions are noteworthy as they have provided the majority of the technological foundations and advancements in the field. For the sake of clarity and relevance, this paper focuses on the European VANETs architecture through the discussion of the C2C-CC and the European Telecommunications Standards Institute (ETSI), two prominent and influential institutions in the field.

The C2C-CC architecture proposal is designed with a high level of abstraction and is therefore fairly easy to understand. In contrast, the ETSI standards are significantly more detailed and technical, since they provide the official standard for the EU. ETSI uses the term ITS to refer to VANETs, as their efforts to connect vehicles together goes beyond cars and aims to interconnect planes, trains, boats, etc.

There are other conventions in this field, such as the C-ROADS platform[5], which are important for the development of this technology. In fact, several conventions have influenced the ETSI standards and would be worth discussing. However, in order to avoid overcomplicating the subject, only the ETSI and C2C-CC architectures will be covered.

2.1 VANET characteristics

The VANET technological domain was established as a specialized subset of the MANET research field based on some unique characteristics, and thus VANET shares most of its properties with MANET. Some key differences require the development of novel solutions, and in this section, we will go through all the characteristics of VANET while shedding light on what sets this technology apart from its origin.

2.1.1 High mobility

MANET topologies are highly dynamic, driven by the mobility of nodes. High mobility leads to uncertain connectivity, which makes ad-hoc topologies extremely volatile and unpredictable, with a high chance of partitions[6]. VANETs not only inherit this characteristic but also amplify its volatility since VANET nodes are vehicles capable of traveling at extremely high speeds.

For instance, two cars traveling in opposite directions on the same road will only be within range of each other for a few seconds. In this short time, the vehicles must attempt to establish a connection before attempting to exchange valuable information, which means some links may be disconnected before there is a chance to use them [3]. On the other hand, two cars traveling in the same direction can maintain a connection indefinitely as long as they keep on the same path at the same speed. These two scenarios are polar opposites, making the reliability of connections inconsistent.

2.1.2 Predicted mobility

Beyond being more dynamic, MANET topologies are also random, as nodes can move arbitrarily and at unpredictable times, with virtually no restraint. VANET topologies diverge from MANET in this matter, as it is possible to reliably predict the path of nodes in an ad-hoc network. Vehicles are primarily used on pre-built infrastructure [3], like roads and motorways, and have to follow predetermined paths to reach their destination, which makes it possible to predict their future location. Vehicles are also forced to obey road signs and traffic lights and have to respond to other vehicles' movements[2], making topologies even more predictable.

On top of all of that, if drivers disclose their desired destination it becomes possible to predict the complete route of a node with near 100% precision. It is important, however, to keep in mind that this is not always true, and in some instances, drivers may adjust their path in reaction to the information received by the network[3].

A negative consequence of road use can be seen in more linear topologies when compared with MANETs, which inevitably undermines path redundancy[6].

2.1.3 Power and Computational ability

Another relevant change from MANETs is the computational capabilities and power constraints of the nodes. MANET was designed with small devices in mind, which have limited battery capacity and computational power constraints. In contrast, vehicles have access to a reliable power source[3][1], so there are essentially no concerns in this regard. However, cars should not be thought of as an infinite power source, and normal precautions should be taken when optimizing routing algorithms, etc.

2.1.4 Congestion and Scalability issues

VANET suffers from an absurdly variable node density. A vehicle can be on a road in a remote location with the nearest internet connection of other vehicle kilometers away from their location, or in the middle of a traffic jam at an intersection surrounded by hundreds of vehicles all trying to communicate within its range.

Congestion scenarios create a bandwidth problem that is exacerbated by the presence of nearby obstacles like other cars and buildings, which common in an urban environment[6]. Wireless links are significantly more fragile than their wired counterparts because they are subject to fading, noise and

interference[7]. This also forces researchers to come up with effective measures for sharing physical media, as another issue of concern is ensuring Quality of Service (QoS) measures for critical messages, like road hazard warnings, which presents a core issue[6].

Variable network density presents a necessity for the protocols developed to be capable of dealing with any situation thrown at them in the real world.

2.2 VANET components

The first step in the understanding VANET scenarios is the analysis of the basic elements involved. With this in mind, the atomic components of both the C2C-CC and ETSI are explained in this section.

2.2.1 C2C-CC approach

According to the C2C-CC architecture, three main components can be identified, these being the RSU, OBU and Autonomous Unit (AU).

1. Roadside Units are stationary devices placed alongside roads, usually in high-density zones such as junctions, intersections and traffic lights.

   These devices have a reliable connection to the Internet and communicate with all vehicles within their effective range, bridging the gap between vehicle ad hoc networks and the rest of the Internet. RSUs can also be equipped with multiple communication devices for optimal wireless connectivity. In addition, RSUs can also take advantage of the ad hoc characteristics of on-board units by using them as transmitters, thereby extending their effective range, as depicted on Figure 2.

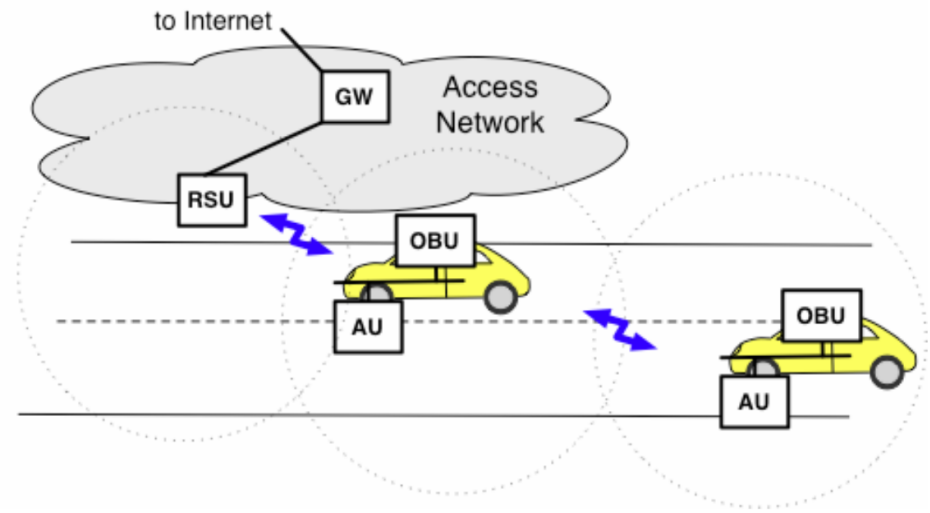


Figure 2: A RSU provides Internet connectivity through an OBU. [8]

RSUs can run safety applications like fixed road hazard warnings (work-zone warnings, bridge warnings, etc.) by broadcasting safety messages to nearby vehicles, making them more than mere connection points for vehicles.

2. On Board Unit is a small device integrated into vehicles that operates in the ad hoc space as they receive, send, and forward messages between each other and with RSUs in an ad hoc manner. OBUs provide it's wireless communication capabilities to any connected device used within the vehicle.

3. The term Autonomous Units refers to all the various devices that run a single or a set of applications that connect to the OBU and harness its wireless capabilities. The AUs can be physically integrated or permanently wired to the OBU being an integrated part of the vehicle. These types of AUs are typically safety-related applications. AUs can also be wireless devices like laptops or smartphones, wirelessly connecting to an OBU and utilizing its capabilities for leisure applications.

4. In addition to these three main components, this institution acknowledges the potential presence of other network entities, like public hotspots, instances of a Mobile IP infrastructure, application servers, control centers, etc. However, these are not categorized and are therefore outside of the scope of this Consortium.

2.2.2 ETSI approach

The ETSI standard diverges from the simple and straightforward definition of the C2C-CC by specifying two categories to define the communication entities of VANETs: the ITS sub-systems and the functional components. The former, comparable to the C2C-CC components, represent similar system categories that exist in the VANET space. The latter are the building blocks that make up the ITS subsystems and are therefore responsible for specific functions or tasks within the overall systems.

2.2.2.1 Functional components

The functional components described in this architecture follow the layered conceptual structure of the Open System Interconnection (OSI) model [9], comprising five such components: the ITS station (ITS-S) host, the ITS-S gateway, the ITS-S router, the ITS-S border router and the ITS-S interceptor.

The most significant component of this architecture is the ITS-S host, since it provides the minimum level of functionality required to run ITS applications. It represents the basic reference architecture for any device wishing to communicate in the VANET space, and is present on all ITS sub-systems. This architecture is described in depth in section 2.5

In addition, all other components are a variant of the ITS-S host, adding communication capabilities to it. The ITS-S gateway converts messages from the Internet to the ITS domain at layers 5 to 7 of the OSI model. The ITS-S router converts various ITS protocols at layer 3, and the ITS-S border router performs a similar task, doing the same as the router, but with a more generic network, without management or

security awareness. The ITS-S interceptor is a generic term equivalent to any of the previous three components, plus it can represent an implementation-specific connection of the ITS station-internal network to another network.

2.2.2.2 Sub-systems

As previously introduced, all functional components come together to form various sub-systems, of which there are four. These sub-systems represent a specific context in which the functional components combine to attempt to provide wireless communication capabilities, either in the form of direct contact with the ad hoc environment or indirectly.

All ITS sub-systems consist of the unique context they support and a specific ITS station to meet the needs of that unique situation. These stations can also be implemented in a single physical unit or in multiple physical units.

1. Personal ITS sub-system:

   The Personal ITS sub-system represents a mobile or other personal device that can connect to the vehicle's internal network and use its ad-hoc wireless capabilities. It includes the user's device and a Personal ITS Station. This station is the simplest of the four, consisting of a single ITS host. This can be observed in Figure 3.

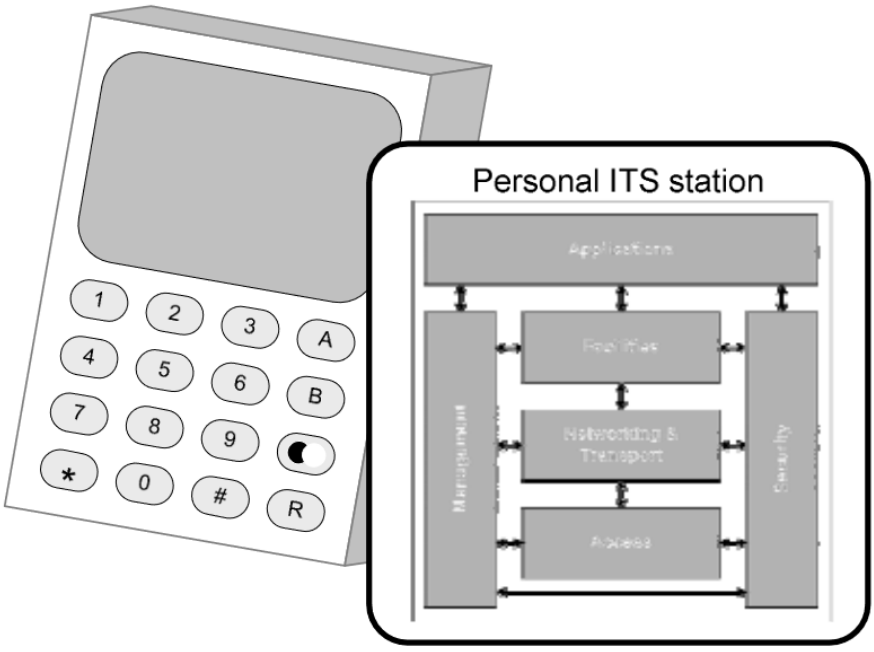


Figure 3: Personal ITS station in a Personal ITS sub-system [9]

This sub-system is comparable to an AU, with a narrower definition. The ETSI architecture defines the existence of an Electronic Control Unit (ECU), which also falls under the AU definition created by the C2C-CC. The ECU will be described along with the vehicle ITS sub-system.

2. Central ITS sub-system; The purpose of the Central ITS sub-system is to provide centralized ITS applications, like traffic operations or content delivery, to VANET users. It is composed of a central ITS station that can be located anywhere. Examples of a Central ITS sub-system are traffic management centers and road operator centers.

   The Central ITS station contains an ITS host and two optional ITS interceptors, being an ITS gateway and border router. The Central ITS sub-system is represented in Figure 4.

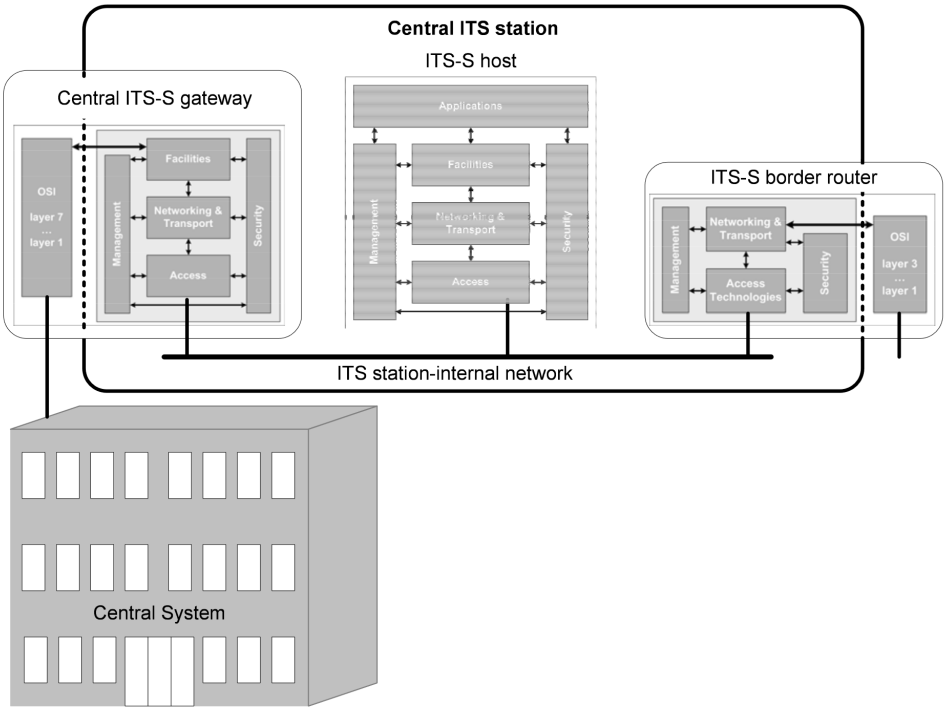


Figure 4: Central ITS station in a Central ITS sub-system [9]

The border router communicates with any other ITS system via any external network, while the gateway provides a more standardized connection.

3. Vehicle ITS sub-system:

   The Vehicle ITS sub-system represents the cars, buses, trains, trucks, airplanes, and any other vehicle or means of transportation that can communicate wirelessly in an ad hoc manner and belongs to the VANET space. This definition is very similar to the OBU definition made by the C2C-CC, but there are some important minor differences that make it more rigorous.

   This subsystem is made up of the Vehicle ITS station and the internal vehicle network, which is connected to a variety of ECUs. ECUs represent any kind of system or application that requires a connection to the Internet or to any other vehicle. Examples of such applications are GPS, traffic control, road hazard warnings, etc.

   Like all stations, the Vehicle ITS station is made up of an ITS host with two optional ITS interceptors, nominally an ITS gateway and an ITS router. This sub-system is depicted in Figure 5.

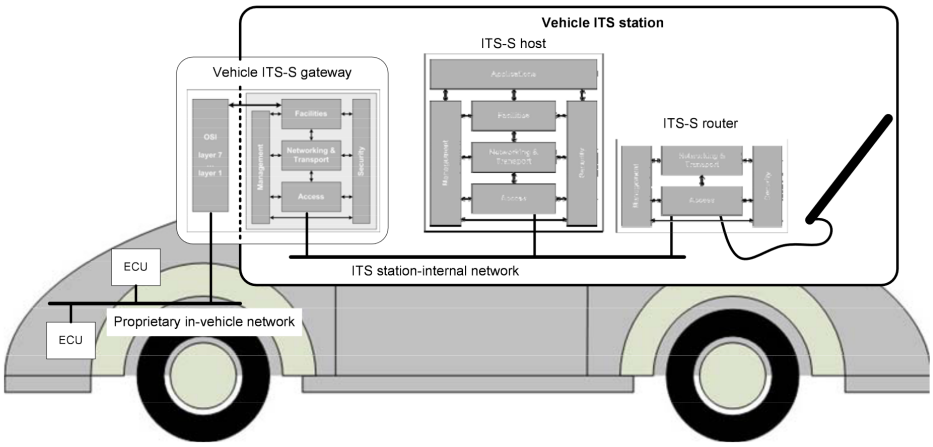


Figure 5: Vehicle ITS station in a Vehicle ITS sub-system [9]

The ITS gateway connects the internal vehicle network, which is proprietary in most cases, to the ITS station. Meanwhile, the ITS router connects to other Vehicle ITS stations and, most importantly, to the Roadside ITS station.

4. Roadside ITS sub-system:

   The final element is the Roadside ITS sub-system. It is the connection point between the vehicle ad hoc network and the rest of the Internet. Like the RSU, it is placed in locations of high vehicle volume next to roads and intersections.

   This sub-system comprises a roadside ITS station and any proprietary roadside network that may exist. It connects to any vehicle ITS station and can provide ITS applications to the vehicle users. This station is also connected to a central ITS station, being the bridge that allows the central ITS station to provide ITS applications to vehicle users.

   The roadside station is made up of an ITS host with optional ITS interceptors. These interceptors can be any of the other types of functional components beyond the ITS host. The Roadside ITS sub-system is better depicted in Figure 6.

   The ITS router connects the station to the ITS vehicle station, the border router connects to an ITS central station and the gateway connects to any existing proprietary existing networks.

2.3 Communication Domains

With the main building blocks of both architectures laid out, we can divide communications into various domains. Both the C2C-CC and ETSI have different ways to tackle this question.

2.3.1 C2C-CC approach

The C2C-CC focuses on the VANET space and divides it into three areas: the in-vehicle domain, the ad hoc domain, and the infrastructure domain. The first represents the network inside a vehicle, composed of an

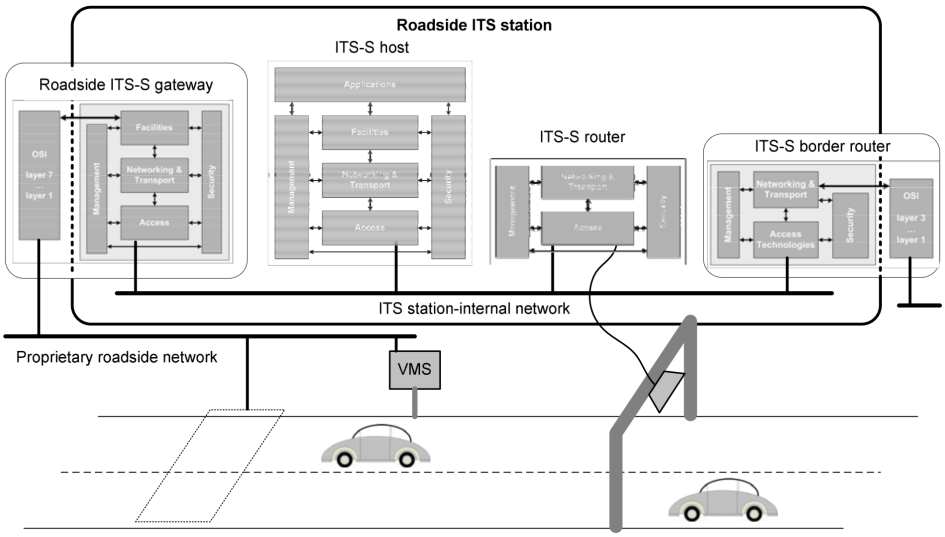


Figure 6: Roadside ITS station in a Roadside ITS sub-system [9]

OBU and the multitude of AUs connected to it. It encompasses all communications between the different AUs and the OBU. The ad hoc domain involves all ad hoc communications between different OBUs and between vehicles' OBUs with RSUs. The remaining components of the Internet that don't directly affect it, but help provide connectivity to vehicles, are included in the Infrastructure domain.

All of these domains, as well as the functional components of the C2C-CC architecture can be observed on Figure 7.

2.3.2 ETSI approach

The ETSI architecture proposes a broader categorization by establishing two domains. The first and most important is the ITS domain, since it contains all the elements and communication engaged by the ITS/ITSC standards. Everything else is encompassed by the generic domain, including everything in the rest of the Internet that might interact and communicate with the ITS domain.

2.4 Communication categories

In addition to communication domains, most authors use a classification based on communication types. This communication-oriented view of architecture aims to classify the multitude of types of communication that can occur between all the different entities that exist.

Different categories have emerged and evolved throughout the years as they have been used by most of the relevant researchers in the field. It is therefore difficult to find an origin for these concepts, and the following are most commonly used terms.

1. Intra-Vehicle communication This term encompasses all communications that occur inside the vehicle between different vehicle components.

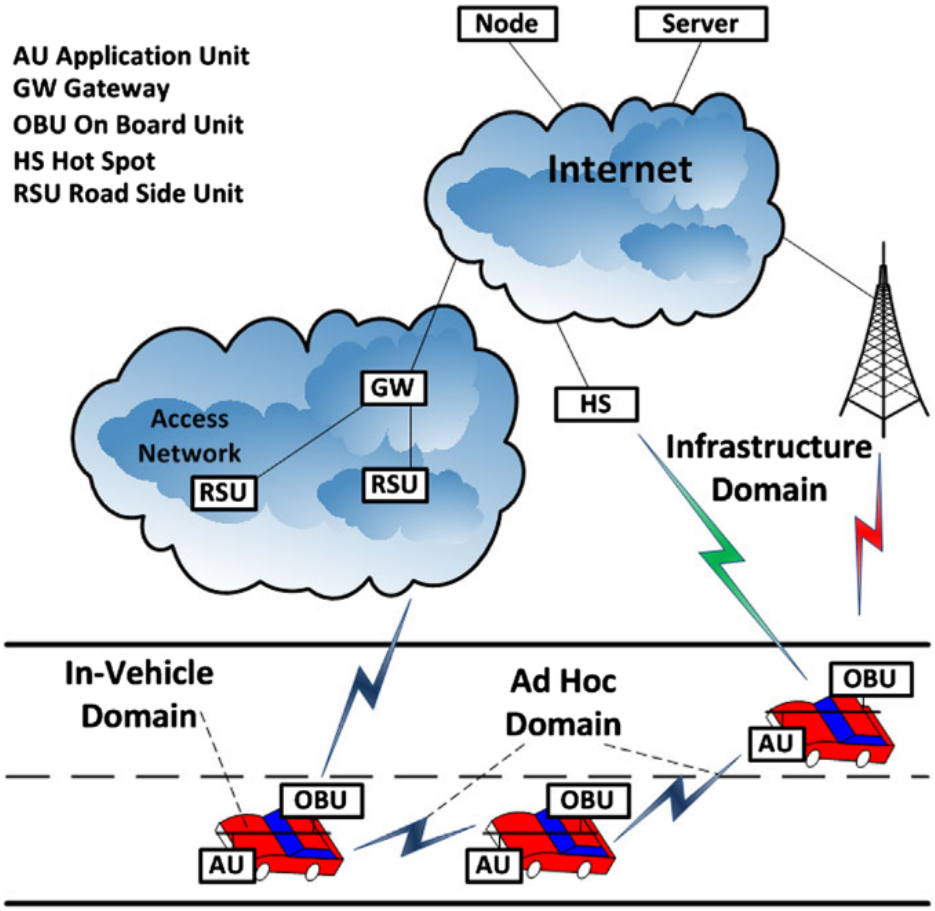


Figure 7: Communication domains in VANET as categorized by the C2C-CC [2]

2. Vehicle to Vehicle communication (V2V) communication V2V communication refers to the ad hoc communication between vehicles.

3. Vehicle to Infrastructure communication (V2I) communication V2I communication refers to ad hoc communication between vehicles and roadside infrastructure. These communications dont encompass all messages that get exchanged between vehicles and the roadside infrastructure, and only include messages whose source and destination are a vehicle and a roadside station.

4. Vehicle to Everything communication (V2X) The last type of communication is the most encompassing. V2X communication includes all communication that can occur in the VANET space between a vehicle and anything else. It is generally used to refer to the previous two types of communication.

2.5 VANET architecture

The architecture of an ITS host can be visualized on Figure 8. This section will dissect this architecture layer by layer in order to explain the functionalities and capabilities expected from it. The last layer, called Applications, will not be described as it represent ITS applications that use the services provided by the rest of the stack to connect one or more applications together and provide services to users[9].

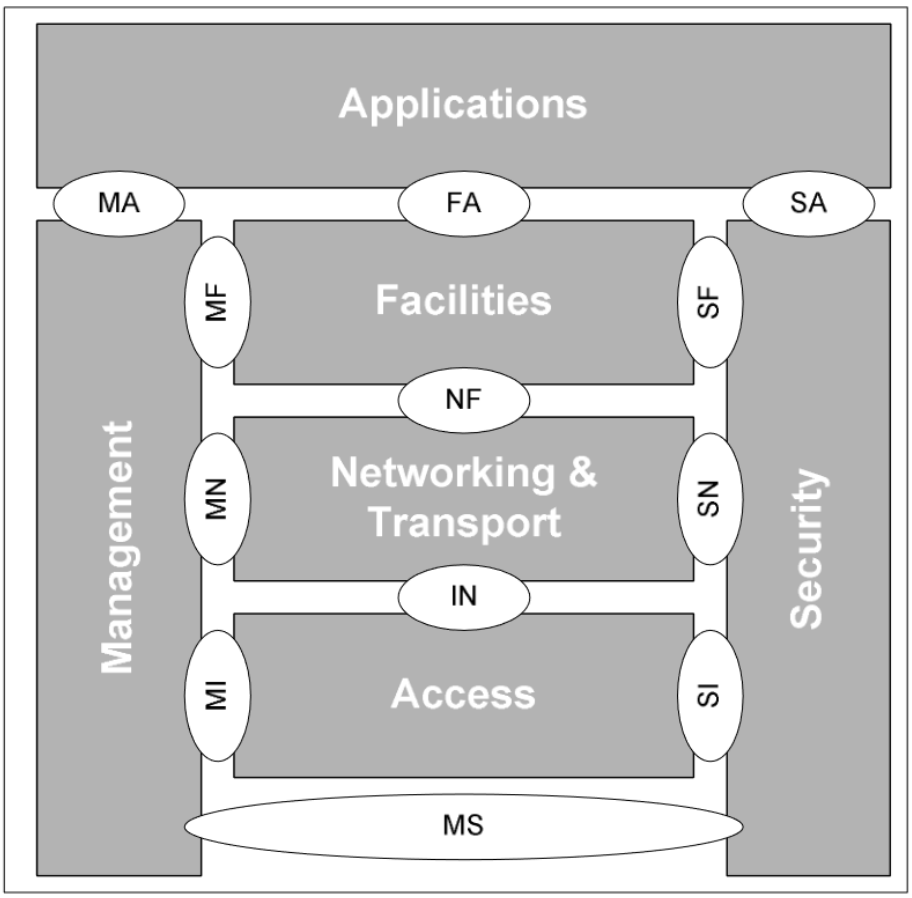


Figure 8: ITS station reference architecture [9]

The functionalities described in every layer are not necessarily function to be implemented in every instance of the architecture, as it depends on the specific implementation of the ITS station[9].

2.5.1 Access layer

The Access layer, which is illustrated in the architecture of an ITS host in Figure 8, corresponds to the first two layers of the OSI model, the Physical layer and the Data link layer[10].

Among all the existing Access layer technologies, the best suited to meet the necessary requirements for use in VANETs were the existing wireless access technologies[2]. These technologies ranged from long range signaling technologies such as Microwave and WiMAX to short range technologies such as Infrared or Bluetooth[11].

Over the years, most of these technologies have been studied and tested as possible solutions for use in VANET communications, and in most cases they were deemed unsuitable for the VANET environment. For instance, short-range communication protocols such as the examples above were discarded due to lack of range, throughput, and reliability and as a result, medium and long range communications took center stage.

While unable to fully address the demanding scenarios of VANETs, a few of these technologies showed

promise as possible solutions. Over the years since VANET research began, new standards and protocols have been developed based on these existing technologies, focusing on solving the problems posed by the characteristics of VANETs. In addition, these technologies themselves have evolved to meet the high data rates required by today's Internet.

Today, the IEEE 802.11 WiFi and Cellular technologies are considered as the two primary solutions for VANETs worldwide.

2.5.1.1 Institute of Electrical and Electronics Engineers (IEEE)-based technologies

The IEEE is an American organization dedicated to promoting innovation and technological excellence for the benefit of humanity. Founded in 1963, it is the world's largest professional society of engineers, scientists, and other technical professionals. IEEEs primary areas of contribution are the electrical, electronic, and computing fields[12].

The IEEE 802.11 family of standards, an Access layer technology, was considered a promising solution for VANETs from the start. Not only would the existing 802.11 standards be cheaper to adopt due to being already established technologies, but WiFi also already met many ITS requirements due to its compatibility with mobile environments[13].

Despite these significant advantages, the existing IEEE 802.11 standards were not able to meet all the requirements imposed by the unique characteristics of VANETs, necessitating the creation of a new variant that would be better suited for such scenarios. Thus, 802.11p was created as an adaptation of the existing 802.11 standards. Given its American origin and contribution to the development of V2X communication technologies, it is important to mention that IEEE 802.11p was conceived in the context of Wireless Access in Vehicular Environments (WAVE)[14]. The EU, while initially open to many different access technologies, eventually converged on a IEEE 802.11p based solution dubbed ITS-G5.

IEEE's main motivation behind the development of IEEE 802.11p was to enable effective communication capable of handling the rapidly changing environments of VANETs[14], while making the minimum necessary changes to the IEEE 802.11 Physical layer. Modifying the Physical layer is akin to a software update, while a Physical level amendment would require the design of an entirely new wireless airlink technology[14], which would be expensive and time consuming.

With this in mind, the IEEE began work on a variant of the 802.11 standard that would be feasible for V2X, capable of operating at speeds up to 200 km/h and ranges up to 1 km[1]. Experimental work began with the full deck of 802.11 technologies[6], but soon narrowed down to IEEE 802.11a because it operates at 5 GHz, which is closest to 5.9 GHz, the desired frequency for ITS operations in the US. This made it easier to configure devices to operate at the desired frequency[14].

IEEE 802.11p uses orthogonal frequency division multiplexing. The 802.11 family of standards offers three channel bandwidths of 5, 10, and 20 MHz, and while 802.11a uses the full 20 MHz bandwidth, the 10 MHz bandwidth was chosen for ITS scenarios. This is achieved either by halving the clock/sampling rate or by simply doubling all the standard orthogonal frequency division multiplexing timing parameters[13].

Reducing the channel bandwidth was done in an effort to increase the root mean square delay

spread[6]. Root mean square delay spread measures the time difference in seconds between the first and last signal components. These components represent different versions of the same transmitted signal, with the first to arrive coming from the shortest path which is the light of sight and the remaining coming from reflections and other interactions with the environment.

The standard 20MHz guard used by 11a proved sufficient to stop intersymbol interference between a radio and itself, so measures had to be taken to reduce the interference caused by multipath delays and by the Doppler effect. Halving channels width was the most simple and sensible solution to this problem[14]. Beyond reducing the signal bandwidth, the data throughput ranges were also reduced to 3 to 27 Mb/s instead of the original 6 to 54 Mb/s[6].

With the goal of enabling effective communications that can cope with rapidly changing environments, 802.11p communications needed to get faster in order to better utilize the sometimes narrow windows available for communication. Such a feat demanded a reduction in the overhead required before actual data transmission, which was achieved by simplifying the Basic Service Set (BSS) operations present in 802.11a[14].

The BSS represents a group of stations that are wirelessly connected to the same access point. It is created by an access point, and any device requesting to join it can exchange information with it after proper authorization. A BSS variant for ad hoc environments exists, called Independent Basic Service Set (IBSS), which works without the need for infrastructure. However, it also proved to be insufficient to deal with the peculiarities of VANETs.

As a solution, IEEE 802.11p uses an ad hoc mode called Out of Context BSS (OCB). OCB simplifies setup operations compared to its counterparts in other 802.11 standards by eliminating management procedures such as channel scanning, authentication, and association. This means that 802.11p has no setup time allowing stations to communicate with each other directly and immediately[15].

Removing all of these processes makes the network less secure. However, these security vulnerabilities are addressed by other standards in the ITS domain.

In Europe, ETSI created a solution for the access layer called ITS-G5. It not only defines the technologies and protocols to be used in communications, but also includes the frequency allocated exclusively for V2V and V2I communications for the whole of the EU.[16] ITS-G5 uses the existing 802.11p standards created for the American WAVE, adapting it for the European region.

In the US, the FCC was responsible for the allocation of the frequency from 5.850 to 5.925 in 1999. In 2003, the FCC divided the American band into seven non-overlapping 10 MHz channels with a 5 MHz unused band at the lower end. These channels were numbered from 172 to 184. This division also allows for operations in the 20 MHz channels 175 and 181, which are the overlap of two 10 MHz channels[17].

In the EU, the Electronic Communications Committee (ECC) is responsible for spectrum regulation and the European Commission is responsible for enforcing the regulated spectrum in all member states. The ECC is made up of radio and telecommunications regulatory authorities from all member countries[17][16].

The spectrum allocation for ITS-G5 began in 2005, with a recommendation from ETSI to the ECC. In its TR 102 492-1[18], ETSI recommended the allocation of 30 MHz in the 5,875 GHz to 5,905 GHz

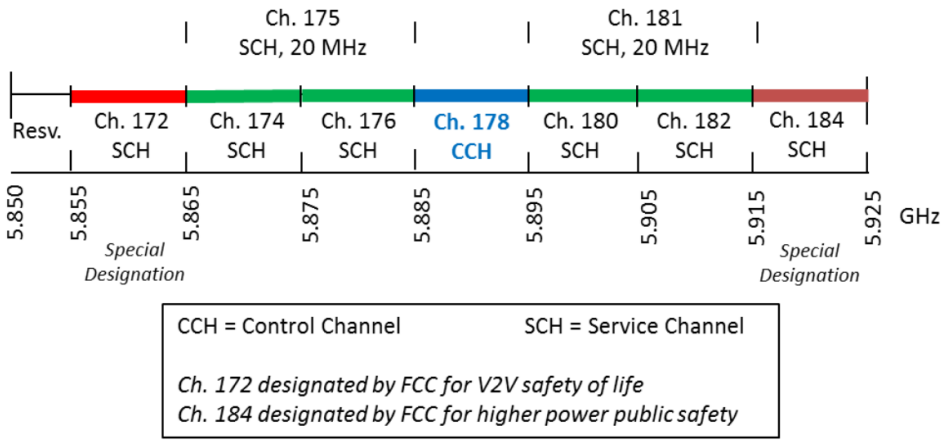


Figure 9: Channels allocated in the US by FCC for ITS [17]

band for safety applications, which would align with the US control channel frequency of 5,885 GHz to 5,895 GHz. This recommendation also expected the allocation of a future 20 MHz band from 5,905 GHz to 5,925 GHz for future ITS extensions[17][16].

ETSI based this recommendation on the allocation in the US frequency band, and also took into account the 5.8GHz toll collection band used in EU countries[17][16].

In 2008, the ECC accepted this recommendation and allocated the requested frequency range plus another 20 MHz frequency band from from 9,855GHz to 9.875GHz intended to be used by non-safety applications[17][16]. All of the spectrum involved for present and future ITS operations can be easily observer in Figure 10.

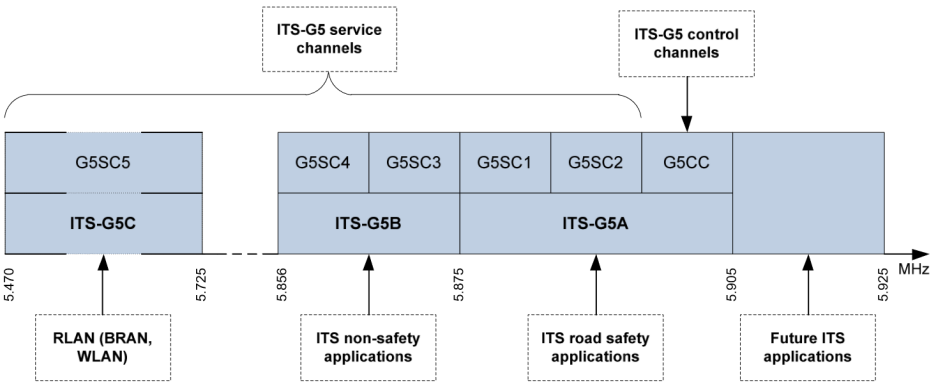


Figure 10: European ITS channel allocation [19]

Even though the American spectrum was taken into consideration, in the end the control channels of both the EU and US solutions were not in the same frequency, with the European channel being 10 MHz higher. This is not a total loss, as the same hardware can still be used in both countries requiring just a software change to work[17][16].

The different allocated frequencies have different standardized transmission power limits based on use cases and interferences on adjacent bands and on the more important control bands[15]. The specific restrictions for each 10 MHz channel can be observed in figure 11.

