Deliverable D6.2

Implementation of the selected scenarios, analysis, and conclusion.

5GRAIL

5G for future RAILway mobile communication system

Deliverable D6.2

Due date of deliverable: 30/06/2023

Actual submission date: 30/06/2023

Leader/Responsible of this Deliverable: Lars Dittmann, DTU

Contributors: Radheshyam Singh (DTU), Tidiane SyLla (UGE), Leo Mendiboure (UGE), Marion Berbineau (UGE), Lars Dittmann (DTU), José Soler (DTU), Imanol de Arriba Ruiz (CAF), David Miguel Louro Gonçalves (IP), Bogdan Burdulean (UIC), Nikolopoulou Vassiliki (UIC), Mário Jorge Cabral Pereira (IP), Miguel Oliveira Baptista Geraldes Freire (IP); Pedro Miguel Dorey Gouveia e Melo (IP)

Document status

Revision	Date	Description
0.1	16/05/2023	First Issue
0.2	02/06/2023	Second Version with contributions of partners
0.3	13/06/2023	Third Version integrating contributions and corrections
1	30/06/2023	Final version

Project funded from the European Union's Horizon 2020 research and innovation programme		
Dissemination Level		
PU	**Public**	X
CO	Confidential, restricted under conditions set out in Model Grant Agreement	
CI	Classified, information as referred to in Commission Decision 2001/844/EC	

Start date of project: 01/11/2020 Duration: 36 months

EXECUTIVE SUMMARY

The main objective of this document is to describe the implementation of the different chosen scenarios within WP6: the simulation frameworks and testbed(s), parameters for simulations and the chosen Key Performance Indicators (KPIs) for analysis and results presentation. It also provides an analysis of a specific common service between road and railway domain and how it has been implemented and verified.

In the introduction, the report presents a summary of the conclusions by D6.1, to link with the previous work done and setting the contextual framework, presenting the scenarios selected from those defined in D6.1. As a result, a set of requirements for an experimentation testbed is outlined.

Based on these requirements, an initial proposal for technological components of an emulator/simulator are presented and a description of the first experimentation sandbox is presented.

Within that sandbox, the first considered scenarios implementation is presented and some of its results commented, together with limitations of the presented experimentation framework.

A second experimentation testbed, developed as a result of the considered initial limitations is presented, with details of some relevant scenarios and comments on its limitations.

Finally, a common/shared service implementation (Emergency Service) definition is provided, together with the description of two different implementations to demonstrate it in the considered testbeds.

ABBREVIATIONS AND ACRONYMS

Abbreviation	Description
CAM	Cooperative Awareness Message
DENM	Decentralised Environmental Notification Message
ETSI	European Telecommunications Standards Institute
EU	European Union
E2E	End-to-End
FRMCS	Future Railway Mobile Communication System
GA	Grant Agreement
UPF	Horizon 2020 framework program
KPI	Key Performance Indicator
Mx	Mobility Scenario Number x
MQTT	Message Queue Telemetry Transport
NFV	Network Function Virtualization
OAI	Open Air Interface
ONOS	Open Network Operating System
Px	Topology Scenario Number x
RAN	Radio Access Network
RAT	Radio Access Technology
UDP	User Datagram Protocol

UPF	User Plane Function
SYxx	Scenario number xx based on Telco Case Y
SDN	Software Defined Networking
TCP	Transmission Control Protocol
VLAN	Virtual Local Area Network
V2I	Vehicle to Roadside Unit or Vehicle to Base Station
V2V	Vehicle-to-Vehicle

CONTENTS

Table of figures

List of tables

1 INTRODUCTION

The work presented in D6.1 was focused on the conceptual framework for the work in WP6, in relation to coexistence of the railway and road domains and from the point of view of telecommunication infrastructure. Within that context, D6.1 presented the systematic work carried out to define coexistence scenarios between both domains, by considering a shared network infrastructure or two independent infrastructures specific to each domain.

The work presented here in D6.2 presents how, based on a selection from D6.1 scenarios, a set of requirements for an emulator framework was derived and a set of existing (software) solutions was integrated in a software emulator. Some of the initial considered scenarios have been emulated with this framework, and a description is provided as well herein (this relates to Milestone 4 of the project, which was demonstrated with a project workshop in May 2022, presenting these results).

Limitations in this initial 5GRail emulator platform and consideration of additional challenges (allocation of services in near-future cloud-edge scenarios), intended to improve the solution and, based on it, a second emulation platform was developed and is herein presented. Moreover, as the work in the project evolved, it was clear that one of the challenges in the project was the issue of cross-border roaming. Due to it, the second emulator platform has been extended to enable testing of roaming cases. This is also included in this report as well as result analysis of these experiments.