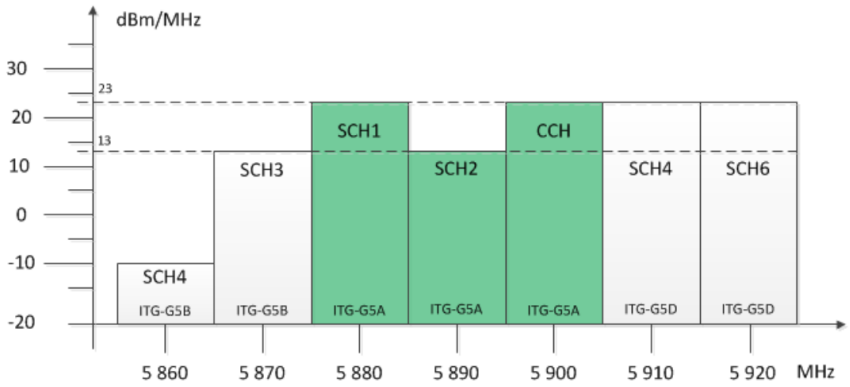


Figure 11: Power spectral limits on each ITS channel [17]

In recent years, the IEEE began work on IEEE 802.11bd, a successor to the 802.11p standard, which was approved in October 2023[20]. As a new standard, it is expected to take several years before it completely replaces 11p, and most automakers will continue to release cars with 802.11p for the foreseeable future.

This new version promises twice the throughput and longer range, while ensuring interoperability, coexistence, backward compatibility and fairness with existing OCB terminals. In addition, it operates beyond the 5.9 GHz band by utilizing the 60 GHz band.

2.5.1.2 Cellular-based technologies

Cellular systems have been used since the 1970s to transmit data over long distances via radio waves[11]. The 3rd Generation Partnership Project (3GPP) is an international consortium established in 1980 with the goal of developing technical specifications and technical reports for 3G mobile systems and is now the leading global reference for mobile standards and is responsible for the development of Long-Term Evolution (LTE) and 5G[21].

Although most work on VANETs has been based on 802.11p, Cellular V2X (C-V2X) has gained renewed support from academia and industry as this technology has matured[22]. Rival performance to the dominant IEEE 802.11p has led to Cellular technologies being considered by some organizations as a replacement for existing IEEE standards, so its status as the second most important VANET technology is not surprising.

In September 2016, the 5G Automotive Association (5GAA) was established as a global, cross-industry organization of companies from the automotive, technology, and telecommunications industries with its main objective to promote the use of cellular 3GPP standards in ITS scenarios[23].

Most researchers dismissed this new technology because its shortcomings made it seem inadequate compared to the strengths of 802.11. C-V2X's main drawback is its centralized nature. Cellular technology relies on a centralized infrastructure, which introduces latency by forcing all data to pass through a base station.

In addition, the large coverage area of cellular antennas can also pose an issue, as the coverage area

of an antenna is typically larger than the zone of relevance of safety messages. This makes broadcast an inadequate solution, as many vehicles would receive warnings not intended for them. Multicast can be introduced in an attempt to reduce this problem, but it introduces new delays from the overhead required to create these multicast groups[22].

On the other hand, Cellular technologies have several technical and economic advantages for their application in V2X communications. On the technical side, cellular technologies provide wide coverage, support high vehicle speeds, support a large number of connections to a single cell tower and can deliver high data rates. Economically, the wide use of these technologies allows the reuse of already deployed hardware for use in V2X[22].

These benefits, along with the renewed support from the industry created from 5GAA, motivated the 3GPP to begin studying the feasibility of LTE technology for supporting V2X communications[22]. In 2019, 3GPP began work on C-V2X efforts with Release 14, which made great strides in reducing existing drawbacks.

In order to work as a wireless access technology capable of serving all VANET test cases, this technology needed to ditch the necessity of using infrastructure and become ad hoc. Therefore, Release 14 established a new mode of communication, called direct communication.[24] This resulted in LTE and 5G supporting two relevant interfaces for communication in VANETs, the Uu interface and the PC5 interface.

The Uu interface is used for long range communication with cellular infrastructure in the commercial cellular spectrum. It uses existing LTE's large coverage to provide vehicles with all kinds of services[24].

The direct communication mode, also referred to as "Mode 4", utilizes the short range PC5 interface in the ITS 5.9GHz band and allows for direct exchange of information between vehicles without passing through a cell tower.

ETSI initially considered Cellular technologies as possible solutions, and at that time included 2G and 3G as possible access technologies in the initial ITS station reference architecture. As stated earlier, ETSI centralized the Access layer of its architecture on the 802.11p-based ITS-G5. Recently, however, reflecting on the efforts of 3GPP, 5GAA, and ITS implementation in other countries like China, ETSI made amendments to existing standards in order to achieve compatibility with C-V2X technologies[24]. This indicates that in the C-V2X architecture, the layers above the Access layer consists of the pre-established standards of different countries, demonstrating significant technological compatibility.

2.5.2 Networking & Transport layers

The Networking and Transport layer, as implied by its name, corresponds to the third and fourth layers of the OSI model.

layer 3 algorithms commonly deployed in traditional networks are unsuitable for use in VANETs[6], and even existing MANETs algorithms have proven to be inadequate for use in VANETs[3]. The characteristics of VANETs, as described in section 2.1, present several unique challenges not only to existing layer 3 algorithms, but also to Layer 4 protocols, the most important of which are high node mobility and variable node density.

The high mobility of nodes leads to frequent topology partitions, which results in highly unstable routes. Therefore, pre-determining routes in these conditions is often irrelevant and extremely complicated. Additionally, the algorithm must operate under conditions of both extreme congestion and isolation due to variable node density, further increasing its complexity.

Scholars have recognized the adoption of broadcast as the primary communication mode as a great solution to mitigate interference in high congestion scenarios. Nevertheless, this approach alone falls short of the desired outcome. For instance, in case of a crash, it is essential to rapidly disseminate security messages as the number of affected vehicles can rapidly rise. However, if every vehicle broadcasts simultaneously, it will lead to channel congestion, making it more difficult for the event to spread[6]. Therefore, novel approaches are required to curb message duplicates.

Although the challenges presented by these aspects of VANETs affect algorithm reliability, researchers have leveraged other characteristics of VANETs to develop novel solutions to the aforementioned problems. Besides the lack of power constraints, the predictable mobility of nodes allows algorithms to perform more effective link selection, which facilitates network management and opens opportunities to optimize the flow of information, ultimately improving routing protocols.

In Europe, researchers at ETSI developed new algorithms and defined new protocols in an attempt to find an adequate solution to the aforementioned challenges while taking advantage of VANETs new-found opportunities. As part of its architecture, ETSI introduced a new layer 3 protocol called GeoNetworking[25][26][27][28][29][30], a new layer 4 protocol called Basic Transport Protocol (BTP)[29] and a sublayer 3 extension called GeoNetworking-IPv6 (GN6)[30].

GeoNetworking is an ad hoc routing protocol that was developed solely for the transmission of safety related messages and therefore it provides the ability to forward packets without the need to exchange any signaling messages beforehand.

The GeoNetworking protocol uses the geographical coordinates of nodes as their address and utilizes location-based data to help make packet forwarding decisions.

By using a device's geographic location as its address, this protocol enables packets to be sent to all nodes within a specific geographical area. Nodes can effortlessly broadcast messages to an entire geometrically shaped area without congesting the entire network. This approach proves to be more efficient than broadcasting because it curbs congestion by minimizing transmissions to unintended destinations. Moreover, it makes the packet relevance area independent of the sender's range.

This protocol was built and optimized for multi-hop communication with geo-addressing, providing more technical features in application support than the alternative in the US. These capabilities come to a cost to protocol complexity and message overhead[31].

The GeoNetworking protocol was designed to be integrated with the BTP protocol. BTP is a transport layer protocol that is similar in function to User Datagram Protocol (UDP), functioning as a connectionless protocol[15].

ETSI has established the GeoNetworking extension GN6 as an alternative to routing packets through BTP. By using the GN6 sub-layer, Internet Protocol version 6 (IPv6) packets can be transmitted over the GeoNetworking protocol without the need to modify the IPv6 protocol[15].

2.5.3 Facilities layer

The Facilities layer encompasses layers 5, 6, and 7 of the OSI model, and its standardization defines application-specific functionalities[31]. The development of Facility layer messages is the responsibility of European institutions such as ETSI, European Committee for Standardization (CEN), and International Organization for Standardization (ISO).

All the messages defined by European standard makers serve safety purposes, and therefore run over the GeoNetworking protocol[31]. Multiple different messages have been defined over the years[15], which makes it difficult to list them all. However, CAMs and DENMs are universally acknowledged as the most significant in the context of ITS.

A CAM[32] is a periodic safety message that contains vital status information about the originating vehicle, with the main goal of enabling other vehicles to take appropriate preemptive measures to avoid potentially dangerous situations[2].

This objective is accomplished by exchanging data, including speed, location, direction, and additional non-safety application data with nearby ITS stations, allowing vehicles to monitor each other's movements.[31].

CAMs begin transmitting as soon as the vehicle enters a safety-relevant context, which is considered to be anytime the vehicle is in operation.[15] The transmission rate of CAMs is subject to specific rules, including both maximum and minimum transmission times between CAMs, relevant changes in position, direction, or velocity, and congestion in the wireless channel[15].

CAM fields are shown in figure 12. It includes an obligatory ITS Packet Data Unit (PDU) header, Basic container, and High frequency container, along with optional Low frequency and Special vehicle containers.

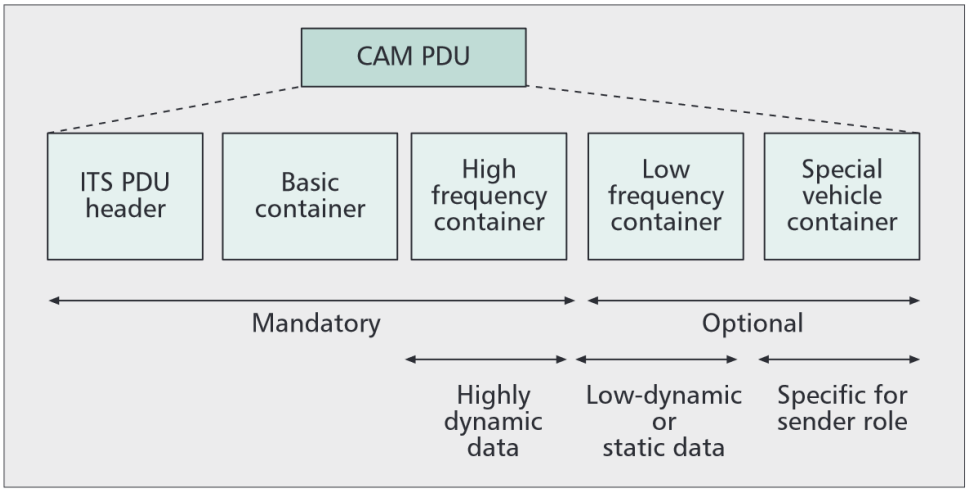


Figure 12: CAM structure [15]

This structure allows for a high degree of flexibility in message format, allowing messages to adapt more effectively to their environment, thereby minimizing congestion on wireless channels[15].

A DENM[33] is an event-driven safety message that can be triggered by an ITS station in the event that hazardous conditions are detected. ITS stations are capable of detecting a wide range of events and this message type is employed to describe such events to other ITS applications.

DENMs serve the purpose of warning ITS applications to a detected event within a designated geographical area[15]. This type of messages are only relevant in a specific location and therefore are only disseminated in that specific geographical area. These geographical areas are the ones defined by the GeoNetworking protocol.

As an event-triggered message, DENMs are considered high-priority messages, as ensuring a quick delivery is crucial to diminishing the consequences of the events that triggered their generation[2].

Throughout the lifespan of the DENM, a number of techniques are employed to ensure the dissemination of the message in its relevant area. DENMs are repeated, generally at a lower frequency than CAMs, with the intent to allow vehicles entering the relevant area to receive said information. Similarly, if the originator ceases to transmit the DENM message for any reason, another ITS station can replace the originator and continue to transmit the message. DENM messages can also be canceled by the ITS station that created them[15].

DENM fields can be observed on figure 13. The ITS PDU header and the management container are mandatory, while the situation, location and a la carte container are all optional. These message fields mainly contain information about the relevance area of the message, the type of event and the time in which the message remains relevant.[2]

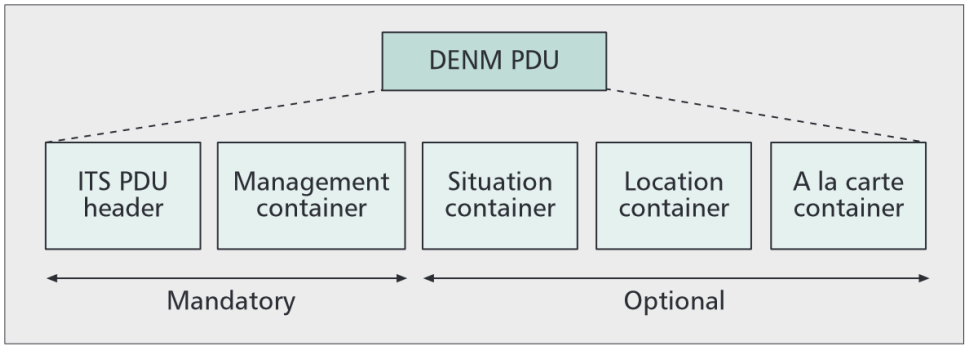


Figure 13: DENM structure [15]

2.5.4 Management Entity

Network management is a critical component of network maintenance as it ensures that a network operates as intended. The unique challenges presented by VANETs contribute to a significantly more unstable network environment than usual, making management even more essential. With this motivation in mind, ETSI incorporated a management entity into its architecture 14.

The ITS management entity is responsible for both configuring and operating its ITS station, while also overseeing cross-layer information exchange between multiple layers. It contains interfaces to every other component in the ITS station and a management information base. [27]

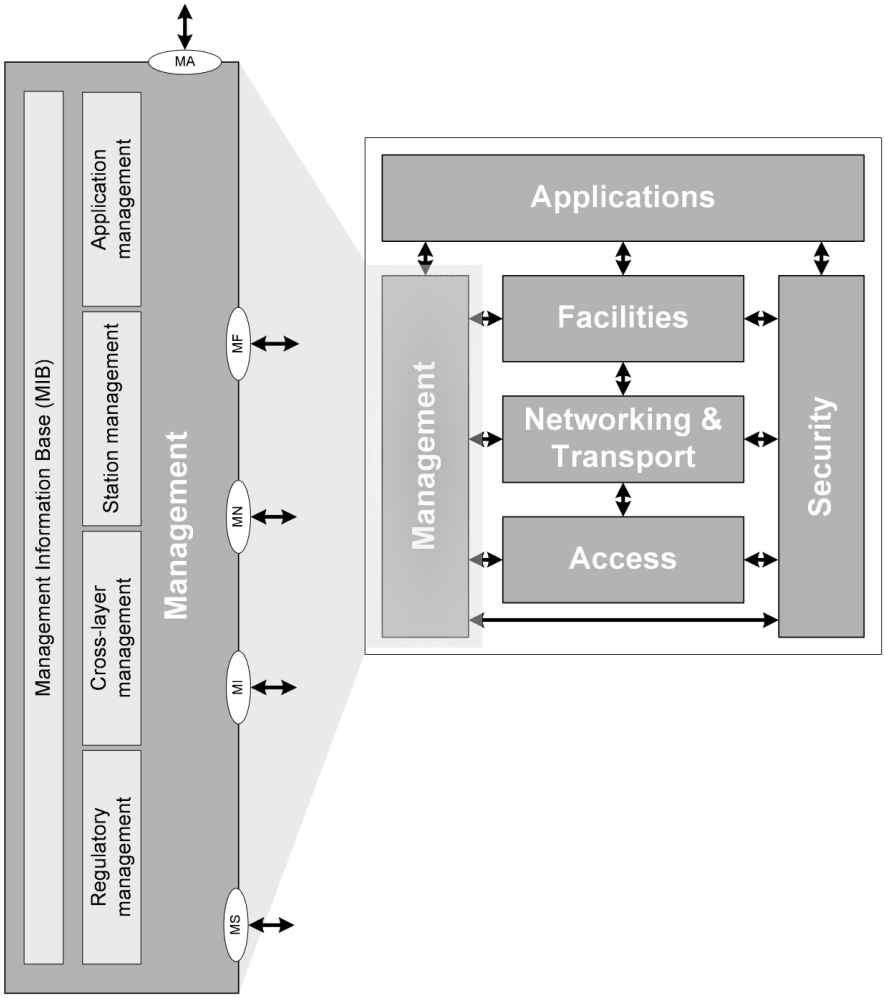


Figure 14: ITSC management entity as part of the ITS station reference architecture [9]

Decentralized congestion control is a cross-layer ITS functionality that is coordinated by the management entity and is one of its most important responsibilities. Its main purpose is to control congestion in the channel by managing the amount of messages exchanged, the transmission power and any other useful parameter. The changes to these parameters, being orchestrated by the management entity, are made based on various information extracted from different layers. [15]

2.5.5 Security Entity

Security in VANETs has been one of the biggest concerns and one of the biggest obstacles to its deployment. It is crucial to protect the ITS domain from malicious attacks and network abuse, and as such it has been a high priority for researchers and developers to address security threats prior to deployment. Vehicle messages can contain a large amount of sensitive information, such as vehicle trajectory and location data. This information can be used to deduce the driver's identity, activities, habits, and so on. This type of information must not only be kept private under EU law, but could also be used by bad actors for extortion.[3] [34] Attackers could also exploit safety messages for their own benefit or to the detriment

of others. For instance, greedy drivers could broadcast false traffic alerts to reduce traffic on their own routes. In a more extreme case, robbers or terrorists could abuse traffic alerts to clog roads and thereby delay emergency vehicles. [34] With this motivation in mind, researchers have laid out several security requirements VANETs must meet. The following list contains the most important ones, based on the works of [35] [34]:

1. Authentication: this security requirement ensures that a recipient can identify the sender of all messages received, thereby ensuring that each message is generated by an authenticated user.

2. Non-repudiation: sometimes called auditability, it is a tricky to enforce but essential security requirement. Non-repudiation ensures that once a message is sent, the sender can't deny ownership of the message. In VANETs, this requirement is essential for identifying compromised users.

3. Integrity: this security requirement ensures that the message remains unchanged during transmission.

4. Confidentiality: a security requirement for any type of network, and VANET is no different. Some messages contain important information and therefore should only be accessible to the sender and receiver to prevent eavesdropping.

5. Availability: one of the most important concerns in VANETs, this requirement seeks to ensure the availability of the wireless channel, as a message that is delayed by seconds becomes useless.

6. Access control: this property creates different levels of access for different entities. Access control bars users from accessing any information or sending message types they are not allowed to. Distinguishing multiple levels of access allows for a higher degree of network control and is essential to stop known bad actors from exploiting network dynamics.

7. Privacy: Privacy is the ability of users to hide their personal information from the rest of network users. Implementing mechanisms to protect the privacy of all drivers is a top priority to ensure driver privacy, with the main goal to provide location privacy. Anonymity is the process of hiding one's identity, which is extremely important to achieve privacy.

8. Data verification: this property is essential to avoid false messaging as it allows all messages to be tested by their time relevance. This is necessary to avoid replay attacks in the network.

9. Physical Security: as vehicles are a widely spread technology that will be distributed indiscriminately, it is important to implement hardware security in order to avoid tampering and compromising of the vehicle.

Authentication and privacy are desirable properties of VANETs but ensuring anonymity while enforcing network liability in VANETs is a contradictory property in securing vehicular networks. This comes from the necessity to protect drivers' privacy while being able to track down attackers. [14] [34] Another important

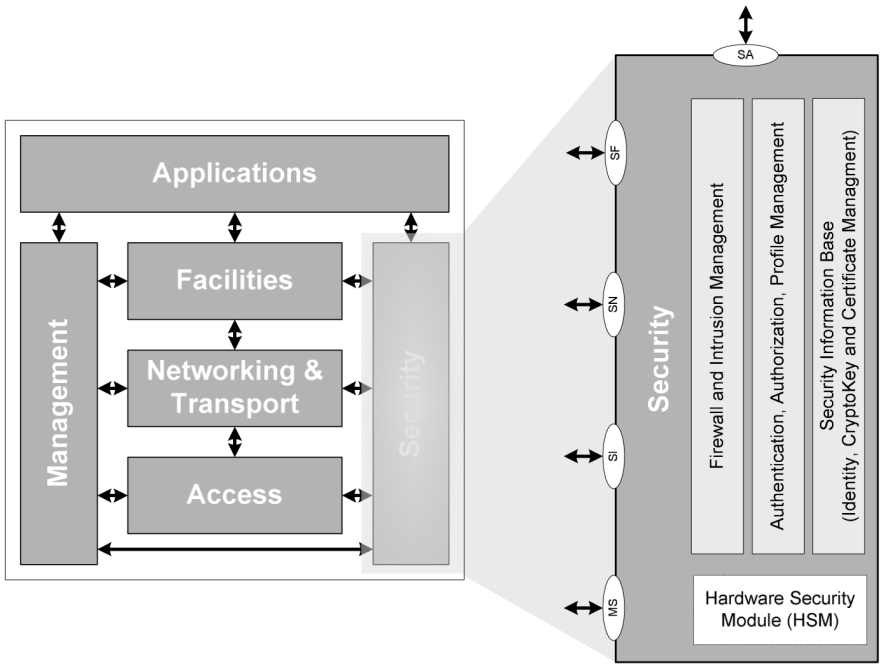


Figure 15: ITSC security entity as part of the ITS station reference architecture [9]

consideration to take is that security measures can create performance problems, as they add overhead time in communications and can significantly degrade message quality. [6] [36]

The Security Entity provides a plethora of services to ensure security and privacy. These include a multitude of secure messages at different layers of the communication stack, management of identities and security credentials, and other aspects relevant to secure platforms such as firewalls, security gateways, and tamper-proof hardware. [27]

ETSI developed an ITS communication system that relies on indirect trust relationships, which are built upon trusted third party certificates. It is a solution to the privacy problem and results in the implementation of a so-called authority hierarchy. This hierarchy is composed of manufacturers, enrolment authorities, authorizations authorities and the ITS station.

When a car wishes to begin communicating, it must use its canonical credentials given by the manufacturer to request valid enrolment credentials to an enrolment authority. Then, using the enrolment credentials it must request an authorization authority for authorization credentials. These last ones are the ones that will be used for communications, and as such in order to protect a driver's privacy are only valid in a time frame. After these expire, the ITS host must request new authorization credentials to the authorization authority. From this example, we can observe that the enrollment authority validates that the vehicle has a valid ITS station and that the authorization authority uses the authorizations given by the enrollment authority to allow the ITS station access to the ITS domain and to utilize a specific type of applications, service or privilege. This process can be observed on Figure 16. [38]

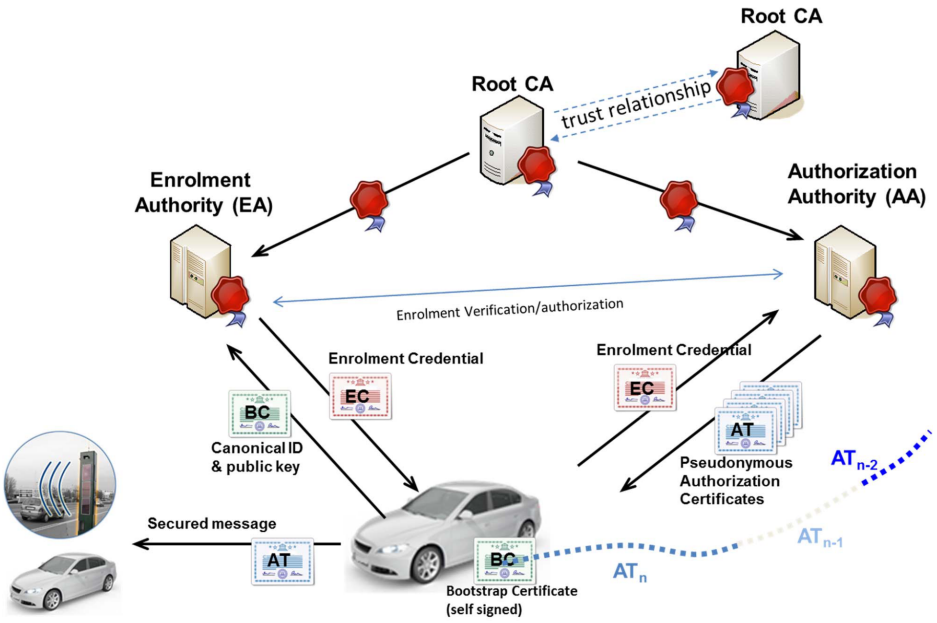


Figure 16: ITS Security Certificate Management System [37]

2.6 Applications of VANET

The motivation behind VANET research and development is to provide wireless services to both drivers and vehicles. The goal of these service applications is to improve driving safety, passenger comfort, and traffic efficiency. Hence, VANET applications can be classified into safety and non-safety applications according to their intended goal. Safety applications are considered the main driving force behind the development of VANETs[3] as the main goal of these types of applications is to decrease both road fatalities and road pollution[2]. Safety applications can be divided into two categories: Cooperative road safety and Cooperative traffic efficiency. Certain applications have the potential to improve both of them. Cooperative road safety applications harness the wireless communication capabilities of vehicles to provide useful information to the vehicle and driver, with the ultimate goal of mitigating the likelihood and severity of accidents. Therefore, any application intended to enhance the safety of individuals is classified as an Cooperative road safety application. Examples of such applications are notifications for upcoming hazardous locations and approaching emergency vehicles. Cooperative traffic efficiency applications aim to make road operations more efficient. These efforts can both greatly reduce unnecessary carbon emissions and save the time of drivers. Examples of such applications are off street parking information and green light optimal speed advisory. VANETs could also become a crucial step in the journey to fully autonomous vehicles, by exploiting expected advanced cooperation systems to exchange sensor information and status information among vehicles. These future applications also fall under the cooperative traffic efficiency umbrella. Some of these envisioned safety VANET applications require a certain percentage of road-wide deployment to become useful[1]. Therefore, ETSI has defined a basic set of applications to be deployed as ITS systems mature, with the aim of deploying them simultaneously at that time. These applications are commonly referred to as Day 1 applications and aim to provide societal and economic

benefits to both the private and public road transportation sectors. Remaining safety applications have been grouped into Day 1.5 deployments, with the goal of improving and extending Day 1 applications. These applications can be further divided into bundles, which can be viewed onl. The remaining applications fall under the category of non-safety applications. Such applications, also known as co-operative local services and global internet services, are those designed to enhance the comfort of drivers and passengers by enabling access to internet services from their vehicles[2]. Based on this, any application that provides value-added services is considered non-safety applications[6]. All types of entertainment and information-based applications, such as music, movies, podcasts, online games, or instant messaging platforms, are examples of the applications covered under this category. Essentially, any application that is hosted on the World Wide Web and is accessible via the IP stack falls into this category. Non-safety applications should not interfere with safety applications, and thus they use different physical media and protocols[1]. In detail, non-security applications are typically delivered using IPv6 over C-V2X, while safety applications exclusively use GeoNetworking.

2.7 Future trends and challenges in VANET research

Predicting a date for when the full potential of the VANET technology will be unleashed is incredibly difficult but it is a case of when, not if, as the benefits of VANETs remain attractive and a lot of work has already been done.

Current and future research seeks to update current technologies in use, such as the aforementioned development of 802.11bd 2.5.1. Additionally, efforts are being made to augment this technology with other technologies such as SDN, Edge Computing, and Artifical Inteligence to take it to the next level[39].

The deployment of VANET infrastructure in Europe is currently underway, with projects such as Trans-European Transport Network. In line with the European Green Deal and the renewed focus on the climate crisis, VANETs are expected to help reduce traffic emissions by increasing traffic efficiency. Besides, the goal to reduce traffic fatalities to 0 by 2050 remains a top priority[40]. Figure 17 provides a comprehensive representation of the plans for VANETs in the EU.

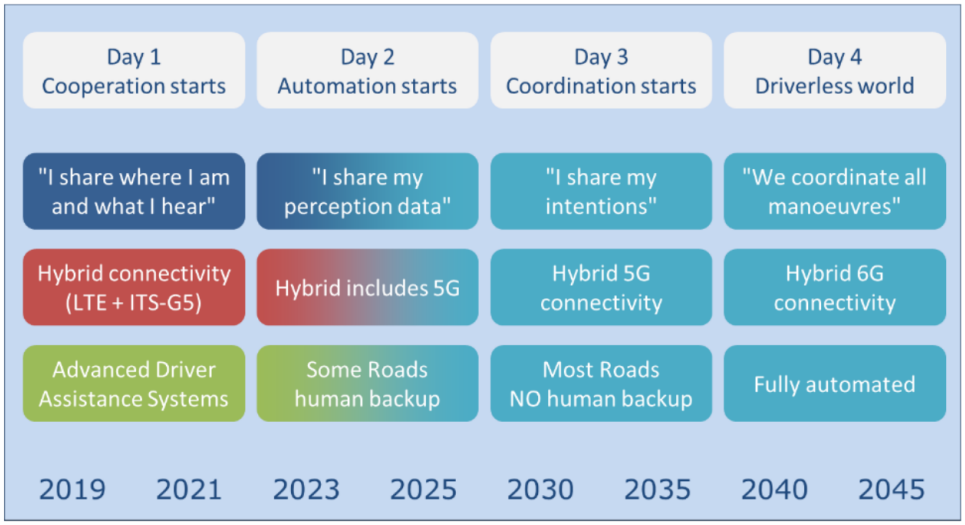


Figure 17: ITS strategy for the EU [40]

3

Software Defined Networking

SDN is a networking paradigm that simplifies network management and promotes innovation through software-based approaches, rather than traditional hardware-based configurations. SDN has the potential to shape the future of the Internet and is therefore often referred to as a "radical new idea in networking" [41].

SDN dates back to the 2000s, and its concepts can be traced as far back as the 1990s [42], so this technology is not exactly new. The core ideas and motivations behind SDN have been worked on and evolved over the past thirty years, with several previous efforts already being undertaken to promote network programmability[42]. Even though it is not the first, SDN is considered the most promising as its uniqueness stands from providing programmability while offering to greatly simplify network devices[43].

In 2011, the Open Networking Foundation (ONF) [44] consortium was founded with support from major players in the technology industry, including Google, Facebook, and Microsoft. ONF develops open standards for SDN technologies to promote the SDN vision. It is currently the leading consortium in this field.

The information in this paper on SDN is largely supported by the book "Software-Defined Networks: A Systems Approach"[45], authored by leading researchers in the field and cited by the ONF itself. Before delving deeper into this topic, it is worth emphasizing that SDN is an approach to networking, rather than a point solution, so while some of the most important and widely adopted technologies will be mentioned in this chapter, they should be viewed as an attempt to achieve the SDN vision.

3.1 SDN definition

Traditional networks have a highly decentralized structure, which was essential for the early Internet to achieve high network resiliency. As a consequence, Internet devices are vertically integrated, meaning the network control logic is directly coupled to the underlying forwarding hardware. This leads to routing decisions being made by complex programs running on each device through the definition of low-level policies[46].

For a long time, researchers and developers have acknowledged that changes in the way the Internet works, nominally in its structure, were necessary to simplify and automate network management. These

changes came to take place as the technology known as SDN, which breaks down the vertical integration of devices[47].

3.1.1 Main principles of SDN

SDN technology nowadays is rather an umbrella term, as many different researchers have their own nuanced definition arising from different levels of implementation of the same SDN vision. Such discrepancies in the SDN definition is acknowledge by [45] and can be visualized in [42], [47], [41], and [46]. The early SDN proposal stood on three main principles, which are still relevant to this day and are as such:

- First and foremost this technology is characterized by the decoupling of the control plane from the data plane.

  The control plane can be considered as the "brain" of the network. It is responsible for commanding forwarding operations in the whole network, making decisions on how to handle network traffic.

  The data plane is the remaining forwarding hardware that follows the commands set by the control plane. By removing the decision making components from network devices, it makes them simple forwarding devices.

  Network intelligence is thus removed from individual devices, transforming all networking equipment into simple packet-forwarding switches.

- The second principle of SDN requires a logically centralized control plane that manages the data plane through well-defined Application Programming Interfaces (APIs). This means that a single entity, commonly known as the controller, oversees all of the network.

- Lastly, the controller must have the power to control network behavior through the programming of network functions. SDN must allow software developers to harness and utilize network resources similarly to computing resources and storage.

To sum up, and as can be seen in Figure 18, SDN requires the decision making elements of a network are decoupled from forwarding devices and logically centralized in a single entity, which is a programmable controller.

As a consequence from these principles, an SDN switch can function as any traditional Internet device, such as a router, switch, firewall, network address translator, or even a combination of these.

3.1.2 Recent advancements

These three principles remain relevant today and form the de facto definition of SDN. However, in recent years, new efforts have been made beyond this initial definition to further fulfill SDN's original goal. These efforts aim to increase programmability to the data plane, allowing network operators to fully customize all traffic in the network.

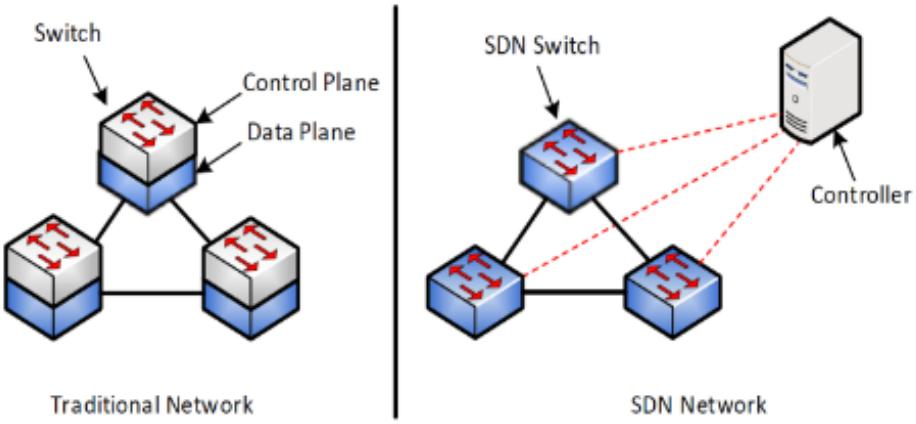


Figure 18: Traditional network view Vs SDN Network view[48]

Some articles refer to these recent efforts as next generation SDN [49][50], while predicting further and further coupling of SDN with network virtualization.

Being one of the first use cases found for SDN, Network virtualization traces its roots to the 90's, around the same time the idea of programmable networks was surging[51] [42].

Network virtualization enables the creation of network topologies decoupled from the physical topology. This allows multiple virtual networks to share the underlying physical infrastructure, functioning seamlessly over the same physical network[47]. The advantage of network virtualization is that it allows each individual virtual network to be simpler than the real network topology, granting increased flexibility.

Network virtualization and SDN are two distinct technologies that do not require each other to exist. However, SDN technology enables network virtualization, effectively creating a mutually beneficial relationship between the two that has grown much closer over the years[42].

Another way to interpret these recent SDN developments is to divide efforts into two phases. In the first phase the control plane is centralized and opened for operators, for more network control, and in the second phase the data plane is opened, allowing network operators to better control packet processing in the network.

In reality, SDN implementations may mean more or less than the original definition depending on the involved stakeholder that is deploying the technology, as each network operator has the freedom to fulfill this view however they wish[45].

From this, and from the fact that SDN is a disruptive technology, hybrid solutions that aim to harness some of the SDN benefits while maintaining the status quo in the industry have appeared[45]. These solutions mainly appear as attempts from manufacturers to maintain their established business models.

Lastly, it should also be noted phase 1 SDN was largely impelled forward by Openflow, while phase 2 is marked by the development of P4. Both these technologies will be further discussed in Section 3.3.

3.2 Motivation behind SDN

The Internet currently faces numerous challenges, ranging from expensive, complex network structures that are difficult to manage and optimize, to significant barriers to the development and deployment of new network design ideas. The networking research community and industry have recognized that these challenges require a novel approach to network structure. SDN promises to solve these problems by providing programmability, flexibility, improved performance, and support for implementing and testing new network ideas.

This section explores the benefits of SDN to demonstrate its usefulness and explain its motivations. As this thesis will not delve into the SDN disadvantages, it is important to keep in mind that SDN is not a universal solution for all networking issues. For example, in high performance networks, where each millisecond in packet processing is precious, the benefits of SDN might not be worth its small drawbacks, as a vertically integrated device is theoretically faster[41].

3.2.1 Network innovation

New ideas in networking have had increasingly less impact in the real world. This phenomenon is commonly referred to as "Internet ossification" and it means that the Internet has become static, like a fossil[41]. Most researchers consider this to be the greatest challenge to networking in the last three decades.

Today's networks are dominated by proprietary equipment. The distributed nature of the Internet already introduces a great deal of complexity, but these components must also support an extensive set of features while maintaining network flexibility, dynamicity, performance, and efficiency[42]. This results in extremely complex network systems that require a tremendous amount of effort from network engineers while being extremely dispendious.

This status quo causes new networking features to require a long and arduous process with immense implementation, experimentation, and deployment problems. Current networking devices and their maintenance also doesn't come cheap, with long monetary returns for manufacturers in investment cycles. These facts hamper innovation by inducing network developers to delay the introduction of new network features until they are widely requested[46][51].

Another important factor is the reluctance to test new ideas for fear of disrupting existing traffic. The internet has a massive deployment base and has long been considered as critical infrastructure[41], so network ideas are not tested for fear of disrupting current services. Consequently, experimentation scenarios are rarely possible and conducted in simplistic scenarios, which diminishes confidence in their results[43]. Realistic testing scenarios are necessary to gain the confidence needed for widespread acceptance[52].

The perfect example of internet ossification can be seen in the transition from IPv4 to IPv6. This transition started more than two decades ago and still remains incomplete, while being a simple protocol update[51].

Network architecture should encourage network innovation, rather than stand in its way. Use cases for

the internet in the form of internet applications are continuing to transform and evolve and it is impossible to predict future application requirements and perfectly design future networks to meet those requirements. Instead, networks should evolve and adapt in the face of surging problems. [43]

The primary motivation behind programmable networks has been to enable innovation and minimize the barrier of adoption to new network services[42], making this an integral goal of SDN. As a methodology, SDN provides a great platform for development with new ideas, techniques, and designs, encouraging experimentation.

The separation of the control plane from the data plane abstracts the definition of network policies from the underlying implementation in the switch hardware. This abstraction separates network problems into smaller traceable pieces, which enables innovation of individual components[51]. As an example, technological developments in the control plane can be made independently of the data plane, enabling faster technological development in both planes[47]. SDN is also based on open standards for communication between planes, which further stands to enable greater innovation through interoperability.

Through network programmability, SDN enables the development of new ideas using these hardware abstractions, which allows the programmer to avoid dealing with low-level hardware details. This enhances convenience and flexibility, which in turn makes network application programming more accessible to a wider audience.

Lastly, the programmable controller also benefits from a network-wide view and control, meaning it can instantly and seamlessly deploy new network applications, such as protocols or policies, across the entire network with minimal network downtime. This ability to easily update devices enables devices to be incrementally tested and refined as feedback is received. In traditional networks, these processes required the installation of expensive hardware and manual updates to individual devices. [43]

3.2.2 Network cost

Routing devices are extremely costly to purchase and maintain, especially given a handful of companies have tight control over this market. The complexity of the devices prevents new competitors from entering, which is the main reason for this dominance. SDN promotes a more affordable alternative to these traditional devices through its open source approach, which allows any hardware device to be used as a networking device, commoditizing networking hardware. [41]

In an attempt to address the lack of functionality in current network devices, traditional networks are filled with a myriad of specialized components and middleboxes performing a variety of different functions[42][51], further driving up network costs. Examples of these middleboxes include firewalls, load balancers, network access controllers, and network address translators [41]. In SDN, these devices are replaced by software programs which are implemented by the network controller as network-wide policies, thus reducing network costs.

SDN can also reduce costs by optimizing energy consumption. In large-scale network scenarios, such as data centers, the cost of energy consumption from networking equipment may seem small, but it still

constitutes a significant expense. SDN allows networks to dynamically power on and off devices based on expected network traffic, reducing energy consumption and costs[41].

3.2.3 Network control

Today's computer networks are often too large, complex, and heterogeneous for conventional methods of configuration, optimization, and troubleshooting to be effective[43][51]. Network management becomes very difficult due to this complexity, which in turn hinders efforts to optimize network performance.

Network operators are required to configure their networks for a wide range of applications through manual translation of desired high-level policies into low-level commands, while coping with changing network conditions. This task is further complicated by vendor-specific configuration interfaces, which are often complex, limited, and can vary from model to model, even within the same vendor[42].

Beyond being extremely tedious, manual network configuration is extremely error-prone. Configuration changes can therefore cause network instability, leading to outages, security flaws and performance disruptions[41]. Additionally, configuration errors require a significant amount of effort to troubleshoot.

Efforts to improve network performance are also severely limited, with current optimization efforts typically confined to small regions of the network or specific network services, efforts that are further constrained by the lack of global network information[43].

Another major motivation for SDN is to shift network control from vendors to operators, and ultimately from operators to users[45]. In today's Internet, the aforementioned complexity of networks impedes this goal so SDN is designed to make networks more flexible, open, and simple to configure.

Networks are made simpler first by breaking down networking into smaller pieces through the abstraction gained from the decoupled control and data planes. This allows network managers to implement the desired network outcome in the control plane without having to deal with complex lower-level instructions.

Secondly, the complete and centralized view of the entire network helps simplify network management[46], given that current network policies are extremely complex due to the need to cope with the distributed nature of the Internet. SDN enables a more consistent application of network policies throughout the entire network, rather than on individual devices, making policies more uniform and easier to implement. [41]

Finally, SDN networks are also designed to be highly programmable, granting unprecedented control that can leverage the aforementioned benefits to enable simple, flexible and agile management of network resources. This in turn simplifies the design, deployment, and management of complex network environments and allows for more effective use of network monitoring through the ability for the entire network to homogeneously adapt to any necessary reconfiguration.

These evolutions in network structure enable new powerful solutions to classical networking problems such as end-to-end congestion control, energy efficient operations, load balanced packet routing, quality of service support, and data traffic scheduling [43]. In contrast to traditional networks, where these would be implemented by deploying specialized middleboxes at precise locations in the network, SDN allows them

to be developed in software as network applications. This capability is what makes physical middleboxes obsolete, as noted above, and contributes to simplifying the deployment of new network services.

Network applications can be tailored for their purpose, ensuring that hardware resources are not wasted on irrelevant functions[46]. They can be easily shared and deployed as third-party "apps" on any network[41][51], making more efficient implementations readily available and thus improving network performance.

Implementing automated network management is another major SDN use case[43][53]. Intelligent network control can reduce reliance on network administrators by enabling network applications to quickly and easily reconfigure the network to meet changing requirements and make real-time adjustments to automatically optimize network performance. SDN enables network performance and traffic flow monitoring, enabling automatic and dynamic network configuration with centralized validation that automatically optimizes network performance based on desired high-level network policies.

SDN enables networks to be more application-aware by allowing applications to communicate with the controller and request network services[47]. An example of this is transport with quality of service insurances.

3.3 SDN architecture

The SDN architecture divides networks into abstracted layers, inspired by computer systems, each with its own unique purpose[51]. It consists of three main layers: the data plane, the control plane, and the application plane. These layers are linked by the Southbound API, and the Northbound API. Variations of this SDN view are numerous and will be explained when relevant.

A fundamental design principle of SDN, which is a cornerstone of the architecture, is to emphasize standard, open interfaces between the different planes. The purpose of this structure is to promote compatibility and interoperability between different data, control and application plane projects so that all components can work together interchangeably[51]. A result of this open approach to development is that SDN relies on the aforementioned open APIs and a collection of software components that support those APIs, rather than any specific protocol stack[45].

When surveying the SDN architecture, it is best to approach it and each of its components from the bottom up. This section will therefore cover all possible layers and components in SDN, starting from the data plane and ending on the application plane. Specific technologies will also be mentioned as they represent the current state of the art. However, it is important to keep in mind that there exist several alternatives, and these technologies are constantly evolving, with different implementation approaches that may become more prevalent in the future.

3.3.1 Data plane

The data plane is the first and the lowest layer of the SDN architecture and is made up of networking infrastructure which in traditional networks are switches, routers and any other middlebox appliance.

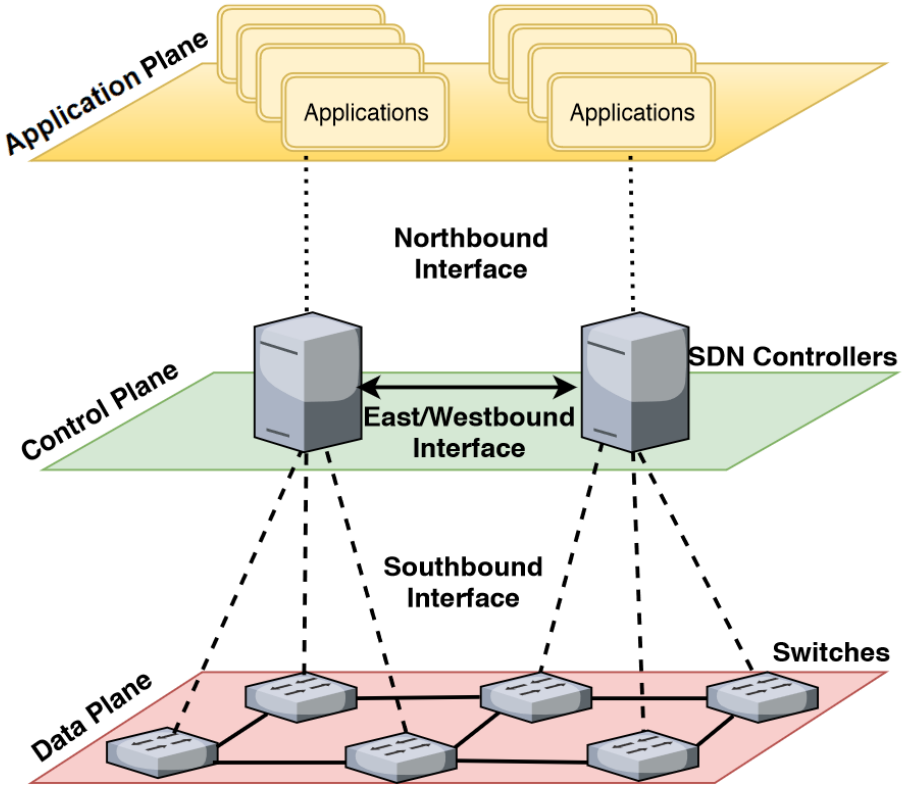


Figure 19: Layered view of SDN Architecture[54]

These distributed forwarding devices are interconnected with each other through traditional means like wireless radio channels or wired cables. Unlike in traditional networks, however, these devices don't have the autonomy to make networking decisions and instead follow the instructions given by the controller[51].

This last property means that in SDN, all network equipment is referred to as "switches" or "bare metal switches"[45] because without embedded intelligence in them, forwarding hardware becomes interchangeable and different names for equipment become redundant. On top of the hardware, SDN switches also have software running on every individual device. This is distinct from the control plane software and its purpose is to abstract the switch hardware, communicate with the control plane and implement forwarding decisions.

The data plane is responsible for receiving, processing and forwarding packets based on rules set by the controller, implementing the packet processing policies defined by the control plane[52] [43]. These devices may also gather and store network status information, including network topology, traffic statistics, and network utilization, to later transmit to upper layers[43]. This data can then be used to verify network behavior, monitor network performance and handle failures, all of which improve overall network health[46].

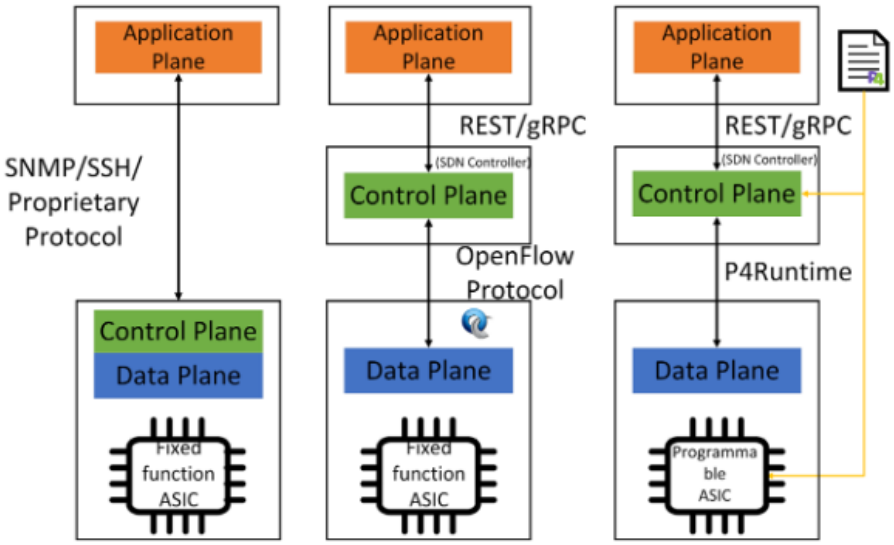


Figure 20: Evolution of SDN[49]

3.3.1.1 Data plane programmability

SDN switches can be divided into two categories based on whether they are programmable or static[45]. This classification is driven by changes not only in behavior, but also in hardware. Static devices use fixed-function chips that process headers based on fixed protocols that cannot be changed. Programmable devices, on the other hand, use programmable chips for dynamic processing.

The programming of data plane devices is a relatively recent development, which has led to the two phases of SDN mentioned in section 1. Figure 20 depicts these two phases, accompanied by a comparison to traditional networks, for a better understanding of this transition.

These efforts ultimately allow for the behavior of the data plane to be modified. Data plane programmability refers to the ability of a network operator to systematically and extensively reconfigure the processing logic of a switch as desired[46][55]. Simply put, data plane programming enables users to implement their own data plane algorithms on any switch.

Most importantly, data plane programming can unlock complete control over network packet processing. This allows customization of network protocols, meaning they can be added, modified, or removed by the programmer. In traditional networks, this capability can only be achieved by a manufacturer[55].

The main benefit of data plane programmability is therefore its increased flexibility[46]. This feature allows for the optimization of packet processing functions to meet specific use cases or requirements, tailoring the data plane to the network's specific requirements. Additionally, it facilitates the deployment of brand new network protocols.

The concept of data plane programming emerged in response to an issue found with then standard SDN solutions. SDN's ambitious goals naturally lead to initial approaches falling short. The first phase of SDN is defined by the Southbound API OpenFlow. The soundbound API is an interoperable interface that is used by the control plane to communicate routing decisions to devices, dictating how the control plane

Version	Date	Header Fields
OF 1.0	Dec 2009	12 fields
OF 1.1	Feb 2011	15 fields
OF 1.2	Dec 2011	36 fields
OF 1.3	Jun 2012	40 fields
OF 1.4	Oct 2013	41 fields
OF 1.5	Dec 2014	44 fields
OF 1.6	Sep 2016	Restricted to ONF members

Table 1: Fields recognized by the OpenFlow standard[56]

communicates with switches[53]. There are several other Southbound APIs available, with OpenFlow being the most popular in this first phase. OpenFlow is a protocol-dependent API that defines a specific set of headers for controllers to use in defining rules in the data plane. Openflow was standardized by ONF, an organization established with the objective of advancing SDN.

As OpenFlow evolved, researchers realized that its static nature was extremely limiting[56][55][53]. Protocol-dependent Southbound APIs assume a specific data plane functionality that cannot be altered. When working with OpenFlow, network policies must adhere to the existing matching fields and actions of OpenFlow, which undermines the controller's power over the protocols used on the network.

To work around its static nature, and as a necessity to support many different protocols, the number of actions and matching fields increased with each version of OpenFlow, which has a price in complexity. The first version of OpenFlow standardized 12 matching fields and 10 actions. As new versions were released, this number skyrocketed to 44 matching fields and 19 actions in version 1.5 [53][56], as can be seen on Table 1. This remedy can be thought of as a band-aid solution, as it increases complexity without solving the underlying problem of inflexibility.

Another important issue to consider is the reactive nature of definition of these headers. OpenFlow solely defines new headers when they become widely requested. Therefore, any new useful header fields must wait for standardization in a new version of OpenFlow before they can be integrated into networks. A proactive and open approach would be more appropriate, as this approach contradicts the SDN principle of enabling innovation[53].

OpenFlow attempted to fix this problem in version 1.2 through an OpenFlow Extensible Match, which allows for adding custom matching fields[51]. However, the fundamental problem remained and a brand new approach was preferable. Many different technologies aimed to further enhance data plane programmability emerged. This thesis will focus on the most popular, P4. As final observation, the protocol-dependent nature of OpenFlow also limits the protocols that can be used to standardized ones, meaning the defined matching fields of flow tables are limited to traditional network protocols.

3.3.1.2 P4

Programming Protocol-independent Packet Processors is a high-level programming language designed to specify switch packet processing pipelines. Similarly to other data plane programming efforts, it aims to

generalize the OpenFlow match-action framework and it has emerged as the most popular technology in this race, with strong support from both industry and academia[55].

The P4 project began with three main objectives[56]. The first was to enable device reconfigurability, so that the way a switch processes packets could be easily modified even after deployment. The second objective was to promote protocol independence by allowing switches to run any arbitrary protocol. Lastly, to achieve the last two goals while abstracting the underlying hardware, so that P4 code can run anywhere.

P4 achieves these goals by a new revolutionary process that, rather than keeping track of the vast number of standardized protocols used, defines an open, high-level and flexible approach for defining custom packet parsing and header matching mechanisms. This vision argues that instead of having to repeatedly extend the OpenFlow specification, hardware manufacturers should support flexible mechanisms for parsing packets and matching header fields so that controllers can leverage these capabilities through a common, open interface. [56]

P4 can actually be used with both programmable and fixed-function devices[45] and it works in two distinct phases [56][46].

During the initial phase, operators utilize the P4 language to define the parser and deparser, determine the order of match+action stages, and specify the header fields processed by each stage. For programmable devices, P4 prescribes the desired behavior in the pipeline, while in fixed-function devices, P4 merely describes the chip.

The second step occurs at runtime and consists of populating the configured pipeline with rules. This process is device-independent, with the control plane adding or removing entries in the match+action tables of the switch pipeline using the Southbound API. This phase determines the usage of defined protocols and the application of policies at a particular moment, such as adding entries to flow tables and associating actions with flows.

3.3.1.3 Hardware

In the creation of SDN infrastructure, any available hardware can be considered for use in the switches, with each having its own set of advantages and disadvantages. Following the approach of [43], hardware can be classified into three categories: general purpose computers, vendor specific hardware, and open networking hardware.

1. Open network hardware: Open networking hardware refers to switch specific hardware that is based on open standards and can be used with a variety of software. This provides a programmable and vendor-independent solution for building networks. The most appropriate analogy is to compare a switch to a Personal Computer (PC) built from commodity, off-the-shelf components. This allows for the assembly of a high-performance switch[45] using open source software. For example, the Open Compute Project takes advantage of existing commodity switching components and offers full architectural specification for switches online[57].

2. Vendor specific hardware: Continuing the analogy with the PC world, in that ecosystem pre-built computers are available for purchase from several vendors, such as Dell or HP. Similarly, bare-metal switch vendors like EdgeCore and Delta offer pre-built switches[45]. This hardware is specifically designed and manufactured by a particular vendor for networking purposes in SDN and uses proprietary software. Moreover, it is of interest to note that an initiative designated Indigo[58] has been established with the objective of facilitating the integration of SDN capabilities into conventional hardware.

3. General purpose computers: These are regular computers, not specifically designed for networking, that are used as a platform to implement the data plane by running a software switch that performs packet forwarding functions. Hardware virtualization is used to run virtual switches on standard operating systems running on regular hardware. The widespread availability of general-purpose hardware makes it inexpensive, and easy to reuse for different tasks. However, this hardware is not only not optimized for networking tasks, but also relies on computer network interface cards, which typically have a limited number of ports and slower speeds.

Both open network and vendor-specific hardware exhibit similar high performance, while using general hardware can result in decreased performance as it is not specialized for networking. General purpose hardware is usually the least expensive, followed by open network hardware, with vendor-specific devices being the most expensive. This price difference can be attributed to the scalability characteristics, ease of use, and performance of each type of solution. All and all, each approach has its benefits and drawbacks, so the best choice should be based on the specific needs and constraints of the target network.

Figure 21 presents a high-level schematic of a bare-metal switch. This abstraction better represents the open networking approach because Vendor-specific solutions are proprietary and therefore harder to generalize and the general hardware only has a Central processing unit (CPU), virtualizing to operate in a similar manner to the depicted.

The diagram depicts three main components. The Network Processing Unit (NPU), represented as a combination of Static Random-access memory (DRAM)-based memory and an application-specific integrated circuit based forwarding pipeline, is a chip optimized for parsing packet headers and making forwarding decisions. This representation ensures packets are buffered in memory while being processed in the pipeline. A general-purpose processor is connected to and controls the NPU. This chip runs the software of the switch and deals with the off-switch communications with the control plane. The last component is the device ports, which interconnects devices.

To achieve its goal of interoperability between different hardware, P4 programs must be portable across different devices without requiring modification. Achieving this is a problem, however, because different vendors implement different physical pipelines. To address this issue, P4 introduces the concept of an intermediate layer between the core P4 language and the targets[55]. This layer abstracts the details of the physical pipeline into a common logical pipeline that provides a programming model that easily describes how packets are processed. As a result, P4 programs can be developed for a logical architecture that can be deployed on multiple targets. Currently, there is no consensus on a standard

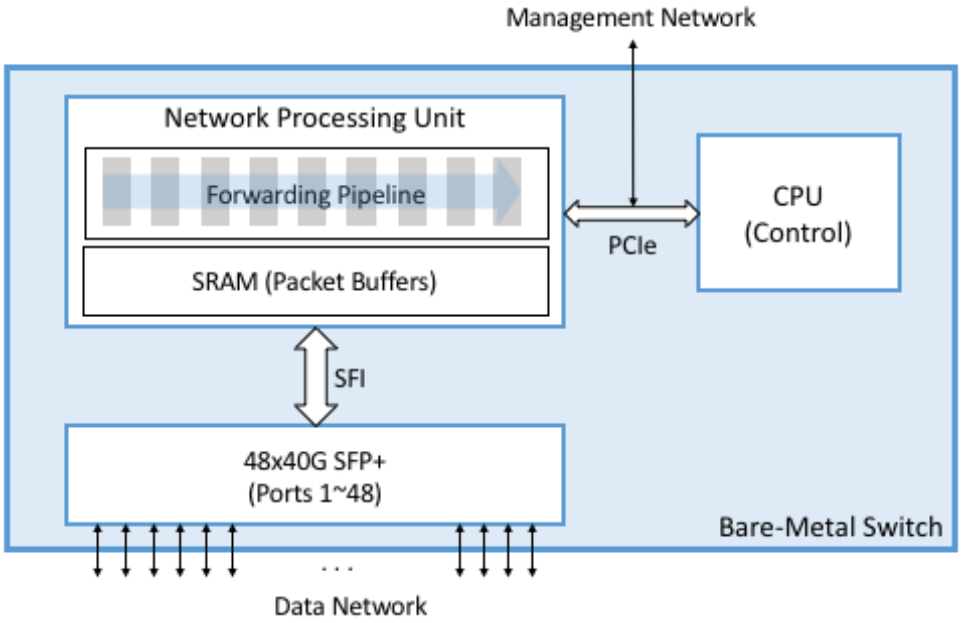


Figure 21: High-Level schematic of a bare-metal switch[45]

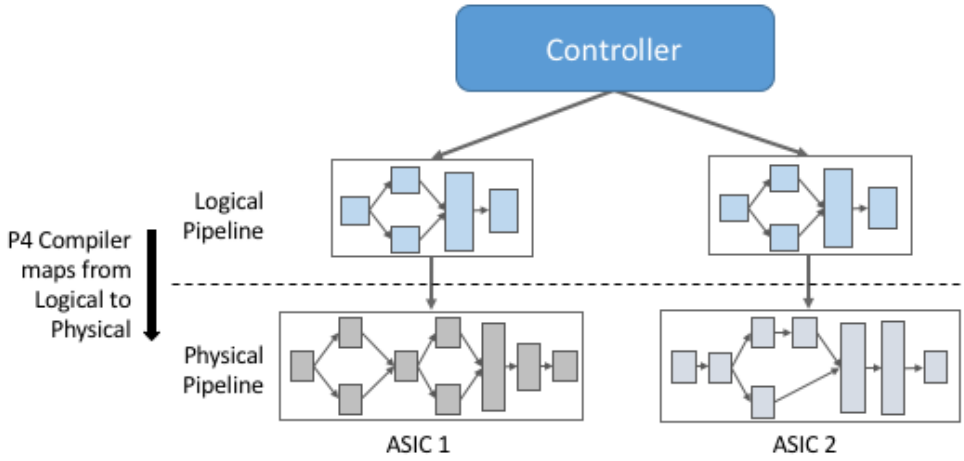


Figure 22: Definition of a logical pipeline to provide a pipeline-agnostic view the control plane[45]

logical pipeline for all devices. Nevertheless, this approach enables pipeline-agnostic controllers as can be seen on Figure 22.

The logical pipelines can be represented using different abstractions. In Figure22, the pipeline is described using the data match-action pipeline. This abstraction describes the packet processing functions of a network device as a pipeline of lookup tables. Each table is responsible for matching specific fields in the packet header and performing an action based on the result[46][51]. Thus, the data match-action abstraction describes the NPU as a sequence of lookup tables.

Other abstractions exist for describing the switch pipeline, but this is the most popular by far. OpenFlow is largely credited with popularizing the match-action abstraction[46]. However, the original proposal for OpenFlow took advantage of the fact that most routers and switches already contained flow tables used to implement additional network functions such as firewalls, network address translation, quality of service, and statistics collection, to define a common set of functions and provided an open protocol for

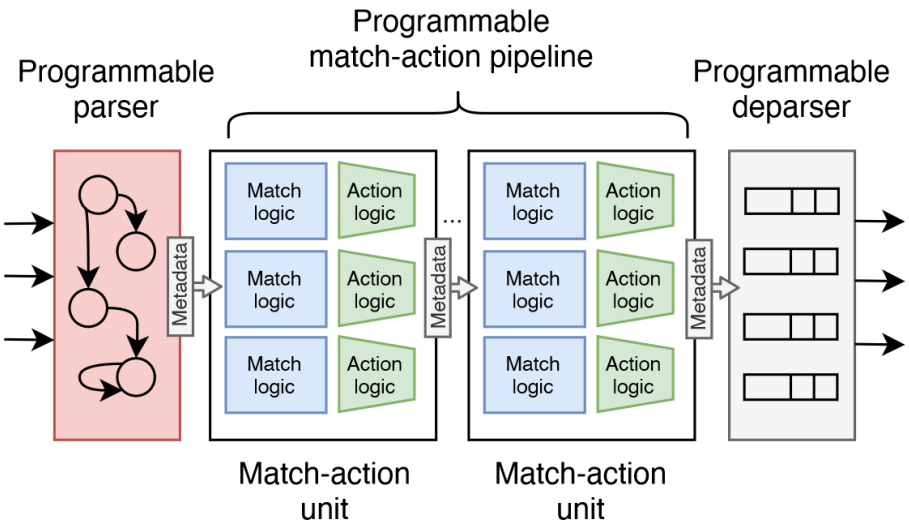


Figure 23: Protocol-Independent Switch Architecture[55]

programming these flow tables[52]. This implies that the architecture of the NPU and the Southbound API strongly influenced each other.

This abstraction depicts multiple tables because high speed switches require a multi-stage pipeline for packet processing. It is the optimal approach for packet processing because this process involves checking multiple header fields, and with multiple stages, different stages can perform different tasks simultaneously[45].

There are various P4 architectures, including Protocol Independent Switching Architecture (PISA), Portable Switch Architecture (PSA), V1Model, SimpleSumeArchitecture, and Tofino Native Architecture (TNA)[45][55]. These architectures abstract the target hardware into a forwarding model representation.

PISA presents a simple representation of the device, so all of these examples can be considered variations of it. The other architectural models follow the same principles with some arbitrary enhancements. The PISA model will be discussed as it is the most general. However, it should be noted that the V1Model is currently the most popularly used.

Protocol Independent Switching Architecture is a data plane programming model of P4. It was introduced in P4_14 and is currently deprecated. It is applied in a variety of ways to the creation of concrete architectures[55]. As an example, Figure 23 gives a high-level overview of this pipeline.

This architecture depicts three main components. The parser is the first component and handles header fields by specifying their size, order, and other relevant details. This ensures accurate extraction and utilization of the fields in the subsequent stage. The next stage is the match action pipeline, which consists of the multiple sequential tables that define this type of data plane pipeline abstraction. It maps a desired action to a given match by utilizing previously defined headers. The third is the deparser and it does the opposite of the first. It deparses the packet metadata into the packet before it is transmitted on the output link. Beyond the packet, relevant metadata about it is also traversing the pipeline. Examples of this may be packet relevant, like input port and arrival timestamp, or device relevant switch counters, queue depth.

In practice, when a packet arrives at a switch, after being parsed, the packet headers are compared to the entries in the flow table. If a match is found, the corresponding action is taken, and the packet is processed accordingly. If no match is found, the packet is typically forwarded to the controller for further processing or dropped.

3.3.1.4 Software

Each switch must be equipped with a Basic Input/Output System (BIOS), which is similar to the firmware found in microprocessors as it provisions and boots a bare-metal switch. The Open Network Install Environment has emerged as the standard for switch BIOSes[45]. On top of it, every switch must run a local Switch Operating System (OS), which is responsible for handling calls from the controller and taking appropriate actions on the switch's internal resources. The most widely used Switch OS is Open Network Linux[59], an open-source project based largely on the Debian distribution of Linux and extended to support hardware unique to switches[45]. Other open source Switch OSs include Stratum[60] and SONiC[61]. Vendor-provided software development kits for the on-board switching chips are also important. These kits are analogous to device drivers used in traditional operating systems.

3.3.2 Southbound API

The Southbound API is the second layer in SDN. As introduced in the previous section, the Southbound API is a logical interface that connects the controller with all network elements in the data plane[47]. It allows the control plane to push configurations to forwarding elements and the switches to push information to SDN applications, enabling the controller to have direct control and constant information on the state of the data plane elements. This protocol describes the capabilities of the switches, which defines the communication protocol and methods used to exchange data between the data plane and the controller. The interaction between the control plane and the data plane is therefore defined by this protocol[51].

A well-defined programming interface is crucial for the success of SDN, and its standardization offers several advantages for designing and deploying efficient SDN solutions[54] [51]. Specifically, it provides flexibility by allowing the control plane to communicate with the data plane through a variety of standardized channels. This enables network operators to choose from a range of controllers and switches, reducing vendor lock-in and expanding their options.

The responsibilities of this interface can be split between Control and Configuration[45] and usually, protocols only fulfill one of these categories. For a long time, OpenFlow was considered the state of the art for Southbound APIs and as testament to that is still the most widely used today[54]. However, the current state of the art for Southbound interfaces is a combination of P4Runtime, gNMI, and gNOI. P4Runtime and OpenFlow are control interfaces, while the gNXI combination of protocols is responsible for device configuration[45].

These different types of Southbound APIs also have the huge benefit of sheltering the controller and by extension the application running on top of it from the diversity of network devices that may exist in the network. [45]

3.3.2.1 OpenFlow

The previous section already touched on the origins of OpenFlow as a means to justify the industry's distancing from it. McKeown et al. proposed OpenFlow[52] as a means to enable experimentation on campus networks. OpenFlow leveraged the flow tables present in most Ethernet devices, which are used for additional configuration such as firewall setup and subnetting, to allow customization of campus network devices. [43]

OpenFlow has been the de-facto standard for the Southbound interface for a long time[41] and remains the most popular Southbound API. This is evidenced by the fact that most commercial switches support the OpenFlow API out-of-the-box[51].

3.3.2.2 P4Runtime

Nowadays, Openflow's development has been dropped by ONF in favor of P4, which means it has been replaced with the more open alternative of the P4Runtine. P4Runtime is a surging data plane API[55] that is closely coupled with the P4 programming language. P4 enables the data plane to be programmed according to network requirements. A static interface would eliminate the advantages of P4 since the controller would not be able to benefit from the custom programmable policies defined in the data plane.

To fully reap the benefits of P4, a dynamic communication interface between the controller and the switch is necessary[45]. To establish this interface, not only must the switch pipeline be known to the controller, but both the switch and the controller must be informed of the P4Runtime contract. P4Runtime is automatically generated along with the client and server-side stubs for the controller and devices, depending on the defined switch architecture[45].

3.3.2.3 gNXI

The term gNXI refers to the combination of gNMI and gNOI. The role of these two is somewhat blurred, but in general, gNMI handles the persistent state, whilst gNOI is responsible for clearing or setting the ephemeral state. In other words, gNMI is used for switch configuration, defining actions for retrieving or manipulating the state of a device via telemetry or configuration data, while gNOI is used for executing commands on a device, such as rebooting, pinging, and accessing other operational variables on the switch rather than directly changing its configuration[45][62].

In traditional networks, the configuration part of the Southbound api has been called the operations, administration, and maintenance interface which is usually accessed in command-line[45]. The equivalent of gNXI in traditional networks is Simple Network Management Protocol (SNMP). SDN has brought about the possibility for a different approach. SNMP was originally designed for closed devices and primarily focuses on reading rather than writing onto devices, making it more similar in capabilities to the configuration part of gNXI[45].

The gNMI has three methods. The well-known get and set, similar to their counterparts in SNMP, are used for reading and writing. The third is the subscribe method, which is used to request a stream of telemetry data[45].

OpenConfig[62] is responsible for the development of gNXI. OpenConfig is an industry-wide standardization effort to drive the industry toward a common set of configuration models. It is a collaborative effort within the networking industry to move toward a more dynamic and programmable approach to multi-vendor network configuration and management. gNXI is expected to become the standard management protocol due to strong industry support[45].

The underlying mechanism used by gNOI and by gNMI is exactly the same[45]. Actually, P4Runtime and gNXI all rely on the gRPC framework with protobuf[55]. The intricacies of protobufs and gRPC will not be detailed here, but these technologies are used for the benefit of not having to deal with the daunting list of formatting, reliability, backward compatibility, and security issues that other protocols, including OpenFlow, must deal with[45].

3.3.3 Control plane

Having outlined the structure of the data plane and its major technological developments, it is now time to move from the individual switch level to the controller network view. The control plane is widely considered the most important component of the SDN architecture[43] because it represents the centralized decision-making authority of the network, acting as the "network brain"[51].

The control plane consists of one or more controllers that oversee, manage, and manipulate the data plane through the Southbound API. All forwarding decisions are centralized in the control plane, which is dependent on a global view of the network to make accurate decisions. In this process, device-specific details are abstracted from the control plane[54].

Traditional operating systems provide abstractions of the underlying hardware that simplify the use and management of the underlying resources. As a result, it is possible today to build more complex applications faster, more securely, and more efficiently[51].

In various ways, the SDN layers bear deep resemblance to a regular computer system[47]. The application layer is comparable to the software applications that perform tasks using computing resources, the infrastructure layer is equal to the computer hardware and, most relevant to this case, the computer OS corresponds to the control plane.

For this reason, the controller is commonly referred to as the Network Operating System (NOS). This analogy introduces the same abstractions found in computer operating systems to networks[51], and is often used to better illustrate that the control plane is responsible for representing the network as a single system for network applications[41]. The terms NOS and controller can and will be used interchangeably from now on.

3.3.3.1 Controller responsibilities

A list of common applications implemented in the NOS can be collected from different SDN deployments and asserted as the base network service functions[51]. While such a list cannot be considered de facto obligatory due to the SDN root of modularity, it is still relevant to have a comprehensive list for familiarity's sake.

The controller is mainly responsible for topology management, enforcing the desired packet processing policies and performing other necessary functions to manage and regulate the network[45][54][46][51][63].

Topology management requires maintaining an up-to-date network view, which involves link discovery and host tracking. The main duty of the controller is to implement the desired packet processing policies. This task involves determining where packets should be forwarded by establishing traffic paths and defining packet flow rules. Furthermore, the controller selects the headers that require modification and installs all of these control commands on each forwarding device.

Lastly, the controller is responsible for performing other necessary functions to manage and regulate the network, such as monitoring device health by collecting status information and other important data from switches and performing maintenance procedures on those same switches.

3.3.3.2 Control centralization

Due to its prominent role, the performance of the controller largely influences overall network performance. The biggest criticism of SDN stems from the belief that centralizing the control plane will lead to scalability and performance issues. However, most practical results suggest that a single controller is capable of handling a substantial number of new flow requests, and therefore should be capable of managing most small to medium sized networks[41][63].

In medium to high network scenarios, however, the controller can quickly become overwhelmed. A single physical controller represents a performance bottleneck in the network with added security issues[63].

The architecture's logical centralized controller does not imply the existence of a single physical controller. For optimal performance, scalability, and availability, the network controller must be physically distributed[51]. A physically distributed but logically centralized control plane reduces lookup overhead and increases reliability while maintaining a centralized view of the network for applications to write in[41].

The physical distribution of controllers can be classified into flat or hierarchical[63]. In the flat architecture, all controllers maintain a global view of the network, whilst in the hierarchical architecture the network is divided into domains, where a single controller has jurisdiction over a single domain. To achieve a global view, a root controller is created to coordinate the actions of the other controllers and maintain the global view.

Another important advantage of distributed architecture is that a network controller can function similarly to other services by scaling based on workload to ensure availability and saved resources[45].

3.3.3.3 East/Westbound API

Some researchers propose the concept of an API specifically used for the coordination of these distributed scenarios[51][43][47][54][63]. This API is called the East/Westbound API and it provides the exchange of information between different instances of the controller in order help maintain a consistent network view. This API has no relevant standards. It is not typically considered a part of the overall SDN architecture, although in practice, something similar must always exist in distributed controller scenarios, as can be seen in Figure 24.

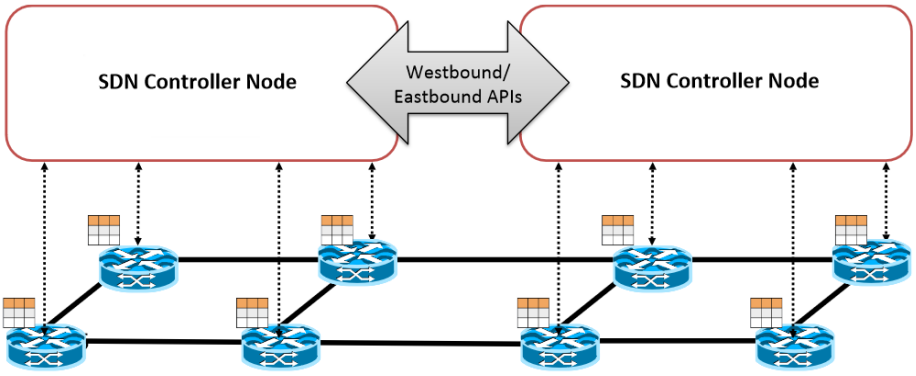


Figure 24: East/Westbound interface between distributed controllers[51]

A finer distinction can be made between the Eastbound and Westbound APIs[51]. The Eastbound API mostly to communication between SDN controllers and legacy routers, while the Westbound API refers to communication between different SDN controllers, whether they are different frameworks or the same in a distributed architecture.