While in D6.1 we concluded that the railway-road coexistence analysis assumed that no common services were envisioned in the near future, in the course of our work in T6.2 and T6.3 we found an example of a cooperative service, Emergency Service, which could be of interest. This is described also in this document as well as two different implementations tested over the first demonstrator.

All these emulation sandboxes, and the experiments carried over them, have been presented in different peer-reviewed publications. The contents of this report are based partially on these publications.

2 INITIALLY CHOSEN COEXISTENCE SCENARIOS & TEST BED REQUIREMENTS

2.1 Coexistence scenarios

The whole range of likely scenarios were described in D6.1. The assumptions at the start of the consideration of activities for T6.2 and T6.3, regarding the proof of concepts (POCs) to emulate, were the following:

- The interesting issues from a communication perspective lay in traffic discrimination within the backhaul/core network for the different domains (cars vs railway), and traffic characterisation in coexistence scenarios (KPI assurance), regardless of the radio access status per scenario (multiple RANs / single RAN, shared RAN / specific RAN);
- This could be implemented based on SDN-based slices over these networks;
- The partners in WP6 did agree that, from all the scenarios identified in D6.1, it made sense to also consider scenarios based on Wi-Fi technology, as a target RAN technology, for complementarity with other cases considered in the project (WP1, WP3, and WP4), and without excluding other technologies (5G as the target).

From the different scenarios defined in D6.1, it is possible to reduce them all to 4 basic cases, from a Telecommunication point of view, as illustrated in Figure 1.

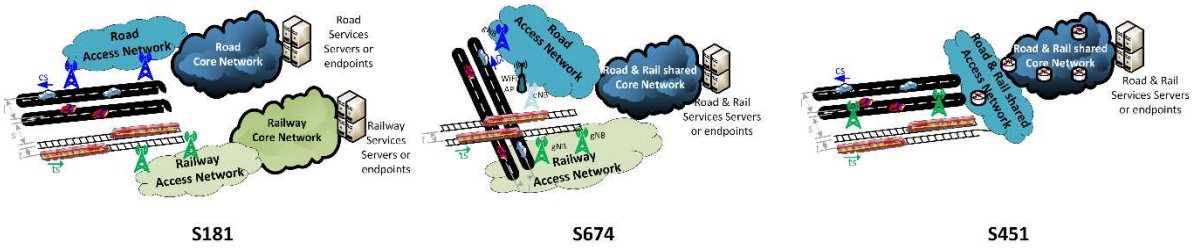

Figure 1: Cases for defined scenarios, from a Telecommunication Networks perspective

Case A: where the different road and railway keep all telecommunication infrastructure separated from each other, as in Scenario 181. This comprises Telco Cases T1 and T5 (Figure 6.4 in D6.1).

Case B: where the backhaul and core network for Road and Railway domains is common and shared, while the radio access networks serving each of the domains are kept separated, as in Scenario 674. This comprises Telco Cases T2 and T6 (Figure 6.4 in D6.1).

Case C: where all telecommunication infrastructure (access and core networks) is common and shared, as in Scenario 451. This comprises Telco Cases T4 and T8 (Figure 6.4 in D6.1).

Note that a fourth case could be considered where the radio access is shared for both domains, but they have different backhaul and core networks (not illustrated in Figure 1). This comprises Telco Cases T3 and T7 (Figure 6.4 in D6.1). This case can be studied from case C.

From this point of view, it can be concluded that the telecommunication infrastructure state is defining the Case baseline. Within each of these cases, the different topological configuration of the road and

railway infrastructure (number of lanes, number of tracks, vehicle-speeds, parallel or perpendicular trajectories) define just variations of what we could term “environment conditions”, which do not alter the essence of the case from the Telecommunication point of view.

It was decided among partners in WP6 that scenarios related to Case A could serve as baseline for KPIs reference, so there should not be a priory discarded or left out of consideration.

As a result, the focus for scenario selection for POCs would narrow down to Case B and Case C: scenarios compliant with Telco Cases 2, 4, 6, 8 or in the following ranges: S2xx (Scenario base on Telco Case 2), S4xx, S6xx, S8xx of those defined in D6.1. For those selected, an equivalent reference scenario compliant with case A would provide baseline KPIs for comparison.