3.3.3.4 Notable Controller Platforms

Most controllers are licensed as open source, with few having a proprietary license. They are programmed in a large variety of programming languages, with Java and Python being the most used, and some even contain more than one language for better performance. However, most don't receive frequent updates and maintenance, and lack proper documentation[63]. A lot of different controller platforms exist, and the most notable are Pythonic Network Operating System (POX), Floodlight, Open Day Light (ODL), ONOS, and Ryu.

All controllers can run on any commodity hardware[51]. They are designed to be highly modular, with the controller being customized and configured to the given deployment, only having the subset of required modules[45]. This means that the behavior of most controllers is generally the same, with the only relevant differences being the deployment architecture. Another important point where controller architecture varies is the support for different noth, east, west and Southbound APIs, as not all controllers are compatible with all interfaces.

3.3.4 Northbound API

The Northbound API is a logical interface that connects the control and application plane. It is offered to the application developers by the NOS[51] and provides programming abstractions for networks, acting as a bridge between the control and application plane[54].

There is no fixed or widely accepted standard for Northbound APIs[51][54]. The reason for the lack of standardization of this interface is the huge variation of applications and user requirements, that makes the existence of a single interface for all of them very difficult.

There is therefore a large amount of different APIs that can be used depending on the chosen NOS. The purpose of this large number of different APIs is to sufficiently provide control over all underlying functionalities to the application plane[45].

As a Northbound interface implementation, programmers and most controllers typically use the representational state transfer API[54]. The gNXI interfaces are also popular, with them corresponding exactly to their counterparts in the Southbound API[45].

3.3.5 Management plane

The application layer is the upper-most layer, sitting above the controller. It consists of network applications designed to meet specific user requirements and, through the Northbound API, this layer has easy access to the capabilities of all the lower layers such as the global network view and constant status information, enabling software applications to instruct the network controller on how to implement network control and operation logic[43][49].

SDN applications can accomplish a variety of goals, from adaptive routing and boundless roaming to enhancing network security and many more[43]. In essence, this layer can request custom control policies for routing, firewalls, load balancers, monitoring, and so on. The controller interprets these policies and determines the actions to be taken by the network devices. The switches then translate the network commands into concrete actions in the forwarding tables[51].

The existence of this layer is sometimes omitted, depending on the interpretation. It can be considered an integral part of the control plane because the two are closely related. The application can be viewed as running on top of the controller as well as on top of the controller[45].

4

SDVN

The SDN paradigm has been identified by some researchers and other experts as a powerful and promising tool for addressing some of the challenges posed by VANETs[64]. As previously discussed, SDN is a methodology rather than a specific technology, which allows it to be applied to VANETs. This line of thought created a new networking paradigm that is primarily known as SDVN, although the term Software Defined Vehicular ad hoc Network (SD-VANET) is also occasionally used.

The concept of SDVN was first introduced by Ku et al. in [65], and since then there have been many different proposals of how to implement the SDN principles in VANETs. SDVN has recently received significant support from researchers, which has resulted in significant technical and architectural advances in SDN-enabled vehicular networks[66].

Similarly to traditional wired networks, non-SDN systems function reasonably well. Therefore, the implementation of SDN should not be seen as a solution to make VANETs work, but rather as a means to enhance them.

4.1 Benefits and Challenges

The implementation of the SDN paradigm in VANETs is driven by the desire to bring some of the improvements of SDN to VANETs to tackle the challenging requirements of VANETs. This paradigm has the potential to introduce flexibility, programmability, and centralized control to vehicular networks[66]. This section reflects upon the benefits of SDN and the characteristics of VANETs to extrapolate the benefits and possible challenges that SDVN can offer today.

4.1.1 Mandated Interoperability

As previously stated in Section 2.6, VANETs require high road acceptance for their useful applications to be effective. This implies that all implemented devices must be able to communicate with each other, and to this end the EU has tried to standardize devices to promote interoperability between different vehicle manufacturers.

In SDN networks, operators have complete control over a static network, making it easy to conceal the inner workings of devices from the rest of the Internet and implement them as desired. However, in SDVN the same is not possible because a vehicle ITS subsystem must be able to communicate with every single roadside ITS station in order to access infrastructure services deployed in the central ITS subsystem. SDVN devices must be compliant with existing regulation and communicate with standardized protocols in order to be interoperable with other devices.

Such an emphasis on interoperability undermines the powerful control that SDN brings. Network operators and developers can't take full advantage of the freedom this technology provides to implement new algorithms and test them in real-world scenarios. Even the effectiveness of the additional control provided by P4 is reduced by the inability to change network protocols.

This is not to say that the increased control that SDN brings is completely useless, as it can still provide some advantages in dealing with certain scenarios, but this VANET characteristic largely limits what can be accomplished with SDN.

It is also relevant to note one of the major goals of SDN, which is to move network control from manufacturers to operators, with the ultimate goal of giving it to network users. In vehicular networks, the identity of the network operator becomes an implementation detail, but it will most likely be the same government of private entities deploying the infrastructure. SDN continues to provide these operators with greater network control, but it cannot be given to users for fear of being misused for nefarious purposes.

4.1.2 Network innovation

Another important conclusion to be drawn from the above is that the technologies that will form future VANETs can be expected to be even more static than in traditional wired networks. On top of that, current hardware defined devices exacerbate this problem, as any fundamental change to VANETs would require the hardware of thousands of roadside systems and millions of vehicle systems to be manually replaced. This presents a logistical nightmare, making it highly unlikely that this scenario would happen except in the rarest of cases, and then only if mandated by the EU. This reality is evidenced by the slow and arduous standardization process that VANETs have undergone over the last 20 years, which has sought to find a single solution for VANETs that would act as a definitive and permanent answer to all of its problems.

SDVN allows new functionalities to be implemented in the same hardware through software updates, facilitating changes to be made throughout the network. This could considerably accelerate the adoption of this technology and facilitate the implementation of any future fix or optimization.

4.1.3 Network management and performance improvements

Even more than in static networks, SDN can provide VANETs with more flexible and intelligent packet flows, channel allocation, and connectivity. Traditional VANET routing algorithms are limited to the individual perception of each node as they are implemented in each individual device. SDN is built on a global view to make better decisions based on the combined information from multiple sources, which results in better network performance since informed decisions lead to more optimal outcomes. The automated network

capabilities brought about by SDN also have the potential to address VANET's adversities. Listed below are some of the characteristics of a VANET and how they can be enhanced through the implementation of SDN.

High mobility In an environment where topology changes occur frequently, it is critical to be capable of adapting to sudden and unexpected events. SDN provides flexibility, which allows for more dynamic network configuration, enhancing the network's responsiveness to emergencies and changing requirements, while also enabling it to better adapt to changing conditions and needs[65].

Predictable mobility SDVN can leverage the centralized view of the network, to better exploit the predictable mobility when compared to traditional approaches. The controller can use a vehicle's information to predict future topology changes and set traffic rules accordingly, improving connectivity and overall network performance.

Congestion and scalability Network congestion and the variable vehicle density are issues that can be reduced by leveraging the global view of the network to better optimize the available mediums. Centralizing network control increases cooperation, allowing the optimal path to be calculated, which reduces delays and overhead[64]. Also, situations of high vehicle density can be more easily predicted and the transmission power levels on vehicles can be jointly adjusted based on future vehicle network density, greatly reducing interference[64]. Overall, automated tools promise to make runtime changes to networks based on the unique characteristics of VANETs, helping to achieve a better performing network.

4.1.4 Issues with maintaining a global network view

The advantages of centralized control are plentiful, but the establishment of this centralized view in VANET is not guaranteed. VANETs are based on wireless connectivity, which causes the southbound API to suffer from instability and even complete unavailability[67]. Combined with volatile topologies, this results in increased communication delays in the southbound interface, which can lead to inaccuracies in the controller's view of the network and in the rules established in the switches, casting doubt on the controller's ability to maintain an accurate and up-to-date view of the global topology[68].

Another major concern introduced with SDVN is the management overhead introduced by the southbound API required for the controller to manipulate the devices, as switches must be frequently updated to accurately reflect the network's state. This could result in additional traffic and increased network congestion compared to the distributed methods used in traditional networks.

At the same time, it should be noted that it may not be necessary for all instances of the controller to maintain a global view of the entire network. Instead, individual instances of the controller may only need to keep track of a small region of the network, as most messages in VANETs are only relevant in their immediate surroundings[69].

In summary, the dynamic state of the network combined with poor controller connectivity presents a major concern in SDVN. The process of maintaining an updated view of the global network topology

becomes costly and time-consuming due to potential inaccuracies in the updated information[68]. To address this issue, it is important to ensure robust protection against loss of connectivity and to guarantee the availability of the control plane, which sometimes requires the integration of other technologies such as fog computing with SDN[68].

4.1.5 Cheaper networks

When considering the deployment of VANET technology, the cost of infrastructure is a major concern, with up to 86% of the total expected cost being attributed to hardware[16]. Just as in traditional networks, SDN can reduce costs by reducing complexity and increasing modularity while promoting interoperability.

4.1.6 Network security

The introduction of the SDN paradigm does not introduce any new significant security or privacy challenge to VANETs. Some researchers[68] mention the centralized controller as a new vulnerability that can be exploited, but in reality current VANET deployment already relies on services provided by the central ITS subsystem.

4.2 SDVN architecture

The SDVN architecture can be represented based on either of the architectures of the two technologies that define it. Of the two, the most suitable is the architecture of SDN. The VANET architecture is best represented by the ITS host architecture, which is built on the OSI model. The SDN paradigm aims to move the internet away from closed and complex standardized protocols and into open and simple network components. As such, the SDVN architecture inherits the format as the general SDN architecture, retaining the three layers present in it as can be seen in Figura 25.

4.2.1 Data plane

In SDVN, as in SDN, the data plane consists of all forwarding hardware but these components can be divided into two different categories based on whether they are wired or wireless. Mobile hardware refers to the vehicle ITS station, while stationary hardware refers to the routers and switches found in the remaining ITS subsystem. Network intelligence is extracted from all ITS stations, but due to the special characteristics of the mobile nodes all SDN enabled devices cannot be treated the same.

4.2.2 Control plane

The control plane remains almost unchanged as the centralized logical intelligence of the network, managing device behavior to achieve desired policies. It remains responsible for communicating the high-level applications defined in the application plane to the devices in the data plane, and for transmitting device

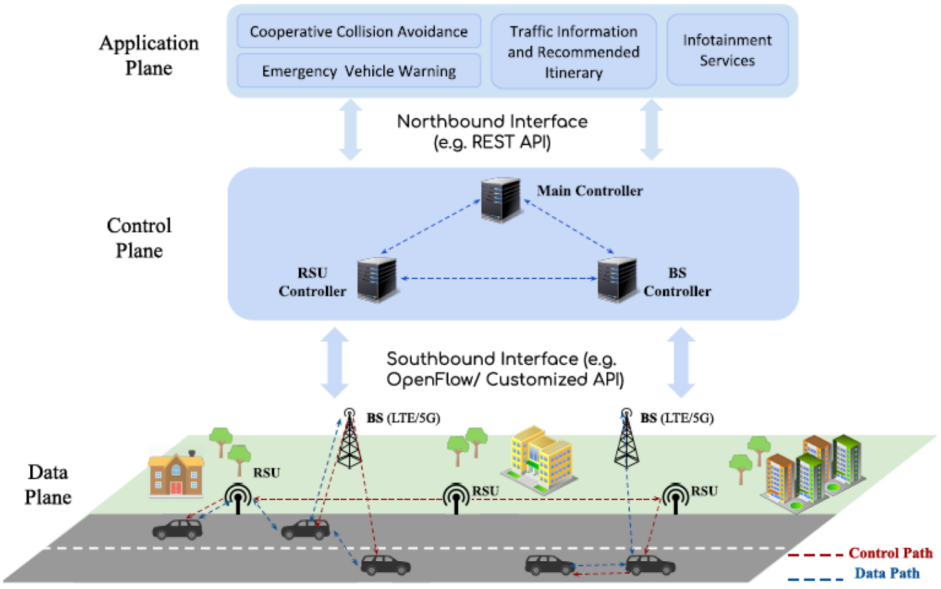


Figure 25: Conceptual view of SDVN architecture[70]

information from the data plane to the application plane. The main difference introduced by SDVN is a consequence of the need to manage devices in the ad hoc space. Due to the two distinct types of components in the data plane, unique and customized networking applications are required to address the unique nature of each scenario. Managing the wired portion of the network can be considered a simple task because the process is the same as SDN, whilst the wireless domain presents unique challenges and requires distinct solutions to manage wireless resources as effectively as wired resources[67].

In SDVN controller distribution becomes a central concern. Vehicular networks have immense coverage given that they include all roads, and due to this vast distribution, a single controller cannot handle the traffic of the entire network. Therefore, it is out of the question for scalability and delay reasons[71]. Instead, the control plane must consist of multiple physical controllers efficiently distributed across the network, coordinating their decision-making efforts to handle the high volume of traffic[68]. Another important motivation for distributing the control plane is to bring the controller closer to the vehicles to reduce communication delays between the data and control planes[72].

It should be noted that a truly distributed controller, where all instances have global information of the network, seems impossible and even counterproductive. To perform effectively, controllers usually only need information about a specific part of the network, which is in part because some messages are only relevant in the context of that particular section of the network[67] and the ones that aren't should be directed to the nearest roadside ITS station. The size and the transition between regions is dependent on implementation.

That being said, there exist many different controller placement distributions proposed in the literature. In practice, the controller can be located in the central, roadside, or vehicle subsystems. Different architectures place the controllers in a combination of these stations, giving them distinct responsibilities and utilizing different methods to distribute intelligence between them.

Most approaches to controller distribution in the literature[66][67][71] converge on a two-tier hierarchical controller. The top-level orchestrator controller, also known as the global or primary controller, is located in a server integrated into the central ITS subsystem. The lower-level controllers, referred to as local or secondary controller instances, are placed closer to vehicles in the roadside ITS subsystem in an effort to improve connectivity and reduce delays.

4.2.3 Application plane

The application plane remains unchanged. It consists of network applications designed to fulfill certain networking tasks using the capabilities given by the control plane. Some of these include monitoring, QoS, analytics, recovery, security, routing, load balancing, and management[66]. As stated in Section 2.6, VANET applications can be divided into two main categories, each with different QoS requirements. The most significant of these, the safety applications, are mostly delay sensitive, while the non-safety applications are more bandwidth intensive[64].

4.2.4 Communication interfaces

All communication APIs have no relevant changes. The only noteworthy detail is that by having access to both ITS-G5 and C-V2X for communication and control[71], a controller can has two main communication mediums through which, using the southbound API, it can communicate with SDN devices.

5

Related papers

Emulation is a convenient and cost-effective method for testing VANET solutions. However, due to the complex and unpredictable nature of real-life environments, relying solely on these tools is insufficient[67]. In order to gain a realistic perspective on VANET, it is necessary to implement it in real-world scenarios so the validity of the expected benefits and drawbacks can be tested. The same is true for SDVN. SDVN is notable for the lack of experimental efforts conducted on actual hardware. In order to enable testing on real scenarios, open-source SDVN tools and frameworks that are compatible with a wide range of hardware are indispensable[67]. This section reviews several related papers that aim to implement VANETs and SDVN in physical hardware using open sourced components, outlining the key details of their approach and architecture.

Raviglione et al.[73] present a demo paper that assembles an open-source platform based on PC Engines' boards and Unex's Wireless Network Interface Controllers (WNICs). The purpose of this paper is to create a testbed for testing applications that communicate in the vehicular environment. Even though this paper does not attempt to implement SDN principles, it is relevant because it provides all the necessary hardware and software components to implement a vehicular testbed. The authors assembled two boards consisting of the embedded PC Engines APU1D board with an AMD G-series dual-core T40E x86 CPU with 64-bit support and 2 GB Dynamic Random-access memory (DRAM). Communication via 802.11p was achieved using the Unex DHXA-222 Mini PCI Express (mPCIe) card as the WNIC. This chip is based on the Atheros AR9462 chipset which is supported by the ath9k Linux driver. To enhance storage and memory performance, a Serial Advanced Technology Attachment (SATA) III Transcend MSA370 MCL NAND Flash Solid State Drive (SSD) was installed. The OS used was OpenWrt release 18.06.1 with Linux kernel 4.14.63. The authors made modifications to the ath9k Linux driver in order to utilize the channels of the 5.8/5.9 GHz frequency band in accordance with ITS-G5 standards. These modifications were then integrated into the OpenC2X project, which was subsequently ported to OpenWrt. The paper also reviews some modifications and implementations made in higher layers, but these are not relevant to the problem addressed in this thesis.

Sedar et al.[74] developed and validated an experimental, standards-compliant OBU. Their experimental platform is based on an open-source software implementation of the ETSI ITS protocol stack. Its

purpose is to facilitate interoperability in communication between various devices and cloud-based services. The OBU was built using general purpose hardware in the form of a generic laptop, running Ubuntu 18.04. To grant cellular connectivity, it was connected to an LTE AirPrime EM7565 modem from Sierra Wireless which was in part connected to the 4G cellular network of Vodafone-Spain. The experimental OBU is also connected to external hardware devices, these being a 4G/5G cellular modem, a GPS/global navigation satellite system receiver and a connector to receive information from in-vehicle sensors. The article presents an overview of the current state of open-source software implementations of the ETSI ITS protocol stack. It mentions OpenC2X and Vanetza as the two primary implementations. Although real vehicles were not used in testing, the authors state that their experimental platform can be easily integrated into any vehicle.

Secinti et al.[75] proposed an architectural model that implements SDN and virtualization principles in order to enable VANET with Wi-Fi access capability. In this architecture, both the OBU and the RSU are implemented using the same type of hardware and software. Both have been implemented using a Raspberry Pi, with the wireless connectivity being provided by the Realtek 5370 Wi-Fi SoC. These switches are implemented using OpenvSwitch v2.3.90 running on OpenWRT. OpenvSwitch is a software switch implementation that is open source and natively supported by the Linux kernel[76]. Finally, the controller is implemented using OpenDaylight and the southbound API used is Openflow.

Rito et al.[77] present the deployment and experimentation architecture of the Aveiro Tech City Living Lab in Portugal. The implementation involves a diverse range of devices connected through fiber, radio ITS-G5, and cellular links, utilizing various technologies such as SDN, named data networking, and fog computing. Vehicle and roadside stations are implemented using the PC Engines APU2 board equipped with an SSD, an IEEE 802.11a/b/g/n mPCIe wireless card and an LTE CAT-1 mPCIe or 5G m.2 module. Additionally, an external USB dual-band wireless adapter was also installed. The operating system used is not specified, but it is a linux distribution because in order to enable the European version of 802.11p in it they used the Linux ath9k driver. This paper implements a myriad of different technologies in vehicular infrastructure, one of them being SDN. It is notable that it does not implement SDN in vehicle ITS stations, but only in the backbone of the network. The purpose of this SDN implementation was to use the increased control in the backbone of the vehicular infrastructure and the vehicle information to predict and execute handovers in advance. The authors developed a custom protocol dubbed OBUInfo to provide CAM information to the controller. This provides the controller with location, heading, speed, and vehicle type information, which is useful in predicting future handovers ahead of time.

Sadio et al.[78] propose a complete SDVN prototype design. The hardware used for the OBU was a Raspberry Pi 3 with Cortex-A53 × 64 1.2 GHz and DRAM 1 GB. This board has access to WiFi 2.4 GHz 802.11 b/g/n and via a Huawei E8372 LTE USB modem to LTE. The operating system used was Raspbian Stretch Lite. The authors utilized the Python Twink library to transform the devices into OpenFlow switches. This implementation fails to use the standard for communication in the VANET environment, 802.11p.
In closing, we present the following table, which provides a concise overview of the pertinent literature.

Paper Reference	Objective	Hardware Used	Software Used	Communication Protocol	Implementation Type	Key Contributions/Notes
Raviglione et al. 2019[73]	Create a vehicular testbed	PC Engines APU1D, Unex DHXA-222 WNIC	OpenWRT 18.06.1, modified ath9k driver	802.11p	VANET	Provided detailed hardware and software to assemble a testbed.
Sedar et al. 2021[74]	Standards-compliant OBU	Laptop, Sierra Wireless LTE modem, GPS receiver	Ubuntu 18.04, open-source ETSI ITS stack	LTE, 4G/5G	VANET	Used general-purpose hardware, open-source protocol stack.
Secinti et al. 2017[75]	SDN-enabled VANET architecture	Raspberry Pi, Realtek 5370 Wi-Fi SoC	OpenWRT, Open-vSwitch v2.3.90, OpenDaylight	Wi-Fi	SDVN	Implemented both OBU and RSU using Raspberry Pi with SDN.
Rito et al. 2023[77]	Deployment of Aveiro Tech City Living Lab	PC Engines APU2, mPCIe wireless and LTE cards	Linux (unspecified), ath9k driver	802.11p, fiber, LTE, 5G	Hybrid (SDN in backbone only)	Custom protocol (OBUInfo) for handover prediction in SDN backbone.
Sadio et al. 2020[78]	Complete SDVN prototype	Raspberry Pi 3, Huawei LTE USB modem	Raspbian Stretch Lite, Python Twink library	Wi-Fi, LTE	SDVN	Used Raspberry Pi for OBU, non-standard protocol for VANET communication.

Table 2: Summary of related literature

6

Conceptual Solution

The comprehensive examination of the two fundamental pillars that constitute SDVN and SDVN itself has provided insights into the potential usefulness of this technology in real-world scenarios, thereby enabling the identification of an ideal practical deployment for this technology. As such, in this section, this ideal view created from the accumulated knowledge presented in this document will be described. Before proceeding to the specifics, it is important to note that this perspective describes some hypothetical optimal objectives for an implementation. Consequently, any practical implementations developed in this thesis will be constrained by the availability and costs of the requisite hardware and the availability and compatibility of the related software, in addition to timing and implementation constraints.

6.1 The impact of interoperability

In light of the requirement for interoperability, as delineated in section 4.1.1, an SDN-compatible OBU deployment is only viable in real-world scenarios if it is capable of communicating with ETSI conformant devices. This limitation for implementation is only applicable in real-world deployments, as the devices must comply with, or at the very least be compatible and able to collaborate with, the surrounding protocols in order to be effective. No existing software stack for SDN is compatible with the ITS-G5 protocol stack. Consequently, the most straightforward approach to create a compatible environment is to utilize P4's capacity to reprogram the packet logic of a network. The vast majority of research papers on SDVN only implement the first generation of SDN, with the few that delve into P4 and its advantages only doing so on a theoretical level, never providing an implementation platform. To the best of our knowledge, and in accordance with the findings of Sarpong et al.[69], no work has been published using P4 as the data plane technology in a SDVN implementation. It is important to acknowledge that the aforementioned limitation does diminish the freedom introduced by P4. However, it does not entirely negate its advantages, because the ability to modify network protocols enables protocol experimentation, thereby allowing existing protocols to be tested, validated, and optimized with greater ease. These observations lead us to conclude that the optimal SDVN scenario would entail the use of a P4 device that emulates the protocol fields of the ETSI-defined protocol stack, modified only to accommodate changes to the controller. The aforementioned conclusion, derived from the collected theoretical evidence, suggests that OpenFlow-based solutions are

not a viable option. Nevertheless, in order to corroborate this hypothesis, the requisite practical test will be conducted.

6.2 Controller placement

The controller brings us to the second issue of controller distribution. Section 4.2.2 displays the three different possible locations where the controller can be implemented, but most SDN developments only implement controller instances in the wired infrastructure. Due to the issues touched upon in subsection 4.1.1 and 4.1.4 devices must be able to function in tandem with traditional devices and without connection to infrastructure. Ku et al.[65], although the first, had already raised these issues deserved attention by incorporating recovery mechanisms in their experimental scenario. They used a backup traditional algorithm when the device lost connection with the controller, demonstrating that SDVN must have failure recovery mechanisms to ensure the network works in all scenarios. The present research will not be able to examine the particulars of the controller, such as identifying the optimal controller distribution scheme in software-defined vehicular environments. This is because it is essential to examine all subsystems within the ITS domain in order to achieve meaningful results, and this thesis will focus on the ITS vehicle subsystem. However, for the aforementioned reasons, it should be considered fundamental for a practical SDVN deployment to contain a local controller able to operate each individual device and take over with something similar to traditional software if the connection to the infrastructure is lost.

7 Device design

This section commences with an analysis of the available options, the inherent limitations that were identified, and other significant considerations related to the implementation of this project. Subsequently, the specifications of the system implemented in this thesis will be described, along with the rationale behind the decisions made during the implementation phase. This process cannot be conducted without first acknowledging the constraints imposed by the unique circumstances of this particular case. The first of these constraints is financial, as the absence of external funding significantly reduces the range of available options. Furthermore, the options for suitable hardware are constrained by their compatibility with the essential free and open-source software that will be utilized in the project. The software in question is of critical importance in meeting the necessary criteria for the system to be considered as vehicular. In the majority of cases of this nature, the issue of performance is also a significant concern. The capabilities of the components of the device are of significant importance when considering the performance of our solution, as they directly impact the device's performance. However, our main goal is to demonstrate the feasibility of such a device, and as such it is necessary to set aside concerns regarding performance in order to reduce the cost. In the event that any of the aforementioned restrictions prove to be applicable, they will be duly noted in the appropriate section.

7.1 Box design

7.1.1 Data plane

7.1.1.1 Hardware components

The selection of appropriate hardware was guided by the different categories of SDN devices outlined in section4.2.2. Accordingly, our analysis began with an assessment of the viability of each of the three proposed approaches for the intended implementation of the solution resulting from this project. In light of these criteria and of the existing financial limitations, general-purpose hardware emerged as the optimal choice for this problem, chiefly due to its comparatively lower cost. The impact of price constraints is so substantial that it effectively invalidates the remaining alternatives, rendering them inconsequential and thus they will be excluded from this analysis. As a result, the proposed implementation will be based

on the use of general-purpose hardware, with the switching functions being emulated through the use of software. The process of assembling any device requires attention to the individual components that will be incorporated and that comprise the complete system. The most notable components include the motherboard, the CPU, the Random-access memory (RAM), the Network Interface Controller (NIC), the power supply, and the case. It should be noted that a multitude of additional components could be considered, given that, in real world development, an OBU is often equipped with a plethora of other specialized dongles. The selection of the motherboard will inevitably exert a determining influence on the subsequent choice of other components, and as such it was where we began. Following a comprehensive analysis, two alternative solutions with significant potential have been identified.

- PC Engines[79]: PC Engines boards are designed for use in academic institutions and for networking purposes. The company provides not motherboards, but complete affordable solutions, which include CPU, RAM, NIC, power supply, and case. Additionally, these solutions rely on active cooling, which facilitates their assembly and reduces operational costs. Another notable feature is the number of ethernet ports and the quantity of Peripheral Component Interconnect (PCI) slots, which are both sufficient for the intended use. Lastly, other projects at this university have utilized boards from this company, which contributes to the overall appeal of this solution given the high level of familiarity with the company.

- Raspberry Pi[80]: Raspberry Pi is a very well known series of small single-board computers developed by the Raspberry Pi Foundation in association with Broadcom. As a highly recognizable brand, it offers comprehensive assistance and is a well-established brand with a substantial code and user base. This solution's main strength lies in its modularity, as the hardware can be assembled in accordance with specific requirements, thereby allowing for a high degree of flexibility and adaptability. These components are mostly inexpensive, which presents a valuable upside when considering deployment.

From these two options, and given the established familiarity, the PC Engines option stands as the most appropriate choice. As stated above, PC Engines' boards are sold as integrated solutions, comprising a pre-selected range of components. The decision regarding the specific model to be adopted from this company will consequently have a significant impact on a multitude of other components. The exact model that will be deployed will be discussed in detail at a later stage. In regard to the potential utilization of dongles in a real-world development setting, it is evident that the device would be equipped with a multitude of sensors intended to gather diverse types of data and enhance the device's capabilities. This work will primarily focus on a proof of concept for the integration of the communication component of VANETs with SDN principles. Consequently, further discussion of the sensors used in vehicles will be omitted. The only noteworthy dongle that we will be employing are bidirectional wireless antennas capable of communication at 6GHz.

7.1.1.2 Operating System

Having completed the analysis of the hardware, the appropriate next step is to proceed with the examination of the software. The foundation of any software architecture stack is the BIOS of the system. It is thus fitting to initiate this analysis at this level. In a context where specific switching hardware had been selected, the BIOS of the system would assume a greater significance. In this particular scenario, it would be reasonable to consider the potential use of Open Network Install Environment[81], an open and free standard specifically designed for these use cases. However, given that we will be employing general-purpose hardware, there is no compatibility with such software, and we are therefore effectively compelled to utilize the BIOS that is natively integrated into the motherboard. The OS is installed on top of the BIOS, and thus, this discussion will proceed with an examination of the extant options for the OS. Once more, in the event that we were to utilize switching hardware, Open Network Linux (ONL), Stratum or Software for Open Networking in the Cloud (SONiC) would be optimal choices. Stratum, ONL and SONiC are open-source OSs designed for use with switch hardware. Given that we intend to utilize generic hardware, any compatible Linux-based OS is an adequate option.

7.1.1.3 Wireless communication

Our research is focused on achieving wireless communication in vehicular environments. To this end, it must be ensured that the machine is able to communicate using one or both of the 802.11p and C-V2X standards. It would be very beneficial for the device to be capable of communicating with LTE and 5G networks; unfortunately, given the time constraints, the priority for this device is the ability to communicate in the 802.11p standard. As one descends through the hierarchical structure of the ETSI standards for vehicular communication, it becomes increasingly challenging for SDN technologies to adapt and conform to existing standards. All layers above the link layer can be readily modified or altered using SDN technologies in a manner that allows them to be considered vehicular. The 802.11p standard, which represents the lowest level of the stack, is not as straightforward as other layers. Our research revealed three distinct methods for enabling it:

- Purpose made hardware: The first option is to acquire specialized hardware that is capable of communicating at the desired frequency while complying with the necessary standards. This solution is the most straightforward, but also the most expensive. Consequently, for the same reasons as for the rest of the hardware, this option is not a viable solution and will not be considered further.

- Driver patch: Some researchers have solved this problem[82] by employing the use of a capable chipset and modifying the diver code to enable communication in the desired manner. This approach has been previously implemented in other research projects at this institution, and as such the first box received for use in this project is already equipped with this patch. This solution is especially appealing because it is the most affordable. However, it is also the most complex solution, as the implementation of such a feat requires a comprehensive understanding of networking, the 802.11p standard, drivers, and kernel modules. Even with this knowledge, a chip that

is capable of communicating at the desired frequencies that is open source is required. The intricate nature of this component renders the development of a solution from scratch an arduous and time-consuming process, which is impractical within the context of this project. Fortunately, as previously stated, there is existing research that can be utilized to achieve this goal. In a similar context, this university has previously employed a patch for an atheros chip. This family of chips operates with open-source drivers, of which the driver in question is named "ath9k." One factor that merits attention is the age of this driver. The ath9k driver is designed for 802.11n, which was released in 2009, and raises concerns about its long-term reliability. Furthermore, there appears to be no initiative within the community to implement this solution on modern drivers, nor to make the existing implementations official [83].

- Software Defined Radio (SDR): SDR is a relatively recent technology which bears resemblance to SDN. It enables the user to modify and configure radio signals. In comparison to the preceding option, this appears to be a more long-term solution, albeit at a higher price point. The subject of SDR research is beyond the scope of this thesis, and therefore it will not be a topic of discussion.

Of the three options presented, the most promising is the utilization of an Atheros chip in conjunction with the existing patch. Should this approach prove inadequate, it will be necessary to consider one of the alternative strategies.

7.1.1.4 Virtualized switch program

The selected hardware type mandates the use of software to emulate the typical functionality of a switch. The implementation of this methodology is contingent upon the availability of virtualized switch software. Our research has identified two potential options: OvS[84] and BMv2[85]. OvS is an open-source implementation of a distributed virtual multilayer switch, designed with the intention of providing a switching stack for hardware virtualization environments. In contrast, BMv2 is a pure P4 software switch, which is primarily utilized as a testing tool for the P4 language. Both options are available as free software, but it should be noted that OvS is not merely an OpenFlow switch. Consequently, it is significantly more utilized than BMv2 and offers considerably broader support. In this study, we will evaluate both solutions to ascertain their capabilities and viability within the context of our conceptual framework.

7.1.2 Southbound API

The decision regarding the southbound API is unambiguous and will be made in accordance with the established conventions within this field. Accordingly, tests will be conducted on both OpenFlow and P4Runtime. In regard to the question of the gNXI stack, its role is not indispensable for demonstrating the research question, and thus it will be set aside for the purposes of this study. While the value of integrating gNXI in communication is of great importance in real deployments, this study aims to prioritize the establishment of communication, and thus the integration of such systems will be considered secondary.

7.1.3 Control plane

Generally, each controller framework comes with its own properties, features and characteristics. The choice of what controller is best for a network is dependent on performance and compatibility. As previously stated, the issue of performance is not a significant concern in this particular context. In regard to compatibility, the term denotes the ability of the controller framework to operate in conjunction with a multitude of different technologies. This is not a cause for concern since the lower layers utilize open and industry-standard protocols and software, which are universally adopted. It is currently not feasible to integrate SDN software with Vanetza, OpenC2X, or other software that emulates the ITS stack. Additionally, none of the controllers currently available offer support for the standardized VANET protocols, necessitating a comprehensive redesign of the system. When all factors are considered, the choice of control technology is ultimately a matter of preference. Smida et al.[64] compared the performance of four of the most popular controllers in an SDVN environment. These were Pythonic Network Operating System, Floodlight, OpenDaylight and ONOS. The article concludes that the best performing controller framework is ODL, although the results demonstrated that ONOS, Floodlight, and ODL exhibited comparable performance when a minimal number of vehicles were involved. In light of the aforementioned stipulations, any of the aforementioned controllers would be an adequate choice. In the absence of a sufficiently compelling argument to make any of the aforementioned controllers the superior choice, the decision was made to select ONOS[86].

7.1.4 Northbound API and Application plane

The choice of either the Northbound API or the Application plane is inconsequential, as it is entirely contingent upon the specific controller in question. Moreover, the present study will concentrate on the lower layers.

7.2 Box version 1

To initiate the practical component of this thesis, the University provided a fully operational device consisting of a PC Engines apu1d4 motherboard, containing an AMD G-series T40E CPU and 4 GB of DRAM. The device also possesses three Realtek RTL8111E Gigabit Ethernet channels, a DB9 serial port, and two external USBs. Beyond these features, the device is equipped with two mPCIe interfaces and one Mini-Serial Advanced Technology Attachment (mSATA) interface. These specifications, along with more specific details, were sourced from the product page and are presented as follows:

- CPU: The device has an AMD G-series T40E, operating at a frequency of 1 GHz. It is comprised of two Bobcat cores, each of which is dual-threaded and capable of supporting 64-bit instructions. Each core has 32K of data, 32K of instructions, and 512K of level 2 cache.
- RAM: 4 GB of DDR3-1066 DRAM

- Storage: the device may be booted from an Secure Digital (SD) card connected via USB, an external USB drive, or a mSATA SSD. The system's mSATA interface is equipped with the requisite power connector for the SSD to function.

- Power supply: the device consumes approximately 6 to 12 W at 12 V DC, with the exact value depending on the CPU load. The jack is 2.5 mm and center positive.

- Connectivity: The device features three Gigabit Ethernet channels, which are Realtek RTL8111E.

- Input/Output: The device features a DB9 serial port, four USB ports (two external and two internal), three front panel LEDs, and a pushbutton.

- Expansion: The device features two mPCIe slots, one of which is equipped with a Subscriber Identity Module (SIM) socket.

- Size: The board measures 152.4 x 152.4 mm. The box that encompasses it totals 15.7 cm x 16.8 cm x 2.7 cm, or 16.7 cm x 18.8 cm x 3 cm when all extremities are included.

- Firmware: As part of its low-level system initialization, this device relies on the Coreboot firmware.

- Cooling: A conductive cooling system is employed, whereby the CPU and south bridge are connected to the enclosure via a 3 mm aluminum heat spreader.

The apparatus was furnished with a 16 GB SSD, a Compex WLE200NX mPCIe card for WiFi, and another specialized card for LTE, which is not germane to our present discussion. To support the necessary functionality of the aforementioned cards, the device was also fitted with four pigtail cables. The assembled device can be seen on Figure26.

The device runs on the Debian 12 OS and has been previously configured to enable communication in the capacity of an OBU/ITS vehicle station. The inaugural stage of implementation aimed to ascertain the feasibility of the proposed approach and identify potential avenues for further exploration. It was thus deemed appropriate to commence with an examination of the switching software. The original intention was to install OvS and BMv2 on the aforementioned device and test their compatibility with the existing software. However, this plan was promptly halted due to the device's lack of the necessary resources for installing both programs, and as such we were compelled to deploy OvS exclusively on this device. A series of tests was conducted to evaluate its capabilities, utilizing both a local controller and a remote ONOS controller. The limited capabilities of the devices also motivated us to opt against installing local controllers on each device. This is contrary to our SDN vision, but it is a consequence of the constraints on resources. Another noteworthy obstacle encountered was the insufficient number of test units, which hindered the assessment of the system's wireless communication capabilities between multiple units with a controller.

Figure 26: Back and inside view from device version 1

7.3 Box version 2

The dearth of a sufficient number of devices for testing halted our progress. In light of these circumstances, the procurement of one or more supplementary devices became imperative. To that end, work began with the aim to to design, and subsequently assemble, two new prototype devices with enhanced specifications. Two recently constructed prototypes were assembled on a PC Engines apu3d4 motherboard. The motherboard in question is equipped with an AMD G-Series GX-412TC Embedded processor, 4 GB of DRAM, three Intel i211AT Gigabit Ethernet channels, a DB9 serial port, and two external USB 3.0 ports. The device is additionally equipped with three mPCIe slots, one of which is also capable of functioning as a mSATA. These specifications, along with more specific details, were sourced from the product page and are presented as follows:

- CPU: The device is equipped with an AMD Embedded G-Series GX-412TC, operating at a frequency of 1 GHz. It comprises quad Jaguar cores with 64-bit processing and AES-NI support. Each core has 32 KB of L1 data cache, 32 KB of L1 instruction cache, and a shared 2 MB L2 cache.
- RAM: 4 GB of DDR3-1333 DRAM
- Storage: The device may be booted from an SD card, an external USB connection, or an mSATA SSD. The system's mSATA interface is furnished with the necessary power connector for the SSD to operate, and it can also function as an mPCIe interface.
- Power supply: the device consumes approximately 6 to 12 W at 12 V DC, with the exact value depending on the CPU load. The jack is 2.5 mm and center positive.
- Connectivity: The device features three Gigabit Ethernet channels, which are Intel i211AT.
- Input/Output: The device features a DB9 serial port, two USB 3.0 external ports, and four USB 2.0 internal ports. Additionally, it exhibits three front panel LEDs and a pushbutton.
- Expansion: The device is equipped with three mPCIe slots. Of the aforementioned slots, one is a USB or mSATA slot that can accommodate a SIM card. The second is a USB-only slot that can also accept a SIM card. The final slot is a full mPCIe slot, devoid of a SIM card, but equipped with a WiFi module.
- Size: The board measures 152.4 x 152.4 mm. The box that encompasses it totals 15.7 cm x 16.8 cm x 2.7 cm, or 16.7 cm x 16.8 cm x 3 cm when all extremities are included.
- Firmware: As part of its low-level system initialization, this device relies on the Coreboot firmware.
- Cooling: A conductive cooling system is employed, whereby the CPU is connected to the enclosure via a 3 mm aluminum heat spreader.

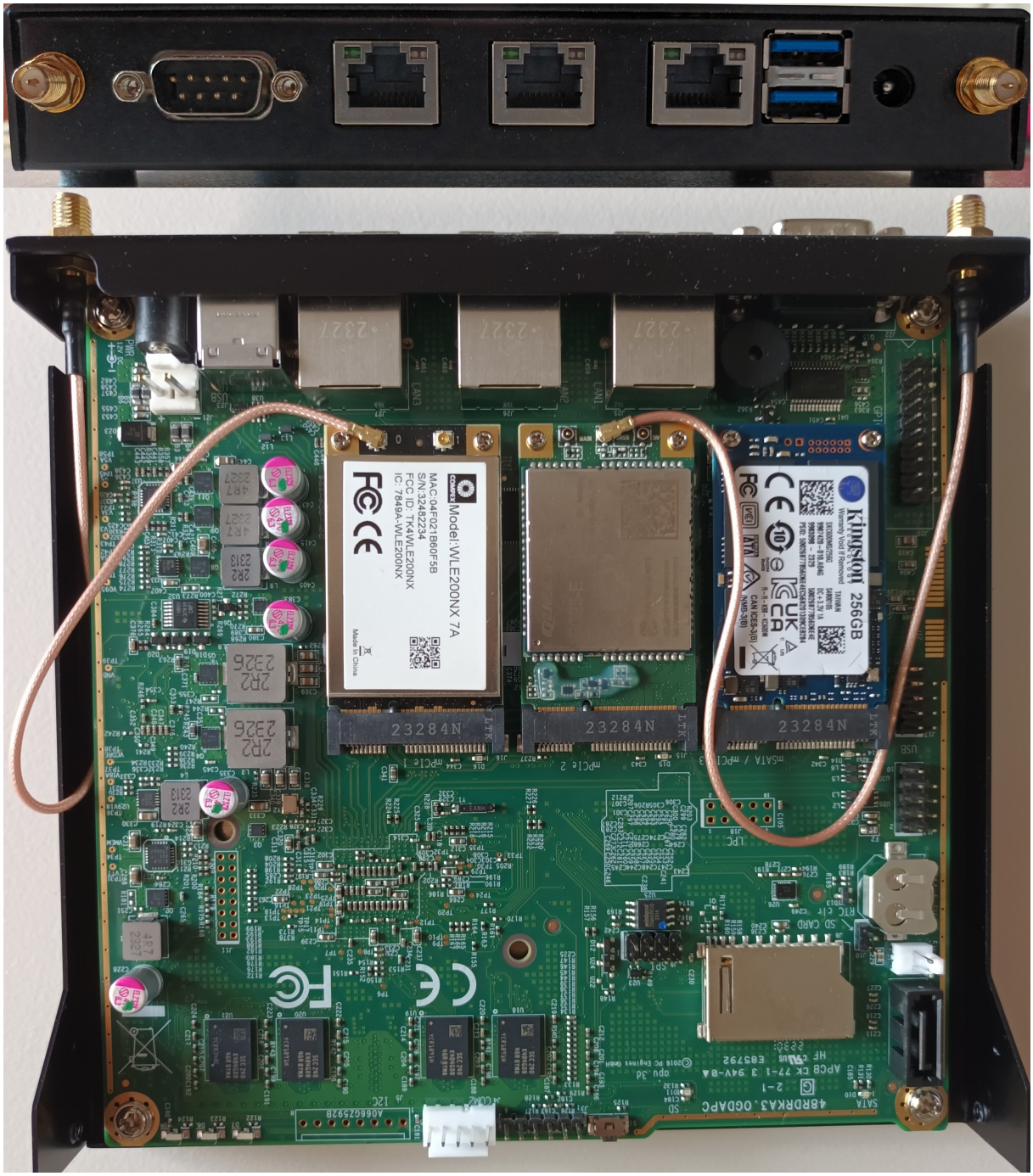


Figure 27: Back and inside view from device version 2

The device was outfitted with a Kingston 256GB SSD, a Compex WLE9000VX 7AA WiFi card, and a Quectel EC25-E for LTE. The prototype case under consideration is only capable of accommodating two pigtail cables. In consideration of our intention to utilize WiFi exclusively for the duration of our investigation, this case is deemed sufficient. Nevertheless, should LTE be required in the future, enhancements to the case will be mandatory. One of the two newly assembled devices can be seen on figure27.

As the device is devoid of any software, we shall adhere to the aforementioned guidelines that have

been previously established. It was thus deemed appropriate to proceed with the Ubuntu 24.04.1 LTS, given that the visual interface of the OS was considered to be of no consequence and superfluous. A minor complication manifested as a result of the incompatibility between the selected WiFi card and the existing solution. During the device setup process, it became evident that, although the WiFi card was from the atheros family, it utilized the ath10k driver. The ath10k driver has been developed as a more recent iteration of the ath9k driver. However, at the time of writing, no patch for 802.11p has been made available for this driver. This necessitated the pursuit of a different chip that was compatible with the ath9k driver. Fortunately, the Linux wireless community maintained an up-to-date list of products that met the necessary criteria[87]. From the list of available options, the Compex WLE200NX 7A was selected as the most suitable choice.

8 Software Tests

In this section, the selected prototype hardware will be tested in conjunction with the desired software. The objective of these tests is to ascertain the viability of the software selected for practical deployment of the prototypes. Specifically, this investigation sought to evaluate the viability of both OvS and BMv2 in vehicular scenarios. In particular, the aim of the forthcoming testing will be to ascertain the degree of compatibility between the aforementioned switching emulation software and a number of other components, including the ath9k patch, the ONOS controller, and the hardware of the device in question.

8.1 Experimental Setup

In consideration of the practical implications of the goals of this thesis, the device must meet the criteria for an OBU/ITS vehicle subsystem, in accordance with the SDN paradigm. For this, the device must be categorized as both SDN and VANETs. In practical terms, for a device to be considered SDN-compliant, it is necessary to utilize software to emulate the functions of an SDN device. Moreover, the minimum requirement for a device to be appropriate for the context of VANETs is its capability of establishing connectivity in accordance with the 802.11p standard. Accordingly, the study employs a three-stage experimental methodology, consisting of installation, integration, and functionality verification. The tests aspire to be performed in a series of phases, with each subsequent phase building upon the preceding one and introducing greater degrees of complexity. Their objective is to assess the compatibility of the various elements utilized, namely the hardware, the ath9k patch, and OvS/BMv2. The success of this endeavor is contingent upon the establishment of a connection between devices, which will be conducted using the ping tool. The experiments were conducted on the devices described in Section7.2 and7.3 with the assistance of an external device, namely a personal computer. Additionally, a switch and six network cables were employed to facilitate communication between the devices and the personal computer, while omnidirectional antennas operating at 6GHz were utilized to enable communication in the requisite frequency range. The software tools utilized in this investigation were as follows: Windows 10 is the operating system of the PC; Oracle VM Virtualbox 7.0.4 was installed on the host computer and employed to both install and run the ONOS and p4c; ONOS 2.7.0 was utilized for the controller; p4c is the compiler for the P4 language; PuTTy Release 0.80 was installed on the PC and is used to interface with the machines; Ubuntu 24.04.1

LTS operating system is used on devices 2; Debian 12 operating system is used on device 1.0; OvS and BMv2 are the focus of the current testing. It is essential to acknowledge that, despite the comprehensive and rigorous nature of these tests, they are not without inherent limitations. Such limitations include the specific conditions of the controller environment and the exclusion of real-world network conditions. The methodology selected was chosen for its capacity to facilitate the systematic identification of critical compatibility issues that must be addressed for successful real-world deployment. Moreover, for the purposes of this study, it was assumed that the devices lacked the requisite resources to install and run ONOS.

8.2 OvS

This experiment is intended to assess the compatibility and practicality of integrating OvS and OpenFlow with VANET technologies. The initial stages of the experiment were conducted using a single device. Around the midpoint of experimentation, the additional two 2 boxes arrived and were incorporated into the test setup.

8.2.1 Phase 1

The opening phase of the experiment sought to examine the functionality of OvS in a relatively straightforward scenario, with the additional objective of becoming familiar with the technology. Accordingly, the procedure commenced with a standalone examination of OvS, which is possible due to the fact that OvS is equipped with a local controller that is capable of autonomously managing network traffic. OvS was installed on the device, and a virtual network was created within the device using VRFs and virtual interfaces.

The setup from this phase can be observed in Figure28, and is diagrammatically represented in Figure29.

8.2.1.1 Results

When viewed through the lens of SDN, the configuration of this phase can be conceptualized as illustrated in Figure30.

The configuration of OvS, together with information of the devices interfaces can be seen in Figure31.

Information about the two VRFs, as well as connectivity between them is displayed in Figure32.

8.2.2 Phase 2

Following the success of this initial phase, the decision was made to proceed with the introduction of the remote controller. The controller, which will be utilized for the remainder of the tests, introduced an elevated level of complexity, providing an excellent opportunity to become acquainted with ONOS environments. Consequently, phase two entailed the creation of a Linux-based Virtual Machine (VM) utilizing

Figure 28: Setup from the first phase of the experiments with OvS

VirtualBox, with the installation of ONOS on this VM. ONOS was therefore incorporated into the preceding scenario and connected to OvS via OpenFlow.

The setup from this second phase is visualy the same as in the first phase, and can therefore be observed in Figure28. The setup of this phase is diagrammatically represented in Figure33.

8.2.2.1 Results

When viewed through the lens of SDN, the configuration of this phase can be conceptualized as illustrated in Figure34.

The configuration of OvS, together with showing connectivity between the two VRFs is displayed in Figure35.

The list of devices and hosts known to the controller, along with the flows that have been injected into the devices, is presented in Figure36.

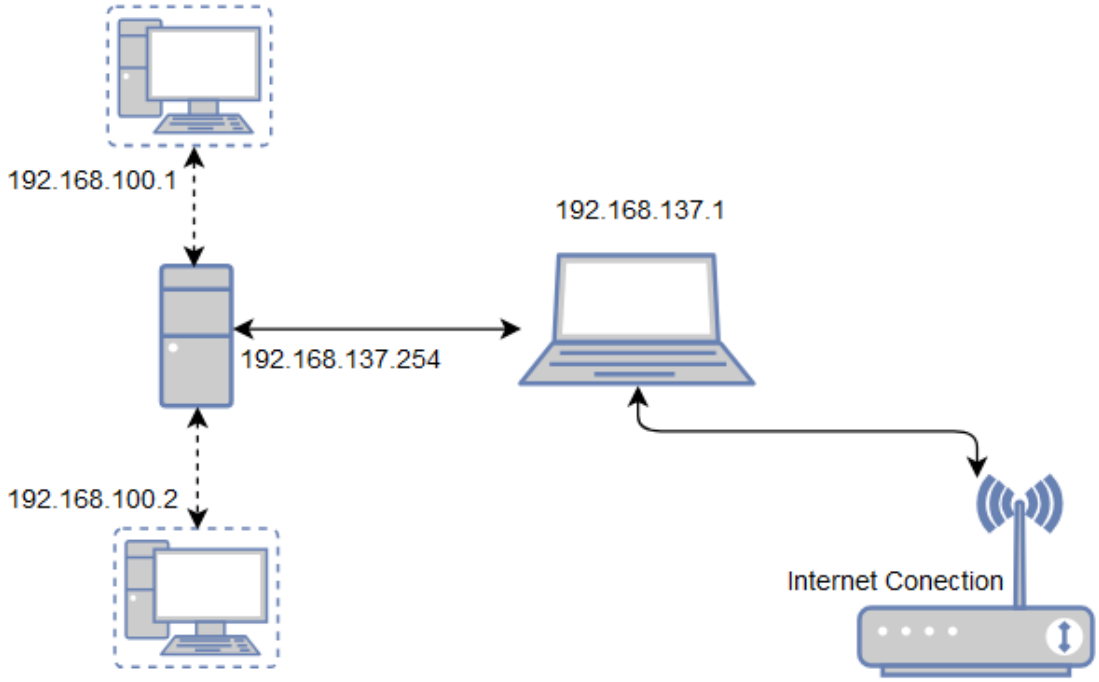


Figure 29: Diagram that represents the setup from the first phase of the experiments with OvS

8.2.3 Phase 3

By the time phase 3 was reached, the two additional devices had been received, thus enabling the substitution of the virtual interfaces with their physical counterparts. This was achieved by connecting the aforementioned two additional devices to the primary unit on which OvS had previously been installed. The objective of this test was to verify the functionality of OvS with non-virtual interfaces.

The setup from this third phase can be observed in Figure37. The setup of this phase is diagrammatically represented in Figure38.

8.2.3.1 Results

From the perspective of SDN, the network configuration of this phase is identical to that of the previous phase. Consequently, it can be observed in Figure34.

The configuration of OvS, together with showing connectivity between the two other devices is displayed in Figure39.

The list of devices and hosts known to the controller, along with the flows that have been injected into the devices, is presented in Figure40.

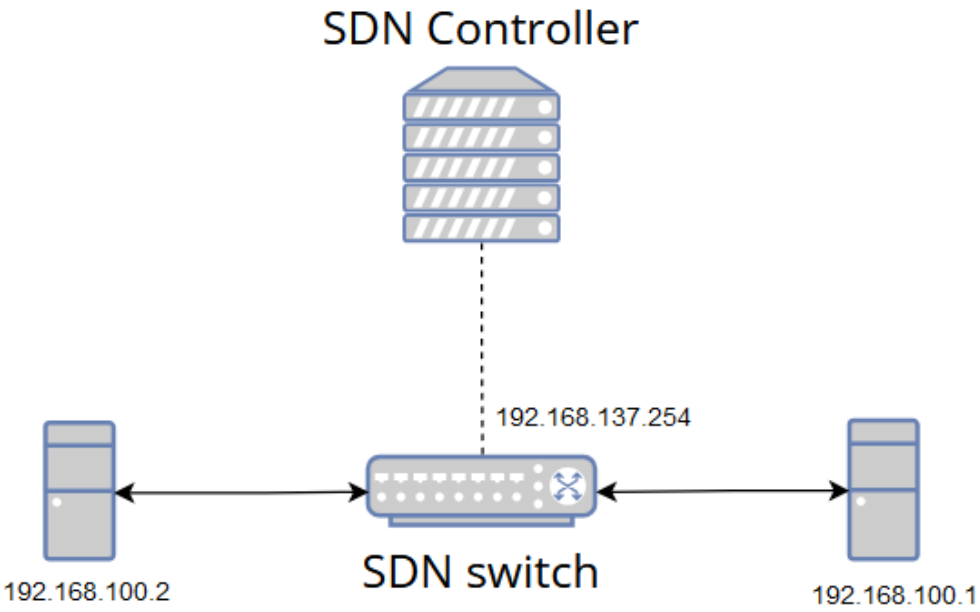


Figure 30: Diagram that depicts the SDN perspective of the setup from the first phase of the experiments with OvS

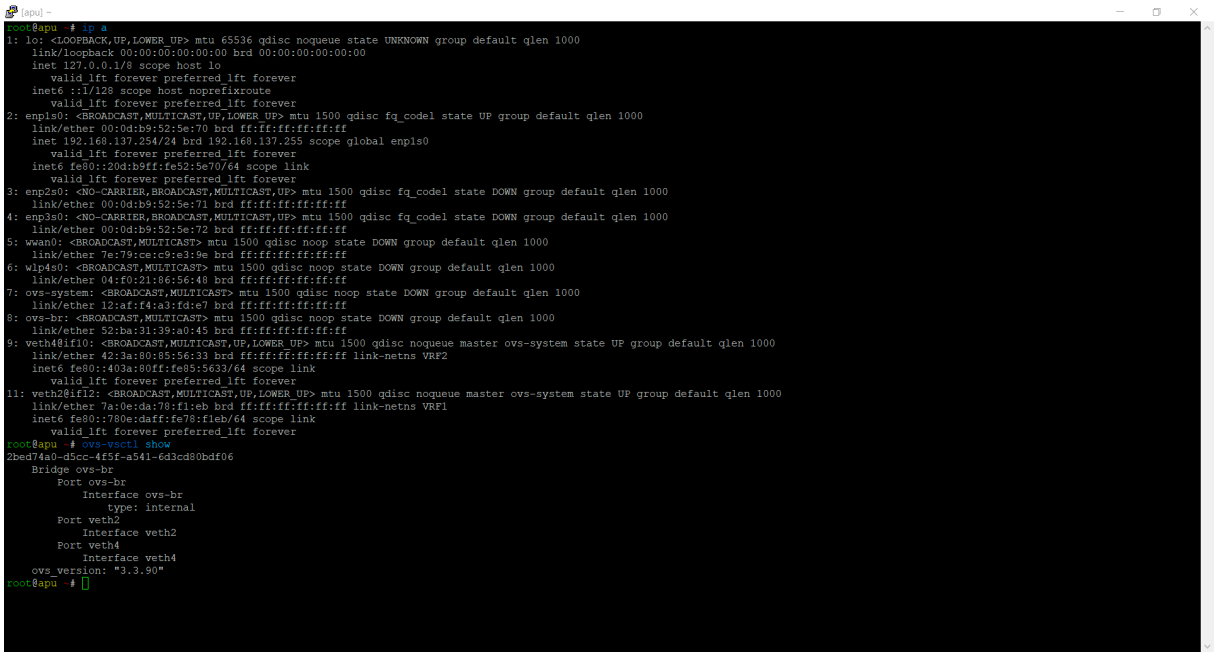


Figure 31: Results from running the commands "ip a" and "ovs-vsctl show" in the device used in phase 1OvS

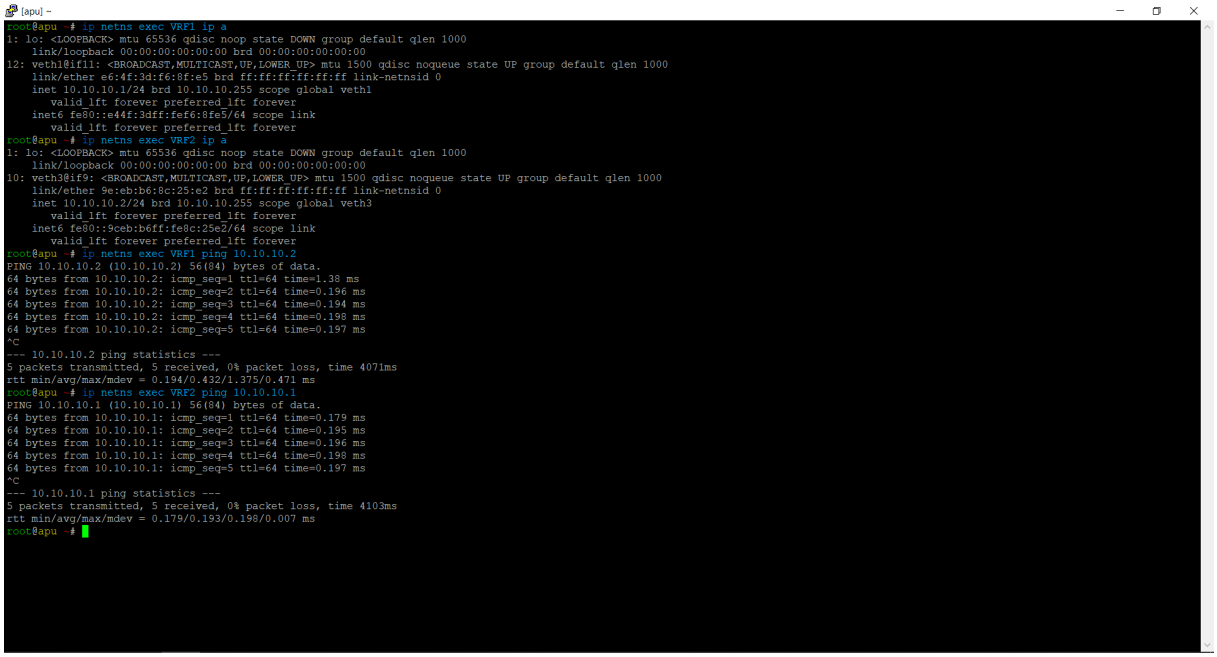


Figure 32: Results from running the commands "ip a" on both VRFs and "ping" between them

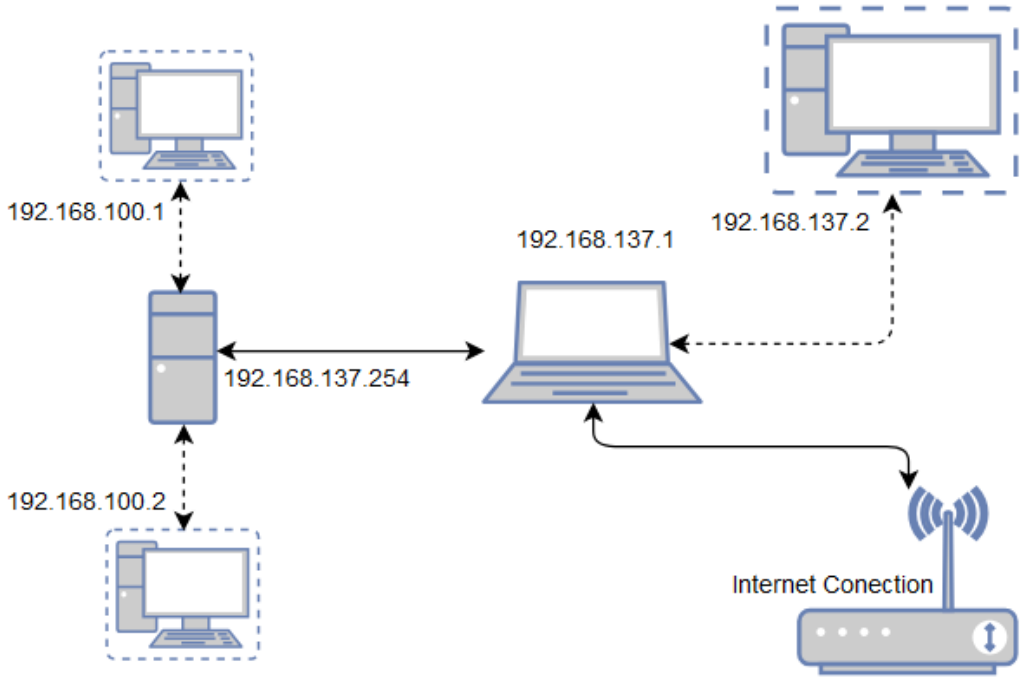


Figure 33: Diagram that represents the setup from the second phase of the experiments with OvS

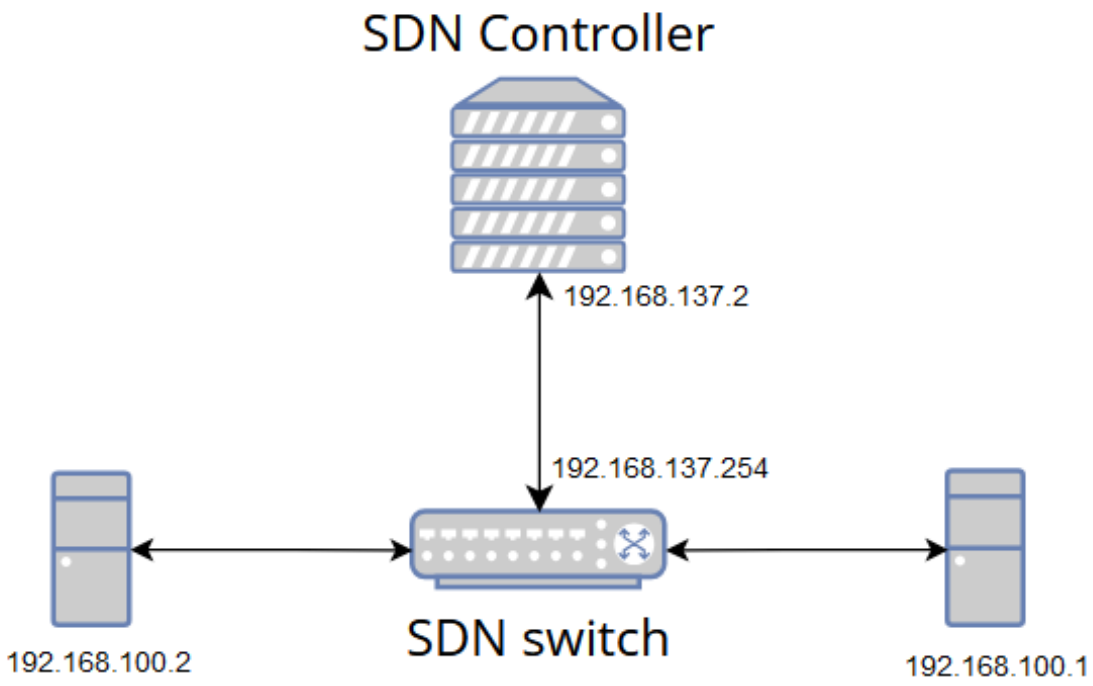


Figure 34: Diagram that depicts the SDN perspective of the setup from the second phase of the experiments with OvS

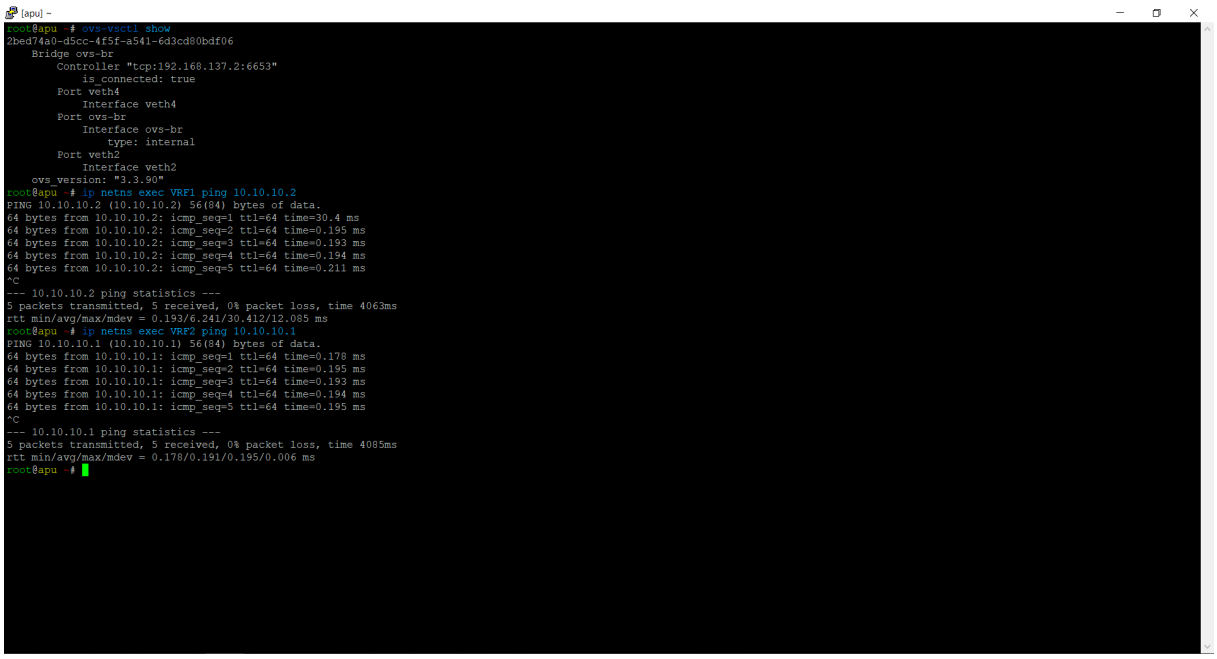


Figure 35: Results from running the commands "ovs-vsctl show" and "ping" between both VRFs

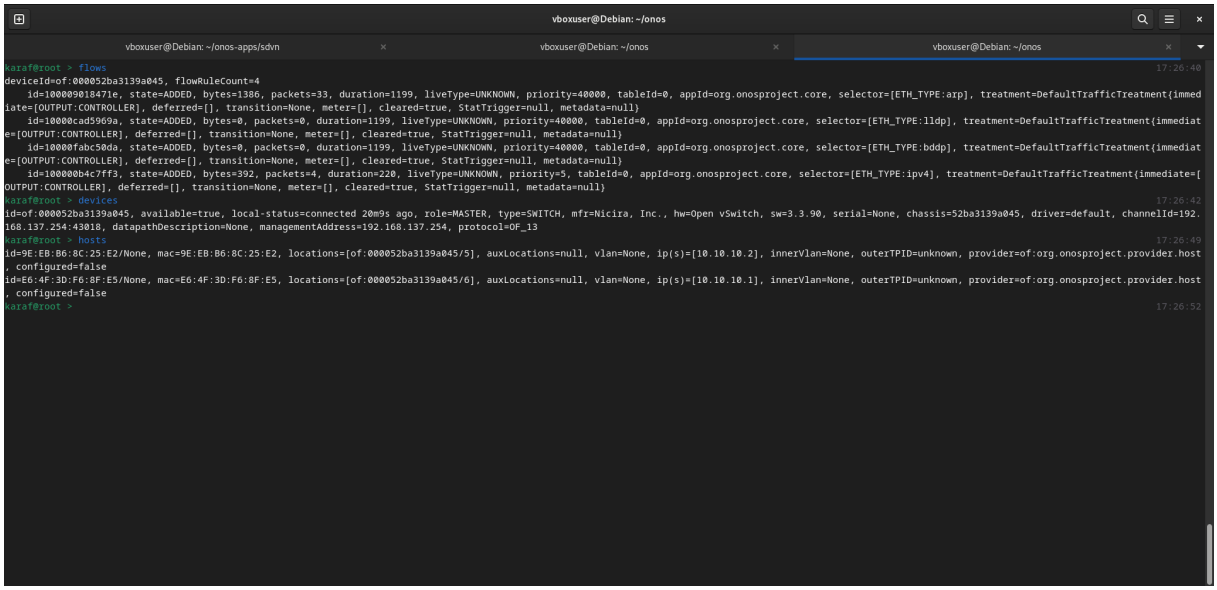


Figure 36: ONOS console with information relating to the devices, hosts and flows of the network

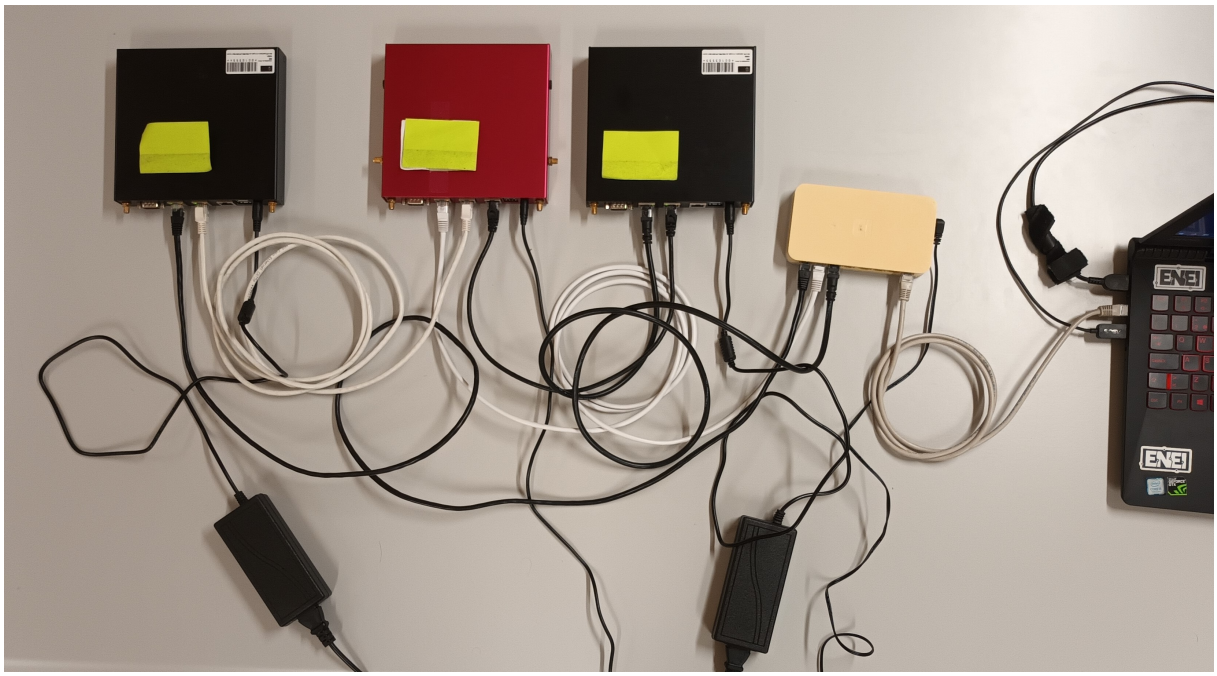


Figure 37: Setup from the third phase of the experiments with OvS

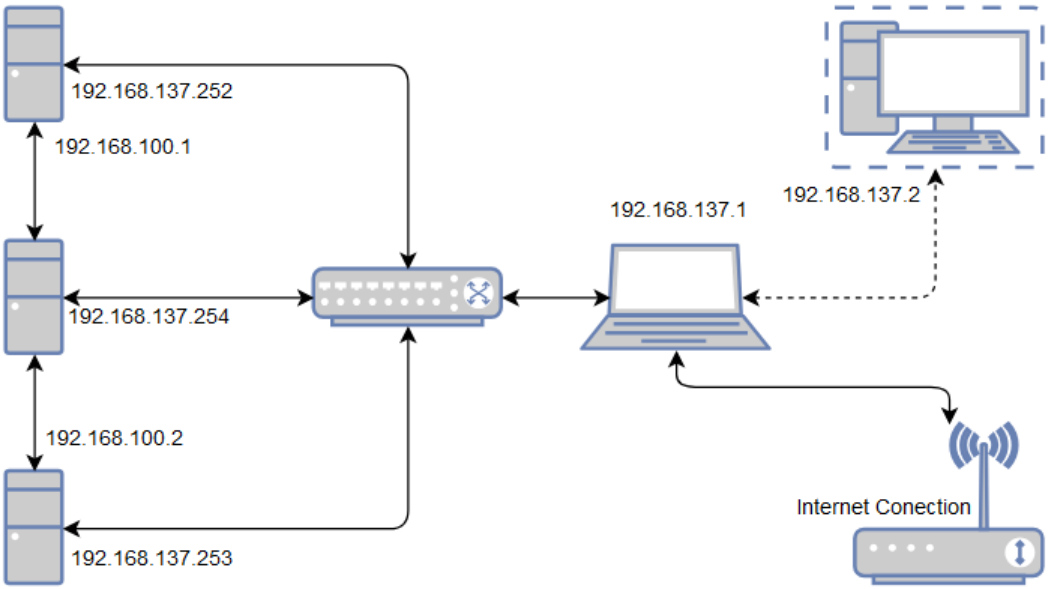


Figure 38: Diagram that represents the setup from the third phase of the experiments with OvS

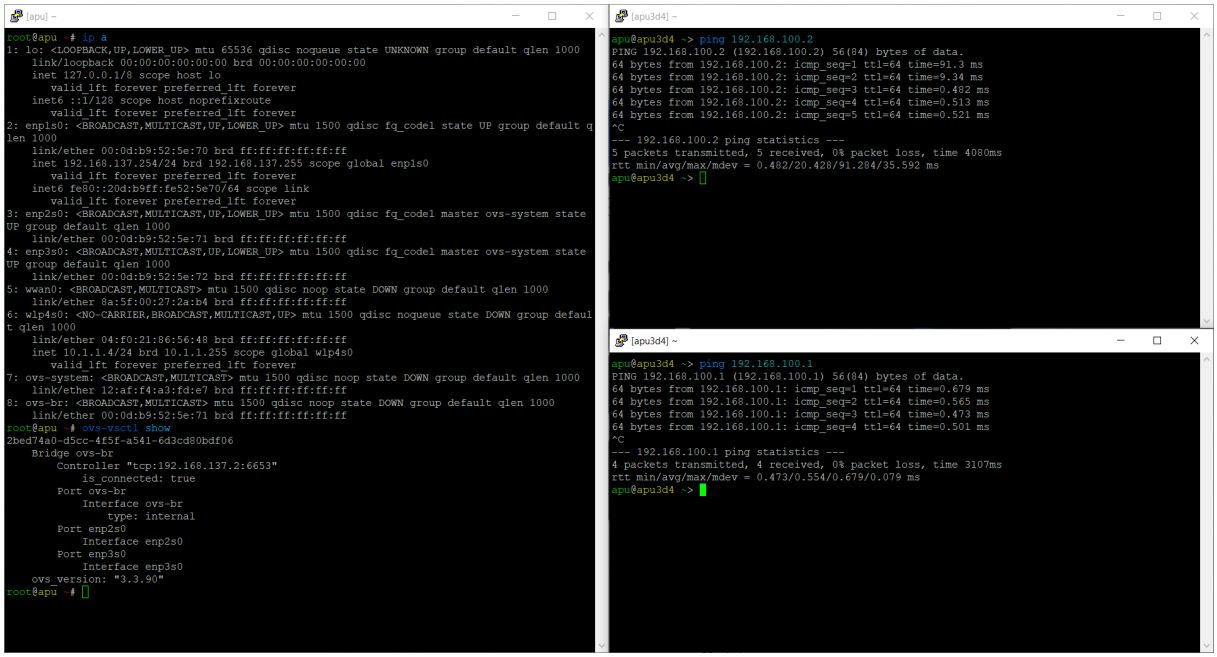


Figure 39: Results from running the commands "ip a" and "ovs-vsctl show" on the device running OvS and "ping" between the other two devices

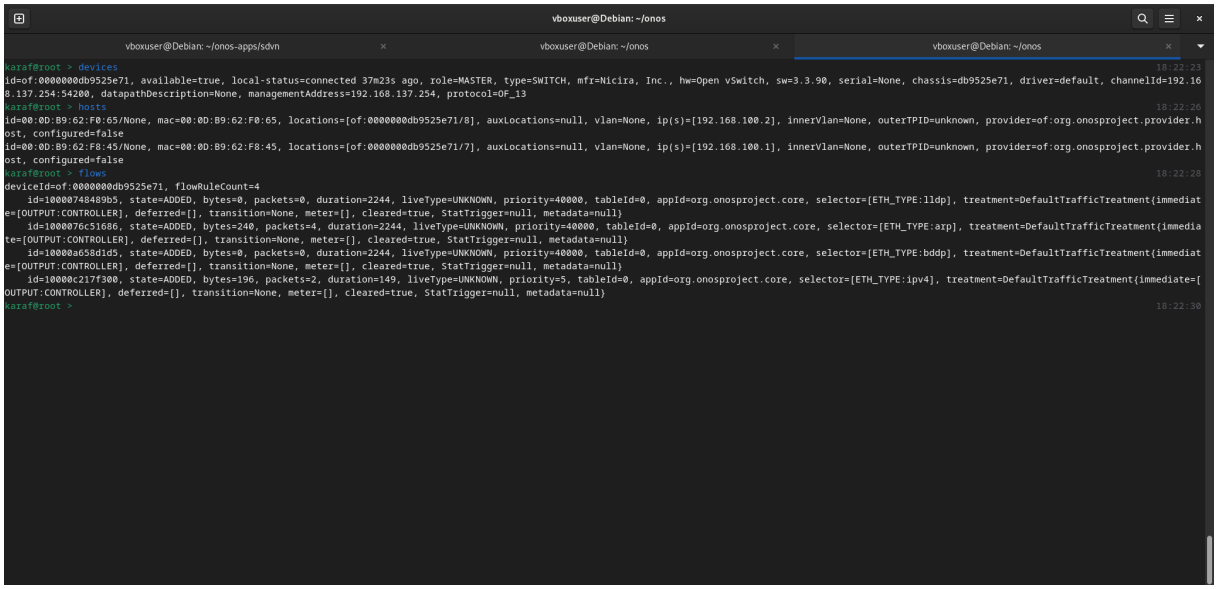


Figure 40: ONOS console with information relating to the devices, hosts and flows of the network

8.2.4 Phase 4

In Phase 4, the test SDN network was expanded to integrate the three devices by installing OvS on the two new devices. Consequently, the three devices were linked together and connected to the ONOS controller, and two VRFs were created, each situated on opposite edges of the network.