A similar discrimination and selection exercise was also performed in relation to the mentioned “environment variables”. These determine case-variations from the point of view of Mobility and from the point of view of Topology (as those variables defined in D6.1):

- Mobility scenarios range from M1 (Mobility Scenario 1) to M8. M4 and M5 comprises all scenarios in the following ranges: Sx4x, Sx5x of those defined in D6.1. As discussed among partners, the priority was identified among cases [M3-M4] and [M5-M6]. The first ones are related to Highway, where the type of train is only characterised by the speed of the train. As such they can be considered a single case. The second ones are related to Road, where the difference between Tram and Urban Train is to be determined by their speed (and maybe some infrastructure separation). Again, they could be considered as a common case, based on the train speed as a variable. Besides, partners considered relevant to include tunnel scenarios: a scenario where the tunnel section is shared between rail and road was considered as likely relevant due to multipath artifacts caused by moving metallic surfaces and their impact. From the point of view of the emulator tools, the impact of these considerations should be translated into data-traffic effects (losses, modified bit rate or other). It was finally concluded that the case of tunnel could be analysed for a single scenario, such as 251, depending on the effort required and based on the availability of resources when all other cases had been emulated.
- Topologies scenarios range for P1 (Topology Scenario 1) to P4. P1 and P4 comprises all scenarios in the following ranges: Sxx1, Sxx4 of those defined in D6.1. It was concluded that for topologies, the most relevant cases are P1 (roads and rail-tracks parallel) and P4 (level crossing), with the case of tunnel (P3) as a special variation of P1 and as mentioned previously.

These initial considerations gave a baseline for POCs, based on the telco cases defined previously (A,B, C). Over them, the mobility and topological changes could be varied to accommodate the different scenarios. Still, these were a considerable number of scenarios to implement and test. Therefore, an additional discrimination between MUST_HAVE (1 or 2 scenarios) and NICE_TO_HAVE (1 to 3 scenarios) was decided among partners.

A major point was made during the decision discussions: if the telecommunication impact is the driving factor for the emulations, **the coexistence scenarios to concentrate would be those that maximise the duration of this coexistence: low speeds and longer duration from a mobility and topology levels.** Therefore, a strategy would be (e.g., for cases A, B, C) to emulate P1 mobility (e.g., for UrbanRail and Road), and for only one of the cases (e.g., C) to emulate the level crossing.

As a result, the following table tries to summarise the selected scenarios and their priority for implementation:

Table A0: Target Scenarios Prioritising (Must Have (MH) vs Nice to Have (N2H))

Scenario	MH	N2H
S241: • There is a single technology in the access network, although each domain has its own dedicated RAN and both share the core network. • Highway and High-Speed Train for mobility cases. • Track parallel to road, open air/bridge, same plane as topological setup.		X
S2(5/6)1: • There is a single technology in the access network, although each domain has its own dedicated RAN and both share the core network. • Road and Tram / Urban Train. • Track parallel to road, open air/bridge, same plane as topological setup.	X	
S441: • There is a single technology in the access network, the RAN is shared, and both also share the core network. • Highway and High-Speed Train for mobility cases. • Track parallel to road, open air/bridge, same plane as topological setup.		X
S4(5/6)1: • There is a single technology in the access network, the RAN is shared, and both also share the core network. • Road and Tram / Urban Train. • Track parallel to road, open air/bridge, same plane as topological setup.	X	
S4(5/6)3: • There is a single technology in the access network, the RAN is shared, and both also share the core network. • Road and Tram / Urban Train. • Track parallel to road, tunnel, and same plane.		X
S4(5/6)4: • There is a single technology in the access network, the RAN is shared, and both also share the core network. • Road and Tram / Urban Train. • Track perpendicular to road, open air, same plane (level crossing).	X	
S641: • There are different technologies in the access network, each domain has its own dedicated RAN and both share the core network. • Highway and High-Speed Train for mobility cases. • Track parallel to road, open air/bridge, same plane as topological setup.		X
S6(5/6)1: • There are different technologies in the access network, each domain has its own dedicated RAN and both share the core network. • Road and Tram / Urban Train. • Track parallel to road, open air/bridge, same plane as topological setup.		X
S8(5/6)1: • There are different technologies in the access network, the RAN is shared as well as the core network.		X

• Road and Tram / Urban Train. • Track parallel to road, open air/bridge, same plane as topological setup.		

Summarising: 3 main scenarios were defined as main targets for the POCs implementation + the baselines corresponding to the respective Cases A: **S2(5/6)1, S4(5/6)1 and S4(5/6)4 + S1(5/6)1 and S1(5/6)4.**

From the Nice to Have scenarios from the table, the proposed priority was the following: S4(5/6)3, S6(5/6)1, S441, S241, S641 and S8(5/6)1.