This configuration is equal to the last phase, so it is illustrated in Figure 37. The setup of this phase is diagrammatically represented in Figure41 and described in the following itemized list:

- 192.168.137.1 - This IP address identifies the author's personal laptop computer. It serves as the central coordinating hub for the entire network, facilitating the observation and monitoring of all network activity.

- 192.168.137.2 - This address represents a virtual machine that is hosted on the author's laptop via the Oracle VM VirtualBox virtualization platform. The virtual environment is configured with Debian 12 and is primarily intended for the deployment of ONOS. This configuration is essential due to the fact that the primary branch of ONOS is designed to run on Linux, whereas the author's device runs on Windows.

- 192.168.137.252 - This corresponds to a device 2. It has a fresh installation of the Ubuntu Server operating system and is running OvS.

- 192.168.137.253 - This device is virtually identical to the device 192.168.137.252.

- 192.168.137.254 - This address represents device 1.0 that was received from the university. This device has debian 12 and is running OvS.

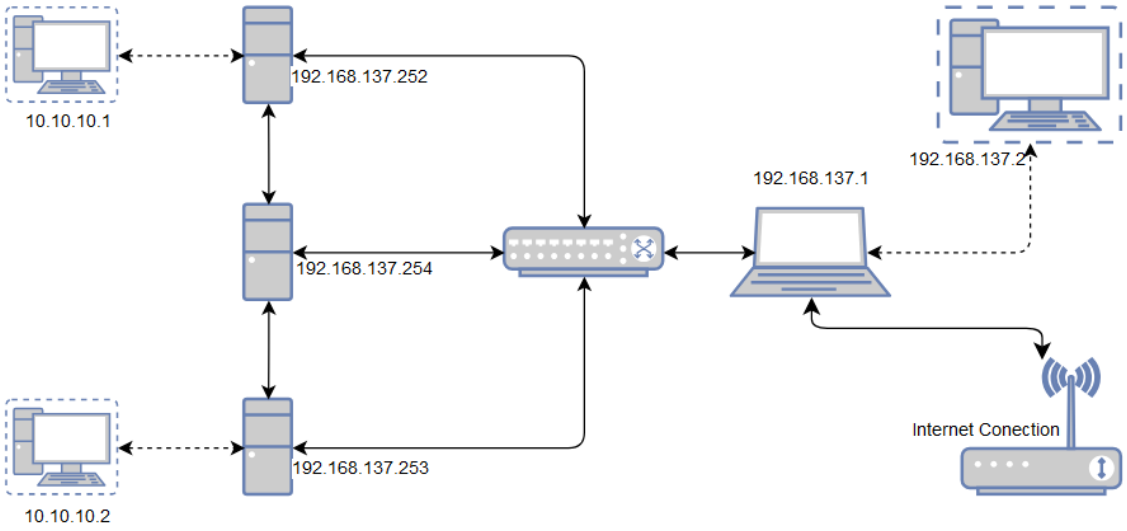


Figure 41: Diagram that represents the setup from the fourth phase of the experiments with OvS

- 10.10.10.1 - This IP address identifies the virtual interface that has been added to the VRF instance created in the 192.168.137.252 device.
- 10.10.10.2 - Identically to the last one, this IP address identifies the virtual interface that has been added to the VRF instance created in the 192.168.137.253 device.

8.2.4.1 Results

When viewed through the lens of SDN, the configuration of this phase can be conceptualized as illustrated in Figure42.

The configuration of the three instances of OvS that are running is Figure43.

Connectivity between the two VRF is demonstraded in Figure44. It illustrates the connectivity between devices 192.168.137.252 and 192.168.137.253, manifesting through the VFRs with IPs 10.10.10.1 and 10.10.10.2.

The list of devices and hosts known to the controller is shown in Figure45, and the flows that have been injected into the devices are presented in Figure46.

8.2.5 Analysis and Interpretation

It was intended for the subsequent phase of the investigation to involve an attempt to introduce the ath9k patch and wireless antennas for use in conjunction with the OvS system. However, Phase 4 proved to be the final phase of the investigation. Through additional research it was discovered that OvS is unable to function with wireless interfaces, which was is corroborated by a statement on the official wiki[88] indicating that OvS is not compatible with wireless interfaces. Furthermore, the testing phase revealed

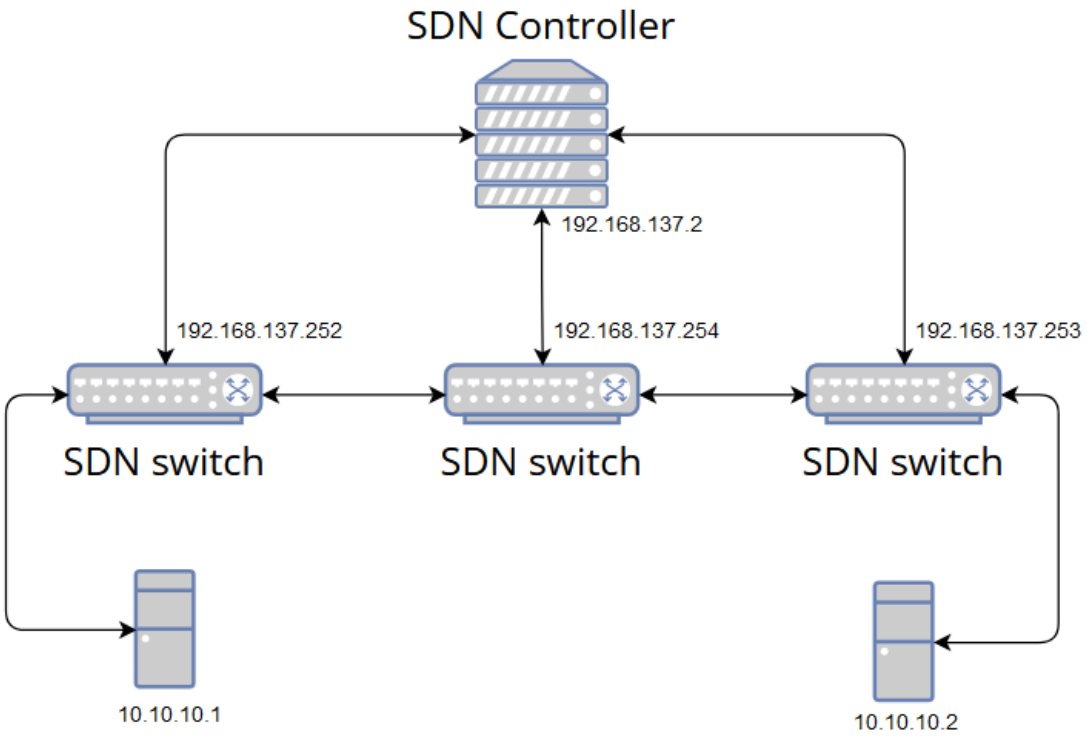


Figure 42: Diagram that depicts the SDN perspective of the setup from the fourth phase of the experiments with OvS

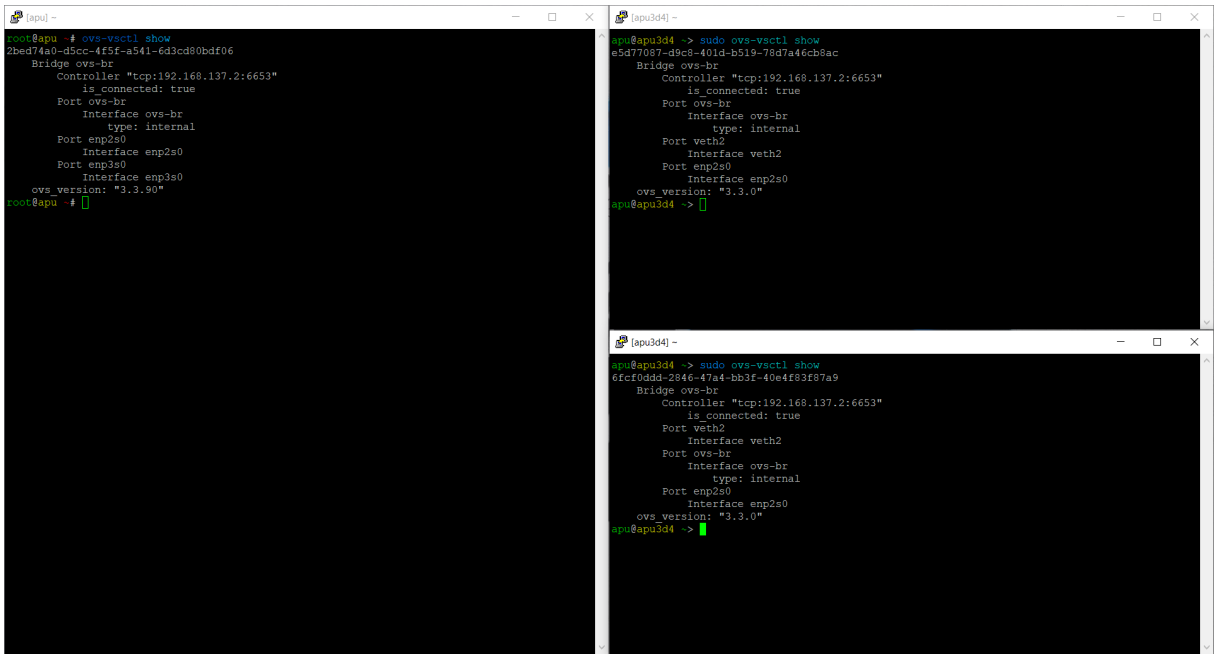


Figure 43: Results from running the commands "ovs-vsctl show" in the devices running OvS

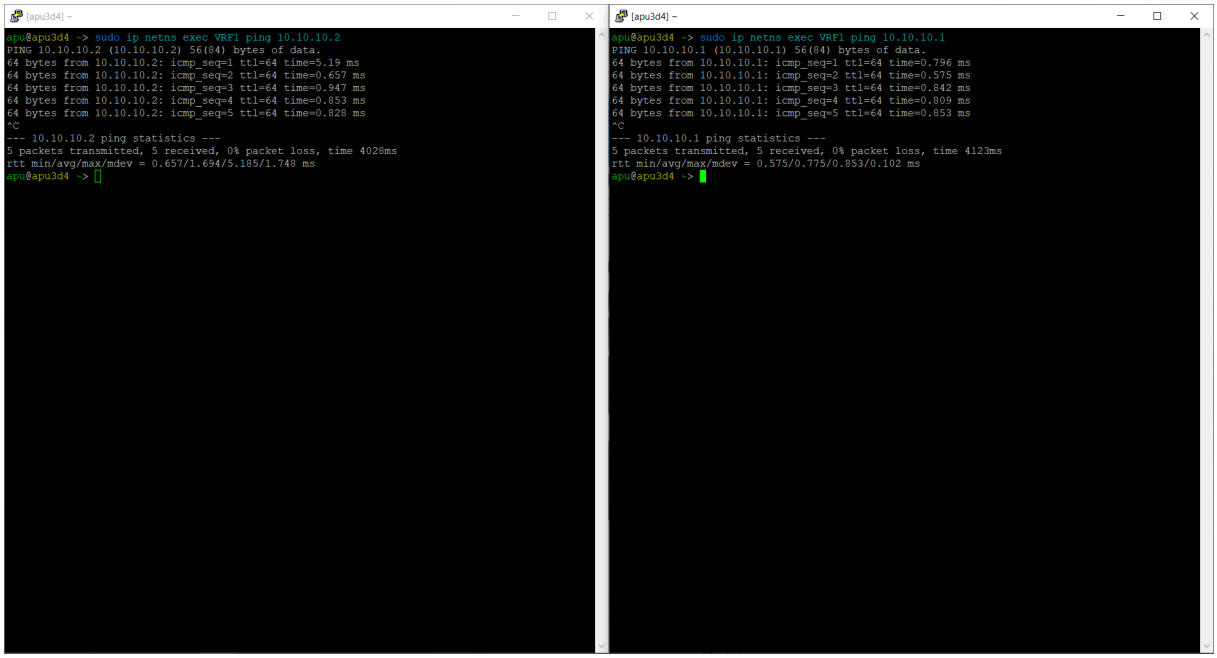


Figure 44: Results from running the commands "ping" in the VRF between the two devices on the edge of the network

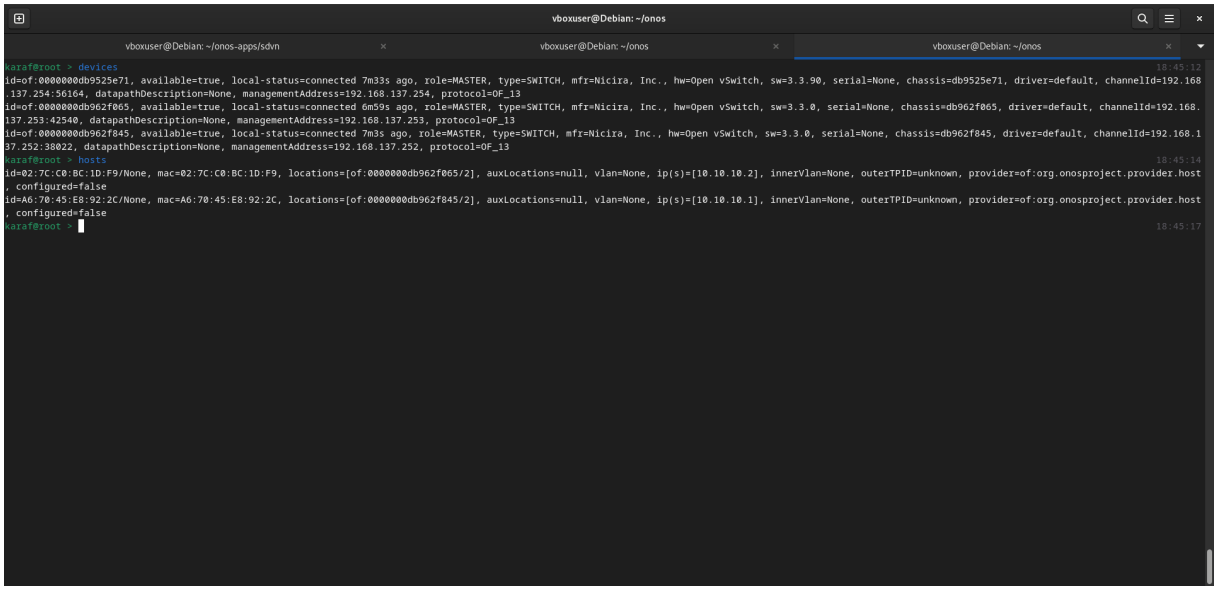


Figure 45: ONOS console with information relating to the devices and hosts of the network

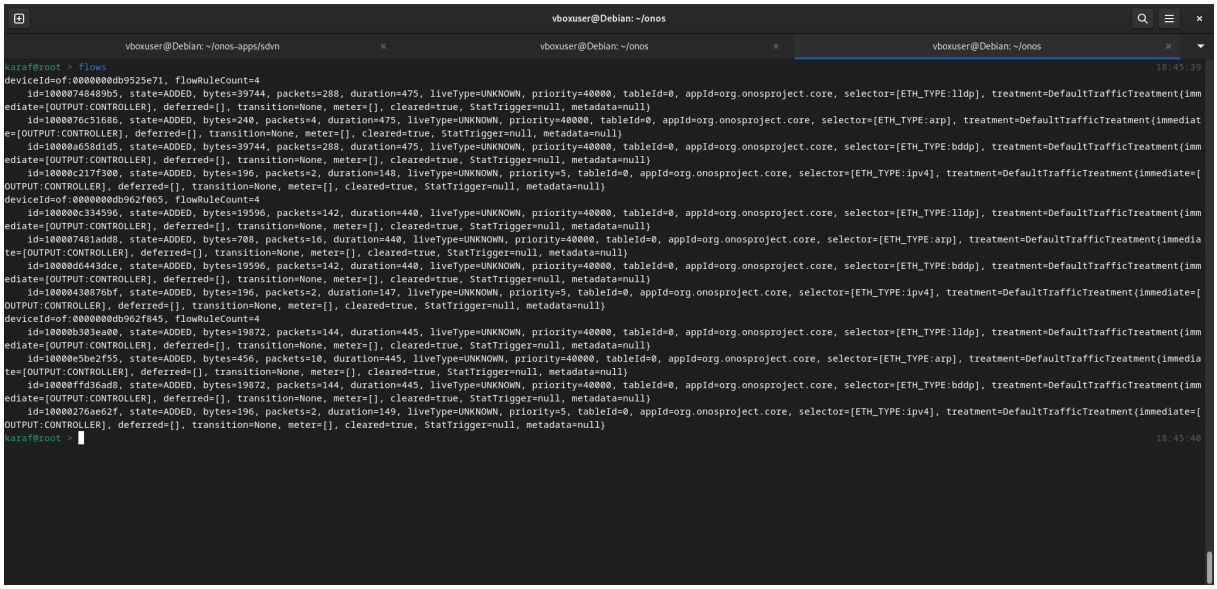


Figure 46: ONOS console with information relating to the flows of the network

another significant limitation in the capabilities of OvS. Despite its compatibility with OpenFlow, OvS is constrained by the traditional definition of a switch, thereby limiting its functionality to layer 2. This inherent restriction disables the system from parsing packets from different subnets. In conclusion, it was demonstrated that the use of OvS is not viable within the context of vehicular scenarios. These findings served to reinforce the hypothesis that OpenFlow-based solutions are not a viable option for use in vehicular environments. As anticipated and corroborated in Section6, OpenFlow is only compatible with traditional software protocols, thereby precluding the use of vehicular protocols under the circumstances presented here.

8.3 BMv2

The objective of this experiment is to evaluate the compatibility and practicality of integrating BMv2 and P4 with VANET technologies. The experiments were conducted using two 2-box apparatuses.

8.3.1 Phase 1

Phase 1 of this experiment will focus on assessing the fundamental operational capabilities of BMv2, thereby offering an opportunity for familiarization with this environment. In order to accomplish this goal, the first step is installing BMv2 on the target devices. It became evident that this process was more intricate than initially anticipated, due to the fact the southbound API of BMv2 was, by default, not P4Runtime, but rather Thrift. Following an investigation into the matter, the existence of an option to install supplementary components to BMv2 was identified, thereby enabling communication via the GRPC-based P4Runtime. The modification was implemented, allowing for the establishment of a connection between the devices and ONOS. The second phase of the testing process was a prerequisite for evaluating the functionality

of the system. In this networking paradigm, the functionality of the network is contingent upon the code that runs on it. It was thus imperative to obtain a P4, and given that the controller is implemented using ONOS, the P4 code must be accompanied by Java code for the controller.

At this juncture, our intention was to ascertain the functionality of the software, for which a simple P4/ONOS application would suffice. Fortunately, ONF, as part of the Aether Project, offers a practical tutorial to teach the fundamental principles of the next-generation SDN architecture. The tutorial was developed for use with Mininet and Stratum, which immediately indicated the necessity for modifications. A more thorough examination of the code revealed the presence of additional issues and errors. It was thus necessary to modify a substantial portion of the code to guarantee its compatibility with the specified environment before it could be utilized. Indeed, a considerable investment of time and effort was necessary to make the program operational. Given these circumstances that require us to engage in the process of writing code, it stands to reason to invest our time in developing a more final solution, rather than expending effort on a generic solution. Therefore, the code we will be writing is already intended to work in an ad hoc scenario with minimal modifications. In order to achieve the desired functionality, it proved indispensable to develop a P4 code that was tailored for this specific scenario. At the outset of the practical component of the thesis, the original intention was to develop a solution that more closely resembled the ETSI protocols. However, the decision was ultimately made not to pursue this approach for two main reasons. It is first necessary to acknowledge the inherent complexity of the ETSI protocols. The standards under discussion have been developed over the past two decades by the most esteemed engineers in the field. Given the considerable time and resources required, it is neither realistic nor reasonable to view this as a feasible task, particularly in view of the limitations in terms of human resources and time imposed by the context of this master's thesis. Secondly, the ETSI norms establish the indispensable presence of the remaining subsystems as a mandatory condition for attaining the standardized functionality. These circumstances provided the impetus to pursue an alternative and more appropriate course of action. In lieu of attempting to create an exact replica of the ETSI protocol using SDN technology, It was elected as more appropriate to pursue the development of a proof-of-concept solution. This proposed endpoint has the dual objective of validating the compatibility of P4 with wireless antennas and 802.11p, while simultaneously demonstrating the adaptability of P4 functional capabilities within these specific contexts. In order to validate the first proposition, the proposed framework will rely on broadcast communication strategies within the wireless domain. The second part will be illustrated through the use of custom headers. While there are numerous potential applications for custom headers, for the purposes of this study, we sought to specify an appropriate research objective. Ultimately, it was determined that the prevention of the perpetual repetition of broadcast messages was a necessary measure to avoid severe network saturation and therefore an appropriate goal. To this end, we created a "marker" system. The markers are unique identifiers generated by the controller and appended to each message transmitted via the wireless antennas. This strategy guarantees that no message is broadcast twice by the same device. The final solution was developed into a P4/ONOS application and the result was published to github[89].

All programming languages depend on a compiler for their functionality, and P4 is no exception. Prior to engaging in any coding activities within the P4 environment, it was indispensable to install the p4c[90].

This step was carried out within the VM that houses the ONOS system. The ONOS system is capable of automatically injecting the P4 code into the devices, thereby eliminating the need for manual transfer of the file on each occasion. Once the requisite setup had been completed, two VRFs were created, one on each device. Subsequently, the aforementioned VRF instances were linked to their corresponding devices via the use of virtual interfaces. Subsequently, BMv2 was initiated, with one virtual and one physical interface allocated to it. Subsequently, ONOS was activated, along with the requisite ONOS/P4 application and all associated dependencies. At last, the configuration was applied to the controller, thereby informing the controller of the devices and providing it with the necessary information to perform its functions.

The test configuration is illustrated in Figure47, which was also used as the basis for the diagram seen in Figure48. A description of said diagram is in the following itemized list:

- 192.168.137.1 - This IP address identifies the author's personal laptop computer. It serves as the central coordinating hub for the entire network, facilitating the observation and monitoring of all network activity.
- 192.168.137.2 - This address represents a virtual machine that is hosted on the author's laptop via the Oracle VM VirtualBox virtualization platform. The virtual environment is configured with Debian 12 and is primarily intended for the deployment of ONOS. This configuration is essential due to the fact that the primary branch of ONOS is designed to run on Linux, whereas the author's device runs on Windows.
- 192.168.137.252 - This corresponds to a device 2. It has a fresh installation of the Ubuntu Server operating system and is running BMv2 with grpc.
- 192.168.137.253 - This device is virtually identical to the device 192.168.137.252.
- 10.10.10.1 - This IP address identifies the virtual interface that has been added to the VRF instance created in the 192.168.137.252 device.
- 10.10.10.2 - Identically to the last one, this IP address identifies the virtual interface that has been added to the VRF instance created in the 192.168.137.253 device.

8.3.1.1 Results

Figure49 shows BMv2 running in both devices.

Figure50 illustrates the connectivity between devices 192.168.137.252 and 192.168.137.253, manifesting through the VRFs with IPs 10.10.10.1 and 10.10.10.2.

When viewed through the lens of SDN, the physical configuration can be conceptualized as illustrated in Figure 51.

The list of devices and hosts known to the controller is shown in Figure52, and the flows that have been injected into the devices are presented in Figure53. The P4/ONOS application made for this phase can be seen in Figure54.

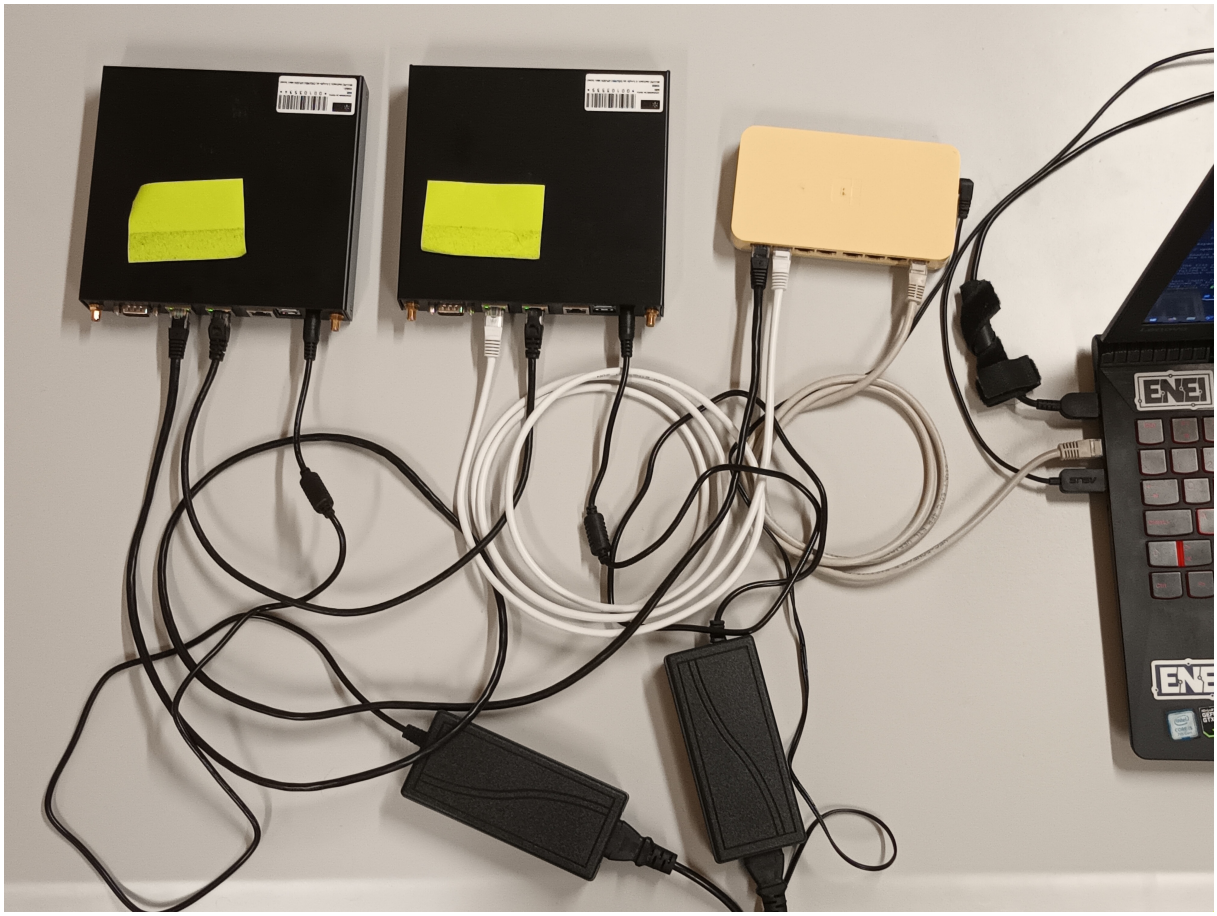


Figure 47: Setup from the first phase of the experiments with BMv2

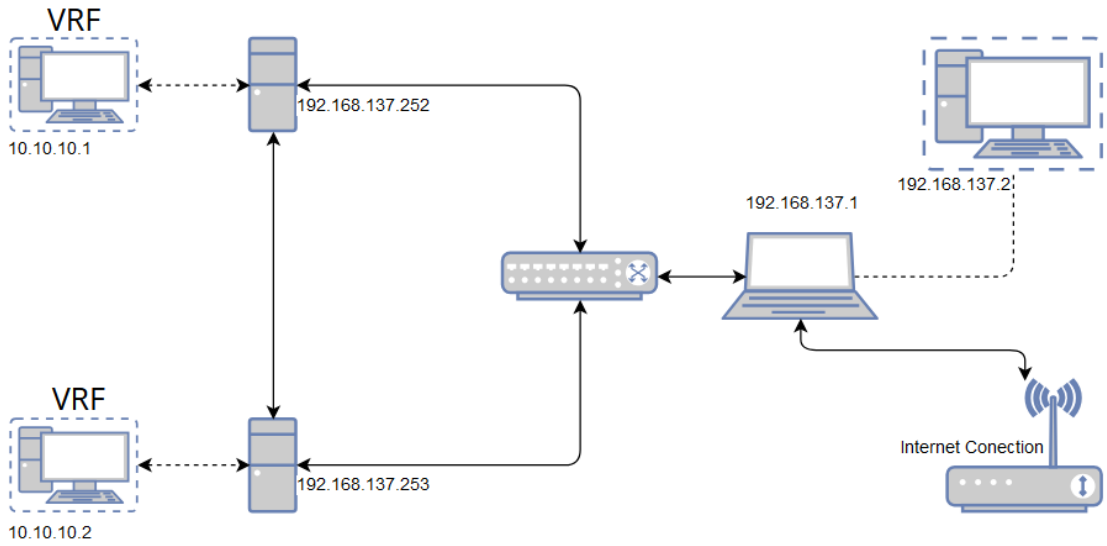


Figure 48: Diagram that represents the setup from the first phase of the experiments with BMv2

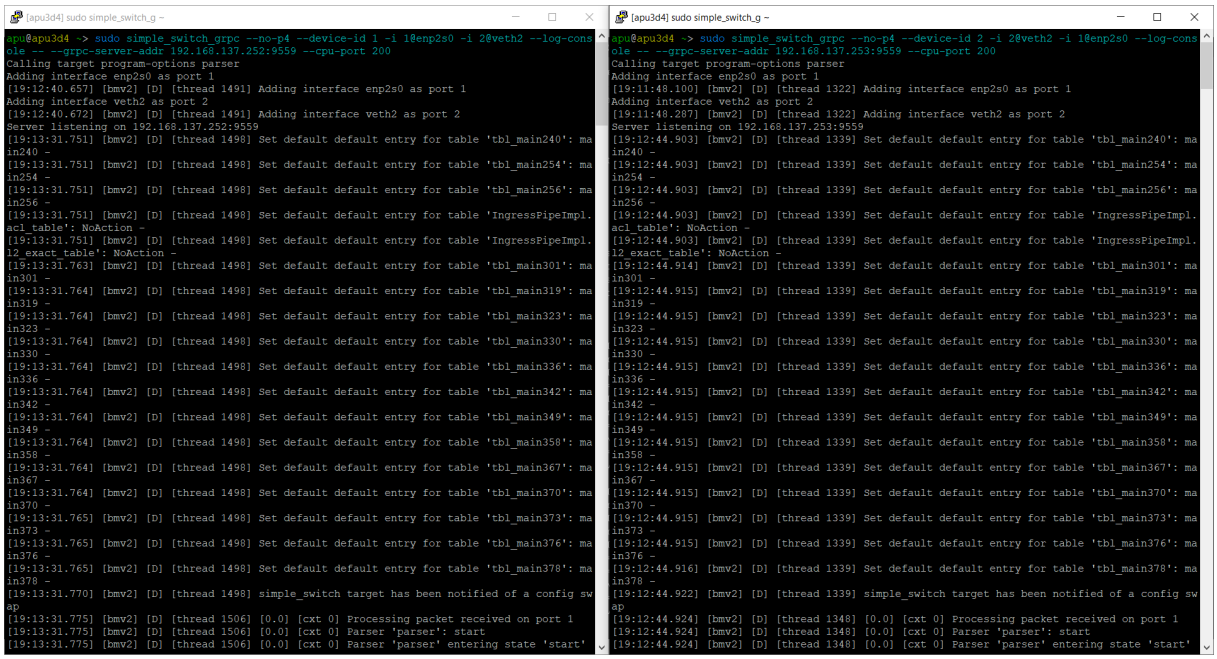


Figure 49: Consoles running BMv2 on both devices

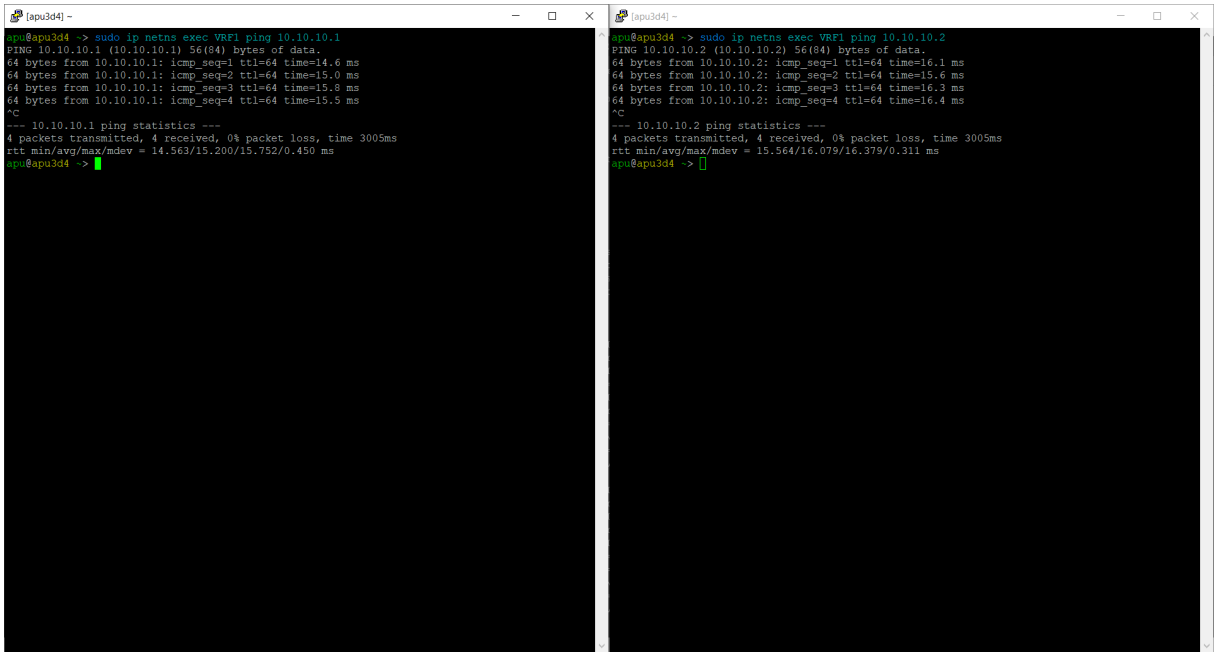


Figure 50: Results from running the commands "ping" in the VRF between the two devices on the network

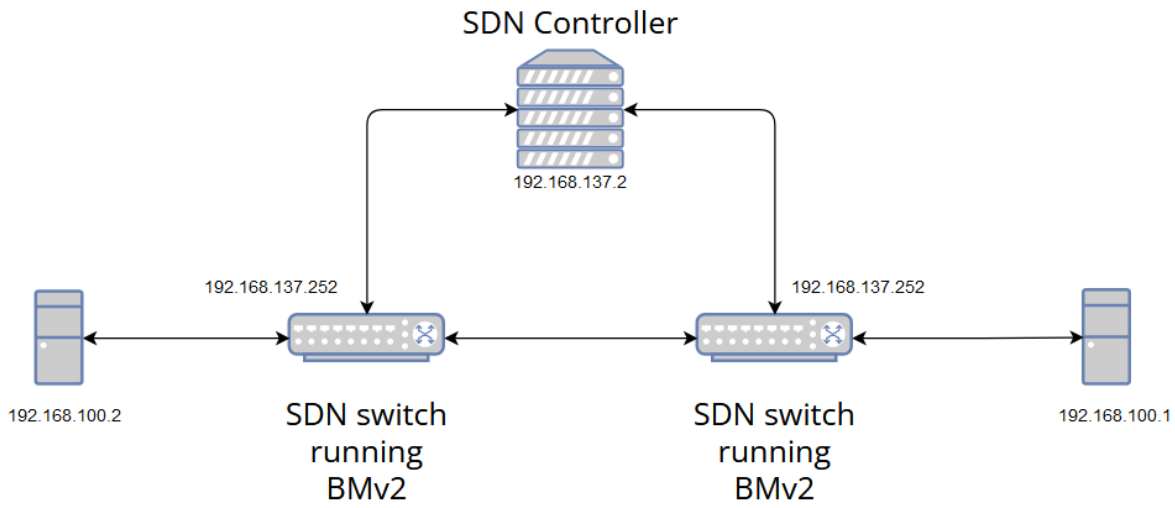


Figure 51: Diagram that depicts the SDN perspective of the setup from the first phase of the experiments with BMv2

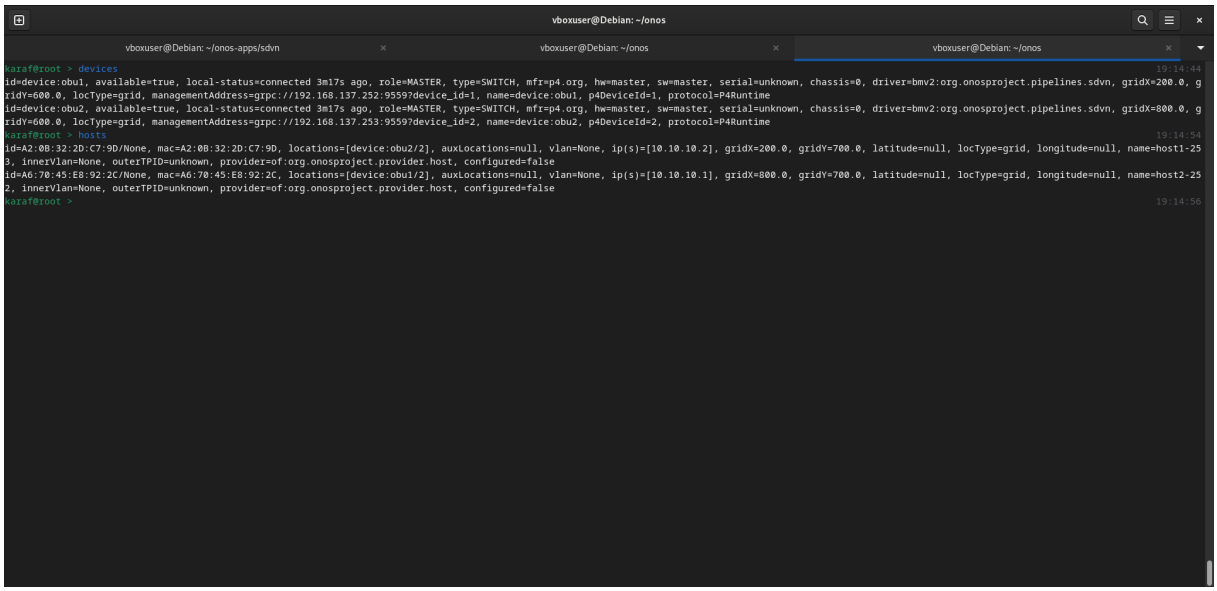


Figure 52: ONOS console with information relating to the devices and hosts of the network

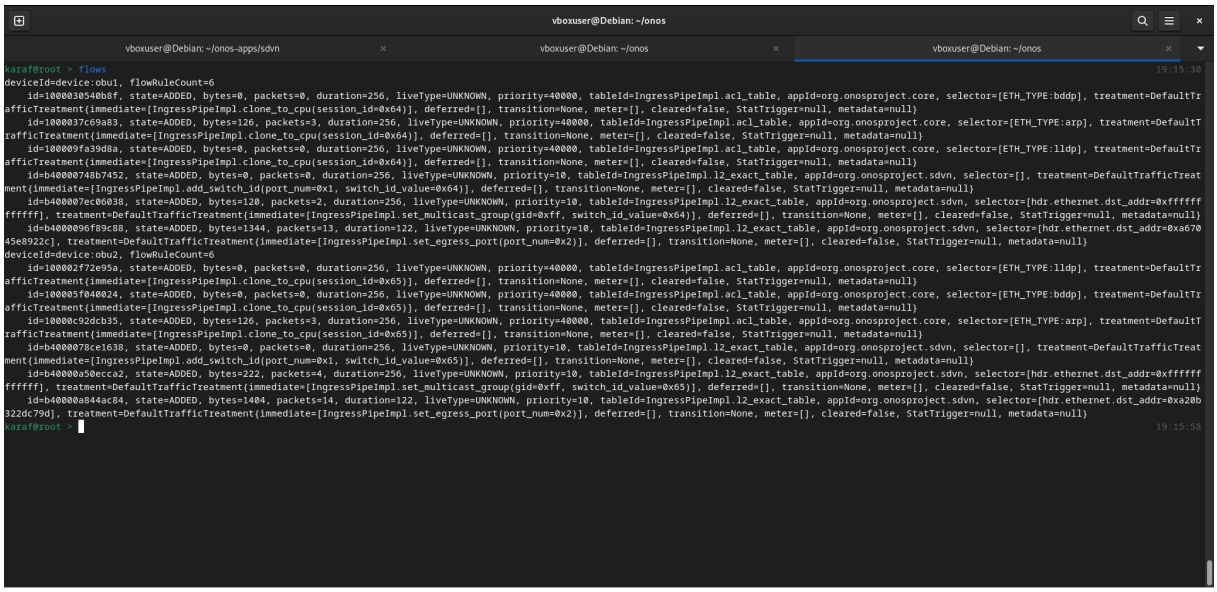


Figure 53: ONOS console with information relating to the flows of the network

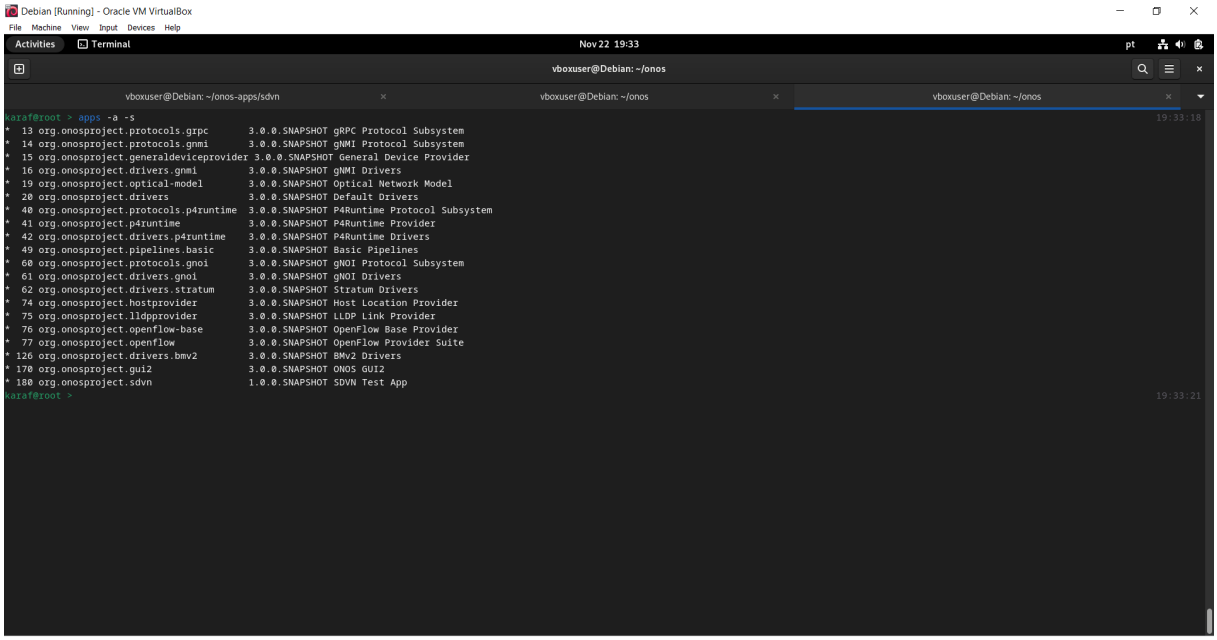


Figure 54: ONOS console with information relating to the applications activated on the controller

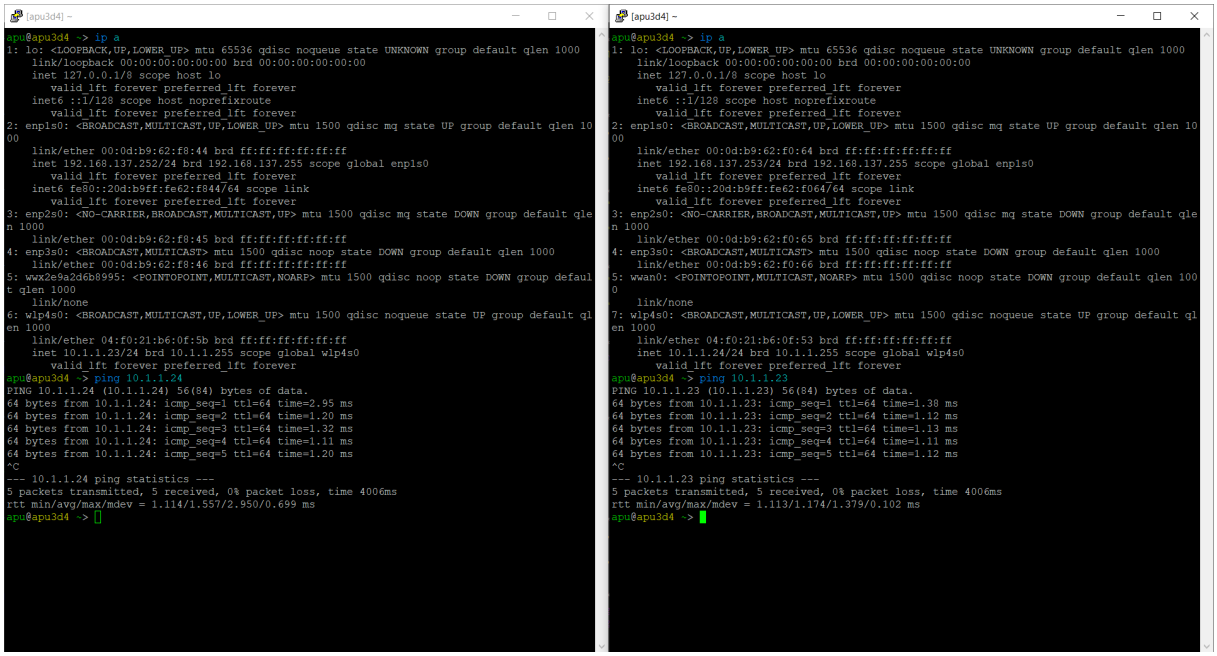


Figure 55: Results from running the commands "ip a" and "ping" between the two devices on the network using the IP address of the wireless interfaces.

8.3.2 Phase 2

Phase two of the experiments with BMv2 represents a significant advancement over the initial phase, as it introduces the crucial element of wireless communication. Accordingly, the ath9k patch was implemented on the devices to enable communication at the 802.11p standard. This can be seen in Figure55.

As a result of this, two changes had to be made. The first of these amendments required the activation of the BMv2 with the wireless interface. The second consequence required modification of the P4/ONOS applications and configuration to accommodate the distinctive characteristics of the wireless antennas.

The test configuration is illustrated in Figure56, which was also used as the basis for the diagram seen in Figure57 and described in the following itemized list:

- 192.168.137.1 - This IP address identifies the author's personal laptop computer. It serves as the central coordinating hub for the entire network, facilitating the observation and monitoring of all network activity.

- 192.168.137.2 - This address represents a virtual machine that is hosted on the author's laptop via the Oracle VM VirtualBox virtualization platform. The virtual environment is configured with Debian 12 and is primarily intended for the deployment of ONOS. This configuration is essential due to the fact that the primary branch of ONOS is designed to run on Linux, whereas the author's device runs on Windows.

- 192.168.137.252 - This device corresponds to the version 2 of the devices. The device has a fresh installation of the Ubuntu Server operating system and is running BMv2 with grpc. Furthermore,

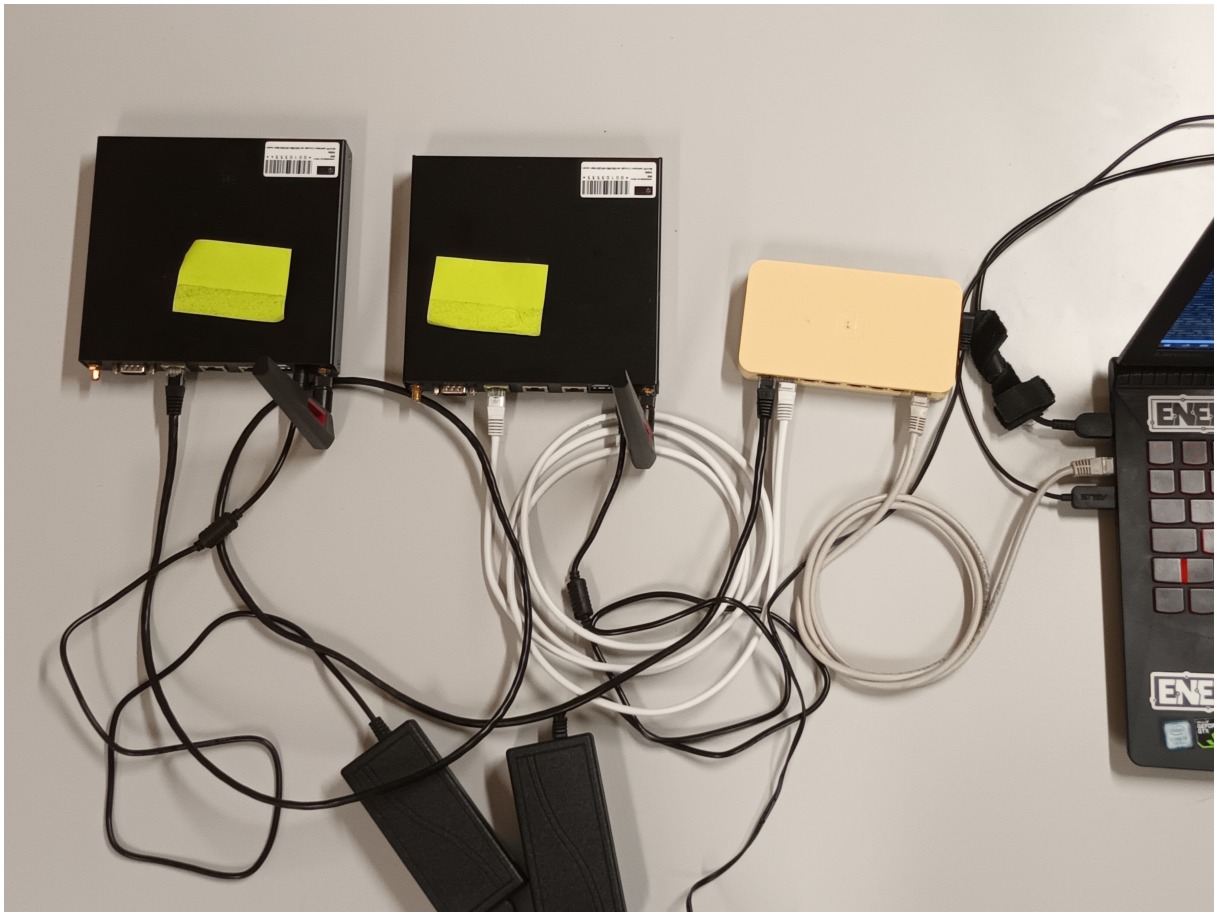


Figure 56: Setup from the second phase of the experiments with BMv2

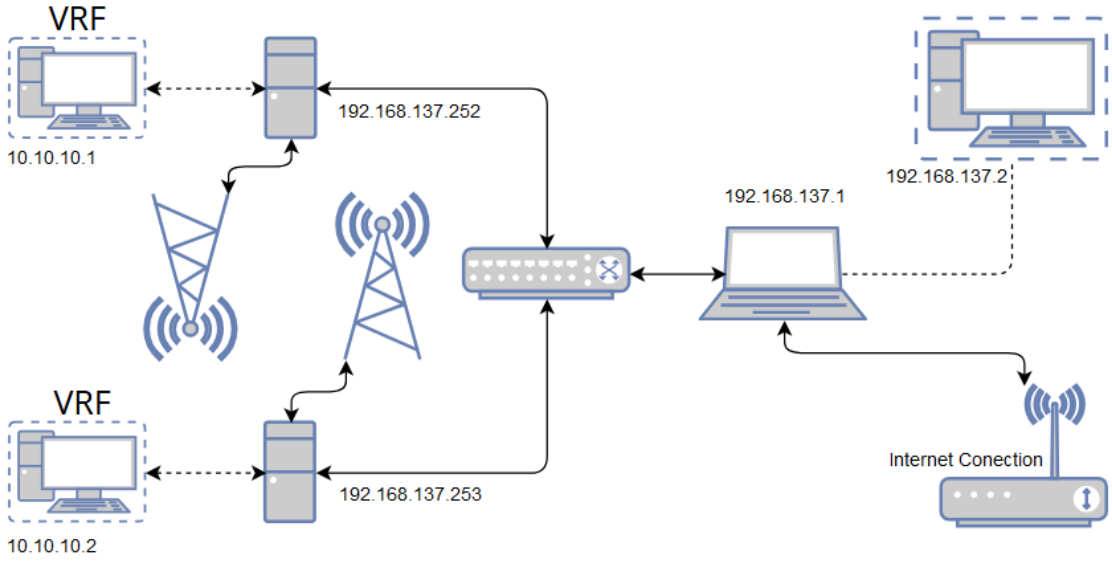


Figure 57: Diagram that represents the setup from the second phase of the experiments with BMv2

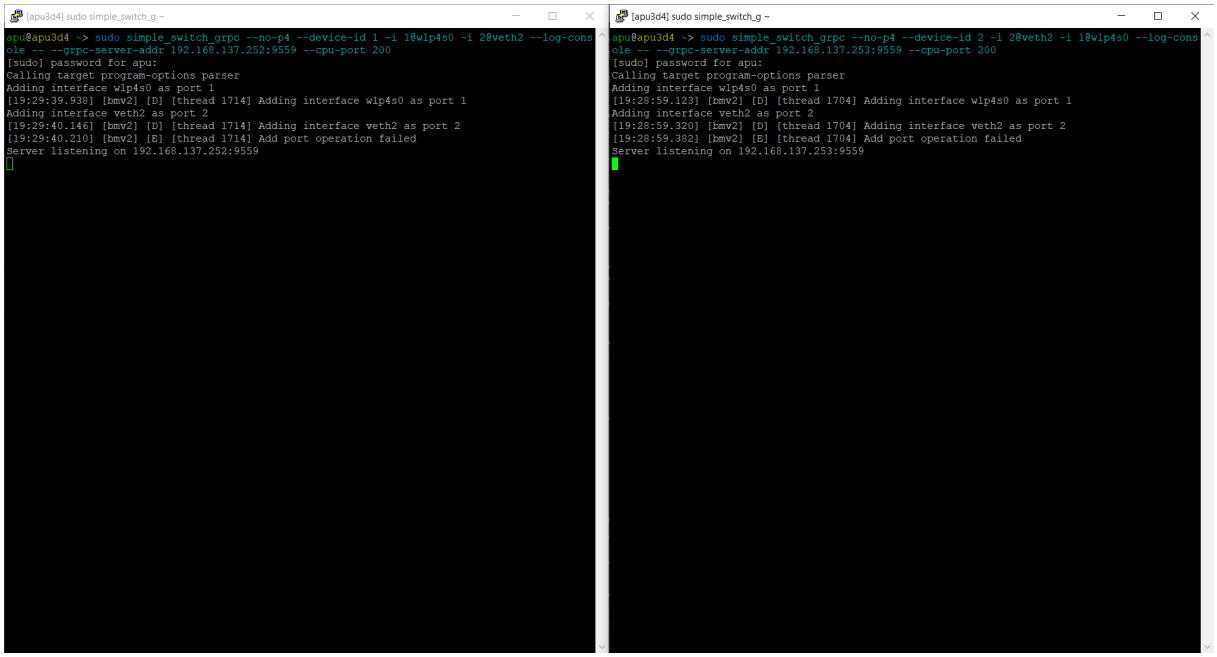


Figure 58: Consoles running BMv2 on both devices

the device is equipped with an antenna capable of communicating at frequencies up to 6 GHz and has been applied with the ath9k patch to the WiFi driver, thereby enabling communication using 802.11p.

- 192.168.137.253 - This device is virtually identical to the device 192.168.137.252.
- 10.10.10.1 - This IP address identifies the virtual interface that has been added to the VRF instance created in the 192.168.137.252 device.
- 10.10.10.2 - Identically to the last one, this IP address identifies the virtual interface that has been added to the VRF instance created in the 192.168.137.253 device.

8.3.2.1 Results

Figure58 shows BMv2 running in both devices.

Figure59 illustrates the connectivity between devices 192.168.137.252 and 192.168.137.253, manifesting through the VFRs with IPs 10.10.10.1 and 10.10.10.2.

In order for communication to be established between the devices, it was necessary to make modifications to the program created in the first phase. For any message to be successfully received by an antenna, that antenna's destination Media Access Control address must either be the address of the antenna or the broadcast address. It was thus decided to pursue the latter course of action, which entailed changing the addresses of the messages sent via the wireless interface to the broadcast address. The original address is stored within the packet and inserted when required.

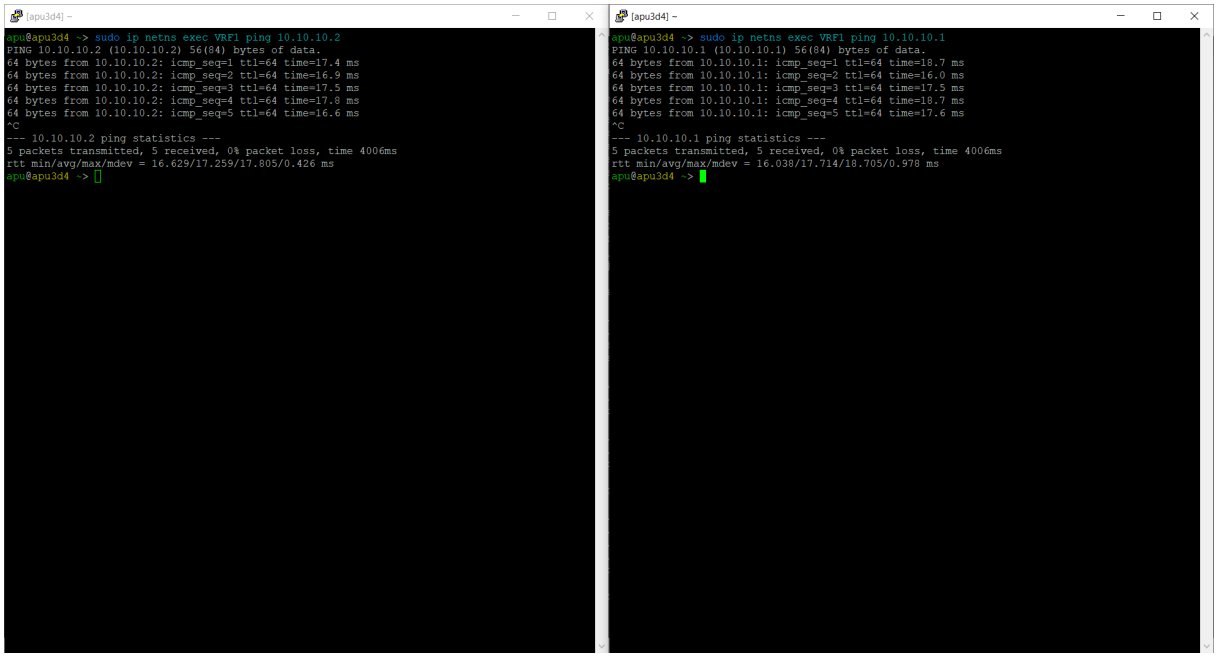


Figure 59: Results from running the commands "ping" in the VRF of the two devices on the network

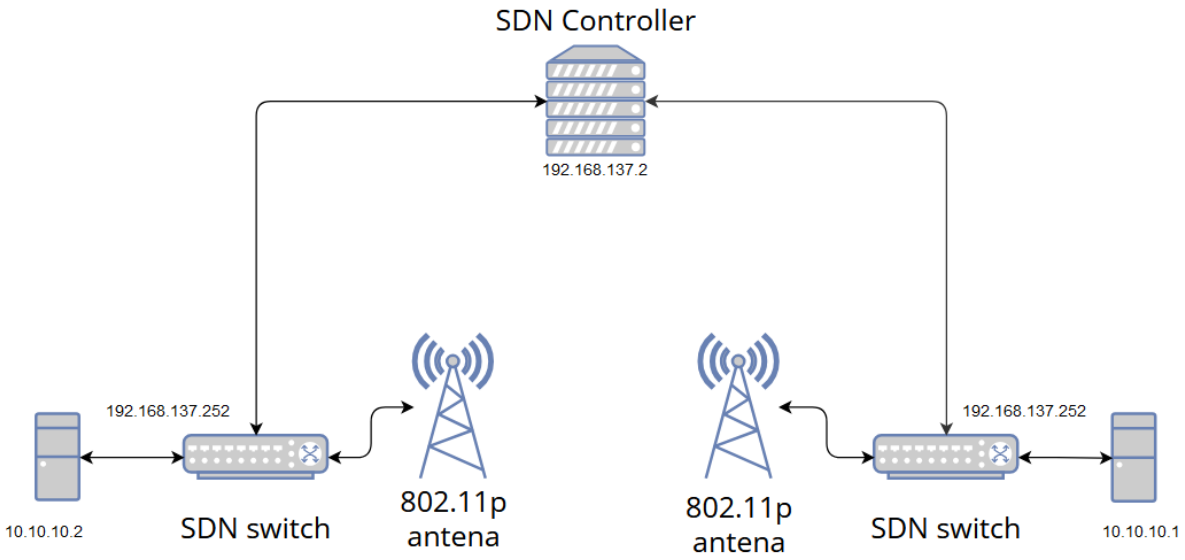


Figure 60: Diagram that depicts the SDN perspective of the setup from the second phase of the experiments with BMv2

When viewed through the lens of SDN, the physical configuration can be conceptualized as illustrated in Figure60.

The list of devices and hosts known to the controller is shown in Figure61, and the flows that have been injected into the devices are presented in Figure62.

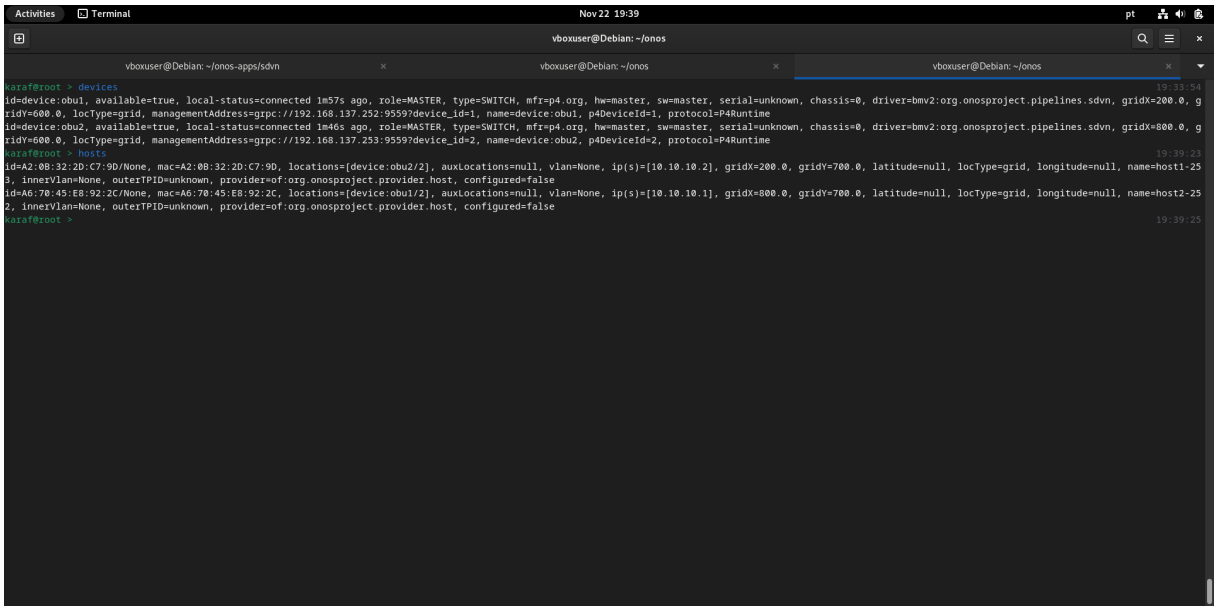


Figure 61: ONOS console with information relating to the devices and hosts of the network

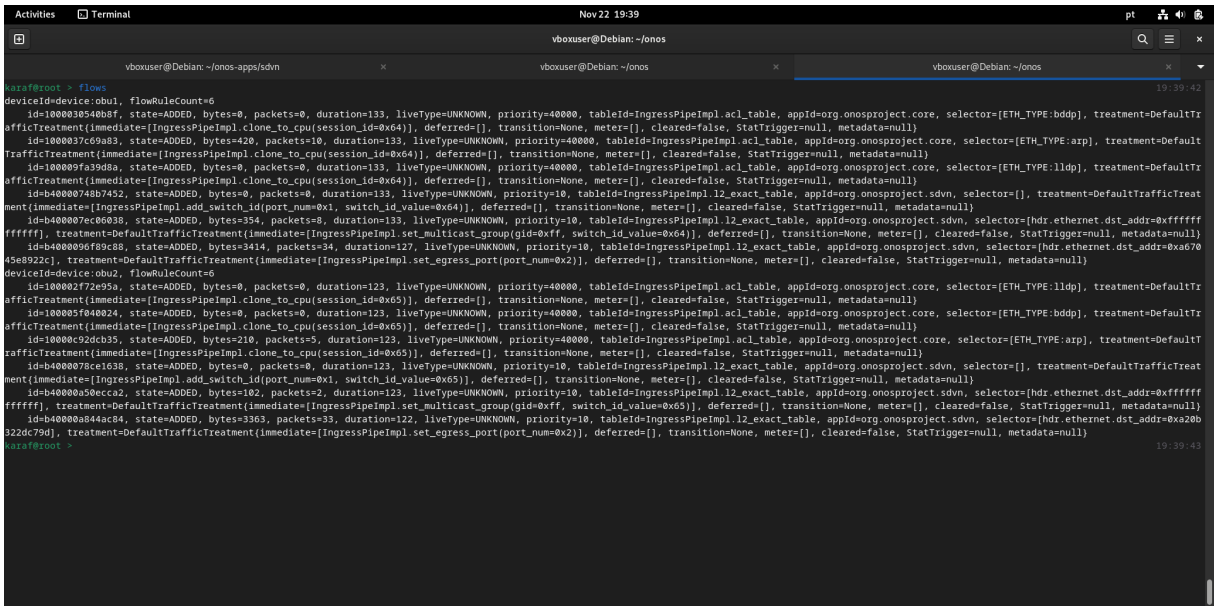


Figure 62: ONOS console with information relating to the flows of the network

8.3.3 Limitations

This experiment was constrained by a significant limitation, namely the availability of only two devices with BMv2 installed for use. As previously stated, the original device provided by the university is not suitable for use in this scenario due to insufficient storage capacity to install BMv2. Furthermore, financial constraints precluded the acquisition of another device of the newer version. Obtaining more than two devices was crucial to effectively recreate more intricate testing scenarios that more closely resemble real-world contexts, such as the transmission of a message through a network of wireless devices. However, despite the inability to demonstrate this specific scenario due to a lack of resources, the observed functionality of the program, its code, and the logs from BMv2 and ONOS provide strong indications that the code will perform as intended in the aforementioned scenario. Nevertheless, tests of this nature are regrettably not feasible at this time.

8.3.4 Analysis and Interpretation

In Phase 1 of these tests, it was an unanticipated finding that the default southbound API of BMv2 is not P4Runtime, as had been previously assumed, but rather Thrift. The Thrift API[91] is a component of the Apache Thrift framework, which originated at Facebook. The goal of Thrift is to facilitate cross-language communication and data serialization by providing a language-agnostic interface that can be implemented in a variety of programming languages. In this case, Thrift is used to turn an easy to read file with controller instructions to device specific instruction in the data plane.

This decision by the developers may initially seem incongruous, but it is, in fact, a logical consequence of the broader strategic context. BMv2 is not a production-grade switch; rather, it is a software tool designed for educational and experimental purposes in conjunction with P4. Thrift offers a more straightforward mechanism that streamlines the process of inserting rules on network devices. In consideration of this fact, it is clear that BMv2 was not developed with high performance as a primary objective. Accordingly, any practical development should seek to implement a faster solution in the event that performance issues arise.

The transposition of the final scenario to an SDN network, which was illustrated in Figure60, to an ITS vehicle subsystem has resulted in the diagram presented in Figure63. From this diagram, a number of interesting observations and comparisons can be made. In this context, the SDN switch can be compared to an ITS-S gateway and an ITS-S router. It is clear that any comparison should be viewed with caution, as the device under consideration does not align perfectly with any of the aforementioned categories. This comparison is intended to illustrate the device's functionality, rather than to provide a definitive categorization. It is inaccurate to categorize this device as an OSI stack to ETSI protocol translator, as it lacks the requisite functionality to perform such a translation. The SDN controller could be integrated into the vehicle ITS station as an ITS application in a manner that would align with the standards and classifications set forth by the ETSI.

As anticipated, the BMv2 proved to be the optimal choice. This experiment was successful in enabling BMv2 to function with an ONOS controller and the ath9k patch. The development of this test demonstrates

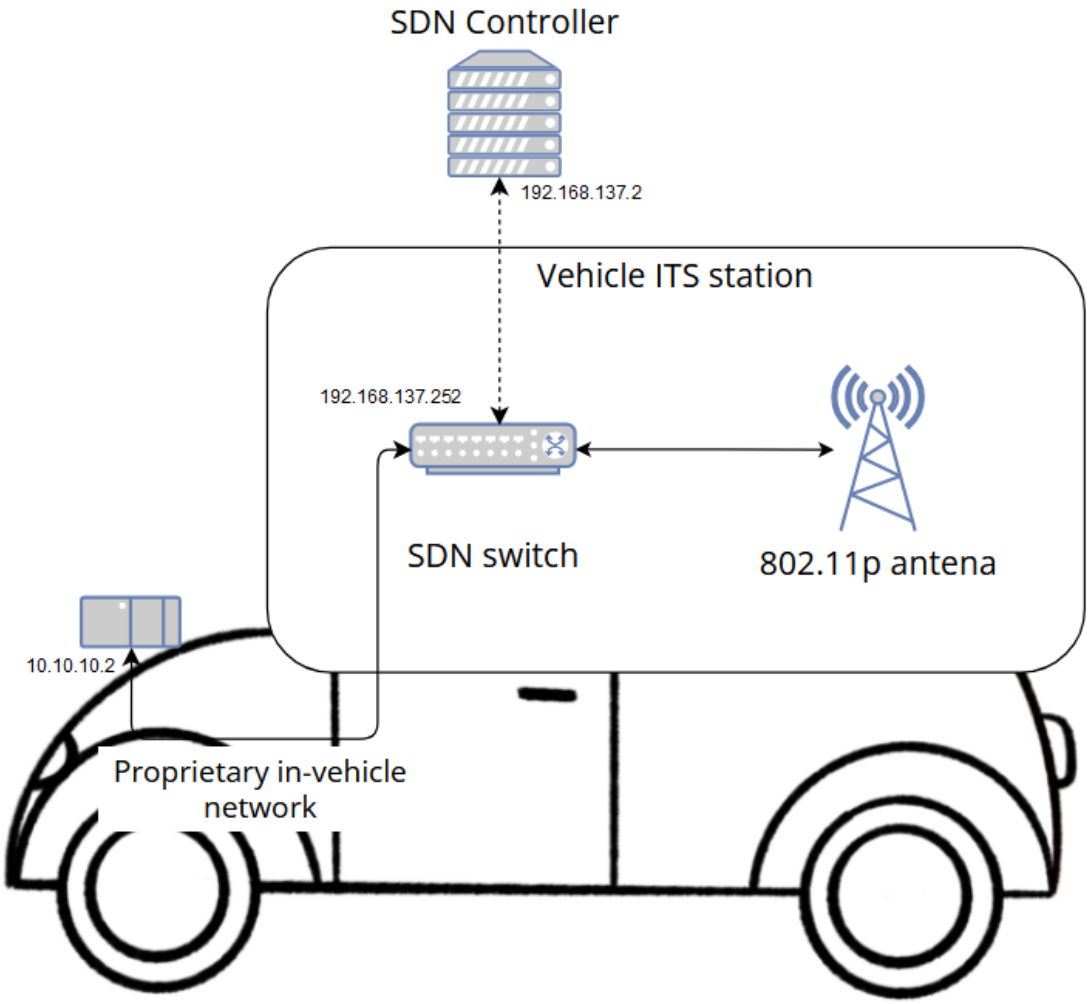


Figure 63: Diagram representing the SDN network as an ITS vehicle subsystem

that P4 is capable of functioning with 802.11p, which indicates that this technology can be utilized for vehicular deployment. The power and control provided by P4 affords users a vast array of potential applications. With it and according to our tests it is possible to create a P4 based network that complies with ETSI norms, making it possible for this technology to be used in the real world.

9 Conclusion and Future work

9.1 Conclusion

In this thesis, it was this author's intention to develop an OBU in accordance with the principles of SDN, thereby aligning this thesis with the domain of SDVN. Throughout the course of this project, a critical examination of the actual implications of this goal was undertaken. The findings of the research revealed that the objective of making an OBU SDN compatible is, in essence, equivalent to developing an SDN ITS vehicle subsystem. Moreover, the insights gained from the research enabled the presentation of a feasible and realistic trajectory for SDVN. This vision is grounded in the recognition that, for SDVN to be deployed in the future, interoperability with traditional devices is of paramount importance. In essence, the success of SDVN will be contingent upon its capacity to integrate with existing infrastructure. In the practical implementation stage, a series of tests were conducted on the most relevant software for data plane emulation in the field of SDN in order to ascertain the most appropriate software for vehicular scenarios. In accordance with the requirements of this thesis, the queries pertaining to the control plane were set aside in order to prioritize the data plane, thereby facilitating the recreation of the OBU. Prototype devices were configured and the BMv2 and OvS software were installed on them. In the absence of any notable requirements, ONOS was chosen as the control plane technology. Subsequently, the ath9k patch was implemented with the objective of evaluating the compatibility of communication utilizing the intended 802.11p protocol. The results of the tests revealed that OvS was incompatible with wireless interfaces, whereas BMv2 demonstrated no such incompatibility. The tests conducted on OvS and BMv2 served to reinforce the hypothesis that P4 is the most optimal data plane technology. The P4 language, in conjunction with the controller, was employed to modify the subsequent layers, thereby establishing communication in 802.11p between two devices. It was thus demonstrated that BMv2 represents the optimal software for utilization in SDVN. The present thesis has been completed in accordance with the stipulated criteria and objectives. All of the proposed goals have been achieved, thereby providing a crucial step in establishing a foundation for the implementation of a complete SDVN testing scenario based on real hardware. Ultimately, the research, work, and subsequent findings presented in this thesis provide sufficient evidence to demonstrate that the integration of SDN principles in the field of VANETs is a viable and achievable proposition.

9.2 Future work

The natural progression of this work is to adapt the remaining ETSI subsystems to be SDN-compatible. This process should begin with the Central and Roadside subsystems, as they represent the most critical components in this context. Upon completion of this phase, the foundation will be established for a comprehensive, standards-compliant SDN system implementation. This second phase would entail the development of the requisite controller and data plane logic to achieve intercommunication between SDN and non-SDN devices in conformance with established standards. The most desirable long-term objective would be the recreation of a SDN system that is an exact functional replica of a traditional device. Once this has been achieved, the technology can be implemented and evaluated in conjunction with actual road networks. It would, however, be a mistake to view the complete system as the final stage of research. Rather, it should be regarded as the foundation for a new line of enquiry. The potential of P4 can be harnessed to facilitate the testing of a multitude of aspects pertaining to the ETSI protocols. The first and most crucial task, which is not directly related to the ETSI protocols, is to identify the optimal controller placement. This, along with related inquiries such as the evaluation of election algorithms for the controller in clusters of vehicles, would represent a vital priority for ensuring the success and advancement of this technology. Finally, this thesis also lays the groundwork for the redesign and upgrade of the device used on the OBU. The patch currently in use to achieve 802.11p functionality is not a long-term solution. Therefore, it would be prudent to explore a more permanent solution, such as SDR.

Bibliography

[1] J. Jakubiak and Y. Koucheryavy. “State of the Art and Research Challenges for VANETs”. In: *2008 5th IEEE Consumer Communications and Networking Conference*. 2008 5th IEEE Consumer Communications and Networking Conference. ISSN: 2331-9860. 2008-01, pp. 912–916. doi: 10.1109/ccnc08.2007.212 (cit. on pp. 5, 7, 16, 27, 28).

[2] S. Al-Sultan et al. “A comprehensive survey on vehicular Ad Hoc network”. In: *Journal of Network and Computer Applications* 37 (2014-01-01), pp. 380–392. issn: 1084-8045. doi: 10.1016/j.jnca.2013.02.036. url: https://www.sciencedirect.com/science/article/pii/S108480451300074X (visited on 2022-11-02) (cit. on pp. 5, 7, 14, 15, 22, 23, 27, 28).

[3] W. Liang et al. “Vehicular Ad Hoc Networks: Architectures, Research Issues, Methodologies, Challenges, and Trends”. In: *International Journal of Distributed Sensor Networks* 11.8 (2015-08-01), p. 745303. issn: 1550-1477. doi: 10.1155/2015/745303. url: http://journals.sagepub.com/doi/10.1155/2015/745303 (visited on 2023-02-22) (cit. on pp. 5, 7, 20, 24, 27).

[4] H. Dinh Thai et al. “Applications of Repeated Games in Wireless Networks: A Survey”. In: *IEEE Communications Surveys & Tutorials* 17 (2015-01-11). doi: 10.1109/COMST.2015.2445789 (cit. on p. 6).

[5] *Platform: C-Roads*. url: https://www.c-roads.eu/platform.html (visited on 2024-11-25) (cit. on p. 6).

[6] Y. Toor et al. “Vehicle Ad Hoc networks: applications and related technical issues”. In: *IEEE Communications Surveys & Tutorials* 10.3 (2008). Conference Name: IEEE Communications Surveys & Tutorials, pp. 74–88. issn: 1553-877X. doi: 10.1109/COMST.2008.4625806 (cit. on pp. 6–8, 16, 17, 20, 21, 26, 28).

[7] S. M. Corson and J. P. Macker. *Mobile Ad hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations*. Request for Comments RFC 2501. Num Pages: 12. Internet Engineering Task Force, 1999-01. doi: 10.17487/RFC2501. url: https://datatracker.ietf.org/doc/rfc2501 (visited on 2023-09-28) (cit. on p. 8).

[8] C2C-CC. *CAR 2 CAR Communication Consortium Manifesto - Overview of the C2C-CC System*. 2007-08-28. url: https://www.car-2-car.org/fileadmin/documents/General_Documents/C2C-CC_Manifesto_Aug_2007.pdf (visited on 2023-04-23) (cit. on p. 8).

[9] ETSI. *Intelligent Transport Systems (ITS); Communications Architecture*. 2010-09. url: https://www.etsi.org/deliver/etsi_en/302600_302699/302665/01.01.01_60/en_302665v010101p.pdf (visited on 2023-04-27) (cit. on pp. 9–15, 24, 26).

[10] ETSI. *Intelligent Transport Systems (ITS); ITS-G5 Access layer specification for Intelligent Transport Systems operating in the 5 GHz frequency band*. 2020-01. url: https://www.etsi.org/deliver/etsi_en/302600_302699/302663/01.03.01_60/en_302663v010301p.pdf (visited on 2023-11-08) (cit. on p. 15).

[11] M. S. Anwer and C. Guy. "A Survey of VANET Technologies". In: *Journal of Emerging Trends in Computing and Information Sciences* (2014) (cit. on pp. 15, 19).

[12] *History of IEEE*. url: https://www.ieee.org/about/ieee-history.html (visited on 2023-11-14) (cit. on p. 16).

[13] R.
bibinitperiod Schwarz. "Intelligent Transportation Systems Using IEEE 802.11p". In: (2019-02-14) (cit. on p. 16).

[14] D. Jiang and L. Delgrossi. "IEEE 802.11p: Towards an International Standard for Wireless Access in Vehicular Environments". In: IEEE Vehicular Technology Conference. 2008-06-14, pp. 2036–2040. doi: 10.1109/VETECS.2008.458 (cit. on pp. 16, 17, 25).

[15] A. Festag. "Cooperative intelligent transport systems standards in europe". In: *IEEE Communications Magazine* 52.12 (2014-12). Conference Name: IEEE Communications Magazine, pp. 166–172. issn: 1558-1896. doi: 10.1109/MCOM.2014.6979970 (cit. on pp. 17, 18, 21–24).

[16] N. Asselin-Miller et al. "Study on the Deployment of C-ITS in Europe: Final Report". In: 1 (2016-05-02) (cit. on pp. 17, 18, 53, 110).

[17] J. Härri and J. Kenney. "Multi-Channel Operations, Coexistence and Spectrum Sharing for Vehicular Communications". In: ed. by C. Campolo, A. Molinaro, and R. Scopigno. Book Title: Vehicular ad hoc Networks. Cham: Springer International Publishing, 2015, pp. 193–218. isbn: 978-3-319-15496-1 978-3-319-15497-8. doi: 10.1007/978-3-319-15497-8_7. url: https://link.springer.com/10.1007/978-3-319-15497-8_7 (visited on 2023-10-05) (cit. on pp. 17–19).

[18] ETSI. *Electromagnetic compatibility and Radio spectrum Matters (ERM); Intelligent Transport Systems (ITS); Part 1: Technical characteristics for pan-European harmonized communications equipment operating in the 5 GHz frequency range and intended for critical road-safety applications; System Reference Document*. 2005-06. url: https://www.etsi.org/deliver/etsi_tr/1

02400_102499/10249201/01.01.01_60/tr_10249201v010101p.pdf (visited on 2023-11-17) (cit. on p. 17).

[19] Ş. ŞORIGA. "ITS-G5 AND MOBILE WIMAX PERFORMANCE IN VEHICLE-TO-INFRASTRUCTURE COMMUNICATIONS". In: *ISSN 1454-234x* UPB Scientific Bulletin, Series C: Electrical Engineering. (2012). url: https://www.scientificbulletin.upb.ro/rev_docs_arhiva/full4d5_755707.pdf (visited on 2023-10-16) (cit. on p. 18).

[20] "IEEE Standard for Information Technology–Telecommunications and Information Exchange between Systems Local and Metropolitan Area Networks–Specific Requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 5: Enhancements for Next Generation V2X". In: *IEEE Std 802.11bd-2022 (Amendment to IEEE Std 802.11-2020 as amended by IEEE Std 802.11ax-2021, IEEE Std 802.11ay-2021, IEEE Std 802.11ba-2021, IEEE Std 802.11-2020/Cor 1-2022, and IEEE Std 802.11az-2022)* (2023-03). Conference Name: IEEE Std 802.11bd-2022 (Amendment to IEEE Std 802.11-2020 as amended by IEEE Std 802.11ax-2021, IEEE Std 802.11ay-2021, IEEE Std 802.11ba-2021, IEEE Std 802.11-2020/Cor 1-2022, and IEEE Std 802.11az-2022), pp. 1–144. doi: 10.1109/IEEESTD.2023.10063942. url: https://ieeexplore.ieee.org/document/10063942 (visited on 2023-10-25) (cit. on p. 19).

[21] *3GPP – The Mobile Broadband Standard*. 3GPP. url: https://www.3gpp.org/ (visited on 2023-11-16) (cit. on p. 19).

[22] S. Gyawali et al. "Challenges and Solutions for Cellular Based V2X Communications". In: *IEEE Communications Surveys & Tutorials* 23.1 (2021), pp. 222–255. issn: 1553-877X, 2373-745X. doi: 10.1109/COMST.2020.3029723. url: https://ieeexplore.ieee.org/document/9217500/ (visited on 2023-10-16) (cit. on pp. 19, 20).

[23] *5GAA*. 5GAA. url: https://5gaa.org/ (visited on 2023-11-16) (cit. on p. 19).

[24] R. Weber, J. Misener, and V. Park. "C-V2X - A Communication Technology for Cooperative, Connected and Automated Mobility". In: *Mobile Communication - Technologies and Applications; 24. ITG-Symposium*. Mobile Communication - Technologies and Applications; 24. ITG-Symposium. 2019-05, pp. 1–6. url: https://ieeexplore.ieee.org/abstract/document/8731783 (visited on 2023-10-23) (cit. on p. 20).

[25] ETSI. *Intelligent Transport Systems (ITS); Vehicular Communications; GeoNetworking; Part 1: Requirements*. 2014-04. url: https://www.etsi.org/deliver/etsi_en/302600_302699/30263601/01.02.01_60/en_30263601v010201p.pdf (visited on 2023-11-27) (cit. on p. 21).

[26] ETSI. *Intelligent Transport Systems (ITS); Vehicular Communications; GeoNetworking; Part 2: Scenarios*. 2013-11. url: https://www.etsi.org/deliver/etsi_en/302600_302699/30263602/01.02.01_60/en_30263602v010201p.pdf (visited on 2023-11-27) (cit. on p. 21).

[27] ETSI. *Intelligent Transport Systems (ITS); Vehicular Communications; GeoNetworking; Part 3: Network Architecture*. 2014-12. url: https://www.etsi.org/deliver/etsi_en/302600_302699/30263603/01.02.01_60/en_30263603v010201p.pdf (visited on 2023-10-06) (cit. on pp. 21, 23, 26).

[28] ETSI. *Intelligent Transport Systems (ITS); Vehicular Communications; GeoNetworking; Part 4: Geographical addressing and forwarding for point-to-point and point-to-multipoint communications; Sub-part 1: Media-Independent Functionality*. 2020-01. url: https://www.etsi.org/deliver/etsi_en/302600_302699/3026360401/01.04.01_60/en_3026360401v010401p.pdf (visited on 2023-11-27) (cit. on p. 21).

[29] ETSI. *Intelligent Transport Systems (ITS); Vehicular Communications; GeoNetworking; Part 5: Transport Protocols; Sub-part 1: Basic Transport Protocol*. 2019-05. url: https://www.etsi.org/deliver/etsi_en/302600_302699/3026360501/02.02.01_60/en_3026360501v020201p.pdf (visited on 2023-11-27) (cit. on p. 21).

[30] ETSI. *Intelligent Transport Systems (ITS); Vehicular Communications; GeoNetworking; Part 6: Internet Integration; Sub-part 1: Transmission of IPv6 Packets over GeoNetworking Protocols*. 2014-05. url: https://www.etsi.org/deliver/etsi_en/302600_302699/3026360601/01.02.01_60/en_3026360601v010201p.pdf (visited on 2023-11-27) (cit. on p. 21).

[31] A. Festag. "Standards for vehicular communication—from IEEE 802.11p to 5G". In: *e & i Elektrotechnik und Informationstechnik* 132.7 (2015-11-01), pp. 409–416. issn: 1613-7620. doi: 10.1007/s00502-015-0343-0. url: https://doi.org/10.1007/s00502-015-0343-0 (visited on 2023-10-16) (cit. on pp. 21, 22).

[32] ETSI. *Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 2: Specification of Cooperative Awareness Basic Service*. 2019-04. url: https://www.etsi.org/deliver/etsi_en/302600_302699/30263702/01.04.01_60/en_30263702v010401p.pdf (visited on 2023-11-27) (cit. on p. 22).

[33] ETSI. *Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 3: Specifications of Decentralized Environmental Notification Basic Service*. 2019-04. url: https://www.etsi.org/deliver/etsi_en/302600_302699/30263703/01.03.01_60/en_30263703v010301p.pdf (visited on 2023-11-27) (cit. on p. 23).

[34] A. K. Malhi, S. Batra, and H. S. Pannu. "Security of vehicular ad-hoc networks: A comprehensive survey". In: *Computers & Security* 89 (2020-02), p. 101664. issn: 01674048. doi: 10.1016/j.cose.2019.101664. url: https://linkinghub.elsevier.com/retrieve/pii/S0167404818312872 (visited on 2023-11-28) (cit. on pp. 24, 25).

[35] H. Hasrouny et al. "VANet security challenges and solutions: A survey". In: *Vehicular Communications* 7 (2017-01-01), pp. 7–20. issn: 2214-2096. doi: 10.1016/j.vehcom.2017.01.002. url: https://www.sciencedirect.com/science/article/pii/S2214209616301231 (visited on 2023-12-15) (cit. on p. 25).

[36] ETSI. *Intelligent Transport Systems (ITS); Security; Security Services and Architecture*. 2010-09. url: https://www.etsi.org/deliver/etsi_ts/102700_102799/102731/01.01.01_60/ts_102731v010101p.pdf (visited on 2023-11-28) (cit. on p. 26).

[37] ETSI. *Intelligent Transport Systems (ITS); Security; ITS communications security architecture and security management*. 2018-04. url: https://www.etsi.org/deliver/etsi_ts/102900_102999/102940/01.03.01_60/ts_102940v010301p.pdf (visited on 2023-12-05) (cit. on p. 27).

[38] ETSI. *Intelligent Transport Systems (ITS); Security; ITS communications security architecture and security management; Release 2*. 2021-07. url: https://www.etsi.org/deliver/etsi_ts/102900_102999/102940/02.01.01_60/ts_102940v020101p.pdf (visited on 2023-11-28) (cit. on p. 26).

[39] M. J. N. Mahi et al. "A Review on VANET Research: Perspective of Recent Emerging Technologies". In: *IEEE Access* 10 (2022). Conference Name: IEEE Access, pp. 65760–65783. issn: 2169-3536. doi: 10.1109/ACCESS.2022.3183605. url: https://ieeexplore.ieee.org/abstract/document/9797696 (visited on 2023-12-19) (cit. on p. 28).

[40] M. Lu et al. "Pan-European deployment of C-ITS: the way forward". In: (2019) (cit. on pp. 28, 29).

[41] B. A. A. Nunes et al. "A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks". In: *IEEE Communications Surveys & Tutorials* 16.3 (2014). Conference Name: IEEE Communications Surveys & Tutorials, pp. 1617–1634. issn: 1553-877X. doi: 10.1109/SURV.2014.012214.00180 (cit. on pp. 30, 31, 33–36, 45–47).

[42] N. Feamster, J. Rexford, and E. Zegura. "The road to sdn". In: *Queue* 11 (2013-12). doi: 10.1145/2559899.2560327 (cit. on pp. 30–35).

[43] W. Xia et al. "A Survey on Software-Defined Networking". In: *IEEE Communications Surveys & Tutorials* 17.1 (2015). Conference Name: IEEE Communications Surveys & Tutorials, pp. 27–51. issn: 1553-877X. doi: 10.1109/COMST.2014.2330903 (cit. on pp. 30, 33–37, 40, 45–47, 49).

[44] *Open Networking Foundation*. en-US. url: https://opennetworking.org/ (visited on 2024-01-12) (cit. on p. 30).

[45] L. L. Peterson et al. *Software-Defined Networks: A Systems Approach*. English. Wroclaw: Systems Approach LLC, 2021-01. isbn: 978-1-73647-210-1 (cit. on pp. 30–32, 35–38, 40–49).

[46] R. Bifulco and G. Retvari. "A Survey on the Programmable Data Plane: Abstractions, Architectures, and Open Problems". en. In: *2018 IEEE 19th International Conference on High Performance Switching and Routing (HPSR)*. Bucharest, Romania: IEEE, 2018-06, pp. 1–7. isbn: 978-1-5386-7801-5. doi: 10.1109/HPSR.2018.8850761. url: https://ieeexplore.ieee.org/document/8850761/ (visited on 2023-12-22) (cit. on pp. 30, 31, 33, 35–38, 40, 42, 47).

[47] A. S. Thyagaturu et al. “Software Defined Optical Networks (SDONs): A Comprehensive Survey”. In: *IEEE Communications Surveys & Tutorials* 18.4 (2016). Conference Name: IEEE Communications Surveys & Tutorials, pp. 2738–2786. issn: 1553-877X. doi: 10.1109/COMST.2016.2586999 (cit. on pp. 31, 32, 34, 36, 44, 46, 47).

[48] A. Alowa. “Scalable Reliable Controller Placement in Software Defined Networking”. In: (2020-09). url: https://core.ac.uk/reader/355870830 (visited on 2024-03-19) (cit. on p. 32).

[49] A. Liatifis et al. “Advancing SDN from OpenFlow to P4: A Survey”. In: *ACM Computing Surveys* 55.9 (2023-01), 186:1–186:37. issn: 0360-0300. doi: 10.1145/3556973. url: https://dl.acm.org/doi/10.1145/3556973 (visited on 2024-01-11) (cit. on pp. 32, 38, 49).

[50] R. C. Sofia and J. Soldatos. “Shaping the Future of IoT with Edge Intelligence; How Edge Computing Enables the Next Generation of IoT Applications”. en. In: (2024) (cit. on p. 32).

[51] D. Kreutz et al. “Software-Defined Networking: A Comprehensive Survey”. In: *Proceedings of the IEEE* 103.1 (2015-01). Conference Name: Proceedings of the IEEE, pp. 14–76. issn: 1558-2256. doi: 10.1109/JPROC.2014.2371999 (cit. on pp. 32–37, 39, 42, 44–49).

[52] N. McKeown et al. “OpenFlow: enabling innovation in campus networks”. In: *ACM SIGCOMM Computer Communication Review* 38.2 (2008-03), pp. 69–74. issn: 0146-4833. doi: 10.1145/1355734.1355746. url: https://doi.org/10.1145/1355734.1355746 (visited on 2023-02-10) (cit. on pp. 33, 37, 43, 45).

[53] S. Li et al. “Protocol Oblivious Forwarding (POF): Software-Defined Networking with Enhanced Programmability”. In: *IEEE Network* 31.2 (2017-03). Conference Name: IEEE Network, pp. 58–66. issn: 1558-156X. doi: 10.1109/MNET.2017.1600030NM (cit. on pp. 36, 39).

[54] Z. Latif et al. “A comprehensive survey of interface protocols for software defined networks”. en. In: *Journal of Network and Computer Applications* 156 (2020-04), p. 102563. issn: 1084-8045. doi: 10.1016/j.jnca.2020.102563. url: https://www.sciencedirect.com/science/article/pii/S1084804520300370 (visited on 2022-11-07) (cit. on pp. 37, 44, 46–49).

[55] F. Hauser et al. “A survey on data plane programming with P4: Fundamentals, advances, and applied research”. In: *Journal of Network and Computer Applications* 212 (2021-08), p. 103561. issn: 1084-8045. doi: 10.1016/j.jnca.2022.103561. url: https://www.sciencedirect.com/science/article/pii/S1084804522002028 (visited on 2024-01-26) (cit. on pp. 38–41, 43, 45, 46).

[56] P. Bosshart et al. “P4: programming protocol-independent packet processors”. In: *ACM SIGCOMM Computer Communication Review* 44.3 (2014-07), pp. 87–95. issn: 0146-4833. doi: 10.1145/2656877.2656890. url: https://doi.org/10.1145/2656877.2656890 (visited on 2022-11-30) (cit. on pp. 39, 40).

[57] *Open Compute Project*. en. url: https://www.opencompute.org (visited on 2024-03-15) (cit. on p. 40).

[58] *Indigo - Confluence*. url: https://floodlight.atlassian.net/wiki/spaces/Indigo/overview?homepageId=2392077 (visited on 2024-03-15) (cit. on p. 41).

[59] *Open Network Linux*. url: http://opennetlinux.org/ (visited on 2024-03-18) (cit. on p. 44).

[60] *Stratum*. en-US. url: https://opennetworking.org/stratum/ (visited on 2024-03-18) (cit. on p. 44).

[61] *Sonic Foundation – Linux Foundation Project*. en-US. url: https://sonicfoundation.dev/ (visited on 2024-03-18) (cit. on p. 44).

[62] *OpenConfig*. url: https://openconfig.net/ (visited on 2024-03-15) (cit. on pp. 45, 46).

[63] L. Zhu et al. "SDN Controllers: A Comprehensive Analysis and Performance Evaluation Study". In: *ACM Computing Surveys* 53.6 (2020), 133:1–133:40. issn: 0360-0300. doi: 10.1145/3421764. url: https://doi.org/10.1145/3421764 (visited on 2023-02-04) (cit. on pp. 47, 48).

[64] K. Smida et al. "Efficient SDN Controller for Safety Applications in SDN-Based Vehicular Networks: POX, Floodlight, ONOS or OpenDaylight?" In: *2020 IEEE Eighth International Conference on Communications and Networking (ComNet)*. ISSN: 2473-7585. 2020-10, pp. 1–6. doi: 10.1109/ComNet47917.2020.9306095. url: https://ieeexplore.ieee.org/abstract/document/9306095 (visited on 2024-02-19) (cit. on pp. 50, 52, 55, 65).

[65] I. Ku et al. "Towards software-defined VANET: Architecture and services". In: *2014 13th Annual Mediterranean Ad Hoc Networking Workshop (MED-HOC-NET)*. 2014-06, pp. 103–110. doi: 10.1109/MedHocNet.2014.6849111 (cit. on pp. 50, 52, 60).

[66] J. Bhatia et al. "Software defined vehicular networks: A comprehensive review". en. In: *International Journal of Communication Systems* 32.12 (2019), e4005. issn: 1099-1131. doi: 10.1002/dac.4005. url: https://onlinelibrary.wiley.com/doi/abs/10.1002/dac.4005 (visited on 2022-10-11) (cit. on pp. 50, 55).

[67] N. Cardona et al. "Software-Defined Vehicular Networking: Opportunities and Challenges". In: *IEEE Access* 8 (2020). Conference Name: IEEE Access, pp. 219971–219995. issn: 2169-3536. doi: 10.1109/ACCESS.2020.3042717 (cit. on pp. 52, 54–56).

[68] W. Ben Jaballah, M. Conti, and C. Lal. "Security and design requirements for software-defined VANETs". In: *Computer Networks* 169 (2020-03), p. 107099. issn: 1389-1286. doi: 10.1016/j.comnet.2020.107099. url: https://www.sciencedirect.com/science/article/pii/S1389128619306553 (visited on 2023-12-15) (cit. on pp. 52–54).

[69] R. Sarpong, B. Sousa, and F. Araujo. "The potential of SDNs and multihoming in VANETs: A Comprehensive Survey". en. In: (2023-11) (cit. on pp. 52, 59).

[70] S. Toufga et al. "Towards Dynamic Controller Placement in Software Defined Vehicular Networks". In: *Sensors* 20 (2020-03), p. 1701. doi: 10.3390/s20061701 (cit. on p. 54).

[71] S. Toufga et al. “OpenFlow based Topology Discovery Service in Software Defined Vehicular Networks: limitations and future approaches”. In: *2018 IEEE Vehicular Networking Conference (VNC)*. ISSN: 2157-9865. 2018-12, pp. 1–4. doi: 10.1109/VNC.2018.8628349 (cit. on pp. 54, 55).

[72] L. Nkenyereye et al. “Software-Defined Network-Based Vehicular Networks: A Position Paper on Their Modeling and Implementation”. en. In: *Sensors* 19.17 (2019-01). Number: 17 Publisher: Multidisciplinary Digital Publishing Institute, p. 3788. issn: 1424-8220. doi: 10.3390/s19173788. url: https://www.mdpi.com/1424-8220/19/17/3788 (visited on 2022-10-10) (cit. on p. 54).

[73] F. Raviglione, M. Malinverno, and C. Casetti. “Open source testbed for vehicular communication”. In: *Proceedings of the Twentieth ACM International Symposium on Mobile Ad Hoc Networking and Computing*. Mobihoc ’19. New York, NY, USA: Association for Computing Machinery, 2019-07, pp. 405–406. isbn: 978-1-4503-6764-6. doi: 10.1145/3323679.3326623. url: https://dl.acm.org/doi/10.1145/3323679.3326623 (visited on 2024-03-06) (cit. on pp. 56, 58).

[74] R. Sedar et al. “Standards-Compliant Multi-Protocol On-Board Unit for the Evaluation of Connected and Automated Mobility Services in Multi-Vendor Environments”. en. In: *Sensors* 21.6 (2021-01). Number: 6 Publisher: Multidisciplinary Digital Publishing Institute, p. 2090. issn: 1424-8220. doi: 10.3390/s21062090. url: https://www.mdpi.com/1424-8220/21/6/2090 (visited on 2022-10-10) (cit. on pp. 56, 58).

[75] G. Secinti et al. “Software Defined Architecture for VANET: A Testbed Implementation with Wireless Access Management”. In: *IEEE Communications Magazine* 55.7 (2017-07). Conference Name: IEEE Communications Magazine, pp. 135–141. issn: 1558-1896. doi: 10.1109/MCOM.2017.1601186 (cit. on pp. 57, 58).

[76] *Open vSwitch*. url: https://www.openvswitch.org/ (visited on 2024-10-24) (cit. on p. 57).

[77] P. Rito et al. “Aveiro Tech City Living Lab: A Communication, Sensing, and Computing Platform for City Environments”. In: *IEEE Internet of Things Journal* 10.15 (2023-08). Conference Name: IEEE Internet of Things Journal, pp. 13489–13510. issn: 2327-4662. doi: 10.1109/JIOT.2023.3262627. url: https://ieeexplore.ieee.org/abstract/document/10083171 (visited on 2024-02-21) (cit. on pp. 57, 58).

[78] O. Sadio, I. Ngom, and C. Lishou. “Design and Prototyping of a Software Defined Vehicular Networking”. In: *IEEE Transactions on Vehicular Technology* 69.1 (2020-01). Conference Name: IEEE Transactions on Vehicular Technology, pp. 842–850. issn: 1939-9359. doi: 10.1109/TVT.2019.2950426 (cit. on pp. 57, 58).

[79] *PC Engines apu2 system boards*. url: https://www.pcengines.ch/apu2.htm (visited on 2024-11-25) (cit. on p. 62).

[80] R. P. Ltd. *Buy a Raspberry Pi 5*. en-GB. url: https://www.raspberrypi.com/products/raspberry-pi-5/ (visited on 2024-11-25) (cit. on p. 62).

[81] *Deploy AMD*. url: https://opennetlinux.org/doc-deploy-amd.html (visited on 2024-11-25) (cit. on p. 63).

[82] *https://gitlab.com/hpi-potsdam/osm/g5-on-linux/11p-on-linux#id14*. url: https://gitlab.com/hpi-potsdam/osm/g5-on-linux/11p-on-linux#id14 (visited on 2024-11-25) (cit. on p. 63).

[83] *ath10k support (#9) · Issues · HPI Potsdam / Operating Systems and Middleware Group / ETSI ITS-G5 on Linux / IEEE 802.11p on Linux · GitLab*. en. 2023-10. url: https://gitlab.com/hpi-potsdam/osm/g5-on-linux/11p-on-linux/-/issues/9 (visited on 2024-11-25) (cit. on p. 64).

[84] *Open vSwitch*. url: https://www.openvswitch.org/ (visited on 2024-11-25) (cit. on p. 64).

[85] *p4lang/behavioral-model: The reference P4 software switch*. url: https://github.com/p4lang/behavioral-model (visited on 2024-11-25) (cit. on p. 64).

[86] *opennetworkinglab/onos: Open Network Operating System*. url: https://github.com/opennetworkinglab/onos (visited on 2024-11-25) (cit. on p. 65).

[87] *External products sold which can use ath9k — Linux Wireless documentation*. url: https://wireless.docs.kernel.org/en/latest/en/users/drivers/ath9k/products/external.html (visited on 2024-11-25) (cit. on p. 70).

[88] *Common Configuration Issues — Open vSwitch 3.4.90 documentation*. url: https://docs.openvswitch.org/en/latest/faq/issues/ (visited on 2024-11-25) (cit. on p. 81).

[89] *Baco-66/onos-apps: ONOS application for SDVN scenarios, based on the NGSDN Tutorial (Advanced) repository by the Open Networking Lab*. url: https://github.com/Baco-66/onos-apps (visited on 2024-11-25) (cit. on p. 85).

[90] *p4lang/p4c: P4_16 reference compiler*. url: https://github.com/p4lang/p4c (visited on 2024-11-25) (cit. on p. 85).

[91] *Apache Thrift - Home*. url: https://thrift.apache.org/ (visited on 2024-11-25) (cit. on p. 96).

I

Bundles of services

Service bundle	C-ITS Services	Rationale
Bundle 1 Day 1, V2V, ITS-G5	• Emergency brake light • Emergency vehicle approaching • Slow or stationary vehicle(s) • Traffic jam ahead warning • Hazardous location notification	• Day 1 safety-based V2V services based on ITS-G5 communication, likely to be deployed to vehicles supported by US legislation
Bundle 2 Day 1, V2I, mainly applicable to motorways	• In-vehicle signage • In-vehicle speed limits • Probe vehicle data • Shockwave damping • Road works warning • Weather conditions	• Day 1 V2I, services that deliver most benefit to motorways. Some services listed here may also be applicable to other road types
Bundle 3 Day 1, V2I, mainly applicable to urban areas	• Green Light Optimal Speed Advisory (GLOSA) / Time To Green (TTG) • Signal violation/Intersection safety • Traffic signal priority request by designated vehicles	• Day 1 V2I, services expected to only be applicable in urban areas. Therefore, these services are in a separate bundle to those in Bundle 2
Bundle 4 Day 1.5, V2I, Parking Information	• Off street parking information • On street parking management and information • Park & Ride information • Information on AFV fuelling & charging stations	• C-ITS services intended to provide information regarding parking (and refuelling) to drivers
Bundle 5 Day 1.5, V2I, Traffic and other information	• Traffic information and smart routing	• C-ITS services intended to provide traffic information to drivers
Bundle 6 Day 1.5, Freight specific services	• Loading zone management • Zone access control management	• Zone management services
Bundle 7 Day 1.5, V2X (mainly applicable to urban areas), likely to be ITS-G5	• Vulnerable road user protection (pedestrians and cyclists)	• V2X service expected to be post day 1. Communication method is likely to be ITS-G5. Main benefits are likely to be seen in urban areas.
Bundle 8 Day 1.5, V2V, likely to be ITS-G5	• Cooperative collision risk warning • Motorcycle approaching indication	• Post day 1 V2V services that are likely to be based on ITSG5. As for Day 1 services, V2V and V2I services are in separate service bundles.
Bundle 9 Day 1.5, V2I	• Wrong way driving	• Post day 1 V2I service. As for Day 1 services, V2V and V2I services are in separate service bundles.

Table 3: C-ITS service bundles for scenario building[16]