2.2 Testbed / Demonstrator requirements

With the context described in the previous section as initial foundation for the work in T6.2, the following requirements were established for the emulation environment, as an enabler for emulation of the selected railway/road coexistence scenarios:

1. 5G-based connectivity should be available or emulated to endpoints.
2. Wi-Fi-based connectivity should be available or emulated to endpoints.
3. It should be possible to define mobility for endpoints (speed, positioning / trajectory), frequency, quantity, etc.
4. It should be possible to define multiple wireless network interfaces for endpoints.
5. It should be possible to define / generate traffic from endpoints.
6. It should be possible to define radio channel characteristics for the radio link or to “modulate” traffic to mimic those characteristics (packet losses, delays …).
7. It should be possible to define different network topologies both for wireless “access points” and fixed network components.
8. It should be possible to programmatically control the behaviour of the network elements.
    - Network entities should provide SDN (OpenFlow) interfaces.
    - VLAN tagging/un-tagging and tag-based routing should be supported.
9. It would be nice to have a graphical interface, which could be used to present emulation results.

In relation to Requirement 5, identification of traffic flows/services for all the scenarios followed. It was decided to define similar flows for all cases. From Table 10. 1 in D6.1 the following were selected:

- ∉ Critical Voice/Data/Video Communication
- ∉ Performance Voice/Data/Video Communication
- ∉ Business Voice/Data/Video Communication

These can be mapped to the following uses from Table 10.2 in D6.1:

- 5.1: On-train Outgoing Voice communication from train driver to controller.
- 5.9: Automatic Train Protection communication.
- 5.10: Automatic Train Operation communication.
- 5.15: Railway Emergency Communication.
- 5.27: Critical real time video.
- 5.28: Critical Advisory Messaging services - safety related.

3 INITIAL DEMONSTRATION SANDBOX

Based on the requirements presented in the previous section, an initial design for a sandbox serving as the base for the emulation platform was considered. The primary aim of this emulator was to enable the various coexistence scenarios identified to be implemented in a realistic network environment, allowing both shared and separate networks to be set up. The 5G-related components have been integrated into an improved version of this emulator, presented in section 3. Identification of existing (open source) components providing the required features was performed in this first step and the work in T6.2 was based on integration efforts of those existing open-source components, towards a stable sandbox. This is described in the following section.

3.1 Sandbox Design and Initial Implementation

The first step was to define the tools used to implement the communication networks needed to deploy railway applications. For network emulation, various emulators exist in the literature, such as Riverbed modeller [1], Omnet++ [2] and NS-3 [3]. All these are rather limited in terms of multiple access technologies/potentially shared network infrastructure emulation SDN support (potential solution ofr network management). A network emulator usual for SDN-based networks management is Mininet [4]. A fork to emulate Wi-Fi networks exists in Mininet, Mininet-WiFi, which can integrate with external SDN controllers, such as Open Network Operating System (ONOS) [5]. Considering the scenarios selected initially, this became our initial preferred set for the emulator. Efforts to integrate Mininet and SUMO [6] were carried out with success. As a result, an initial proof of concept for the emulator was built as a virtual machine, which could be reused by both research teams (UGE and DTU) in a common, homogeneous setup.

The requirement to provide 5G native features could be added to this setup by attaching an external 5G emulator such as those from Open Air Interface (OAI) [7]. Mobility issues would be the limiting factor for OAI, so further investigation targeting augmenting Mininet-WiFi with 5G features was carried out without satisfactory results.

While Requirements 4 and 6 (cf. Section 1.2) are inherent features of Mininet-WiFi, it was necessary to determine to what extent the existing interfaces and propagation models could be reused. Likewise, and in relation to requirement 1 on 5G, it appeared necessary to link to an additional tool to generate some channel effects. Instead, we decided on emulating those effects at network level using NetEm [8] or similar tools to “modulate” the traffic from a packet point of view: drops, bursts, delays, etc.

Validating these integration assumptions and the rest of requirements took considerable part of the initial efforts in T6.2. Once this was completed, to speed up the production of results, the issue of 5G radio integration in the emulator was postponed and priority was given to scenario creation and testing. As a result, a number of tests were performed to validate the architecture designed for this emulator as well as the functionality of the container-based SDN controller (ONOS version 2.5) installed in the emulator's virtual platform. Furthermore, a number of example applications was ported to this controller version and issues related to development differences for the updated software version were faced and solved. The efforts resulted in the SDN functionality fully tested and applicability and implementation methodology documented and ready for the creation of the selected cases later on, including:

- Functionality to enable SDN-based slicing of a target network.
- SDN-based management of handover from cell to cell.

Also, a private GitHub repository was created for the project and subfolders specific for the different tasks defined, so that the different work and code could be updated and available to UGE and DTU teams. All the tests and code created during were uploaded and available there.

By early 2022 a complete sandbox with multiple components allowing it to cope with most of the initial requirements was ready and a set of demonstration cases were completed during the spring as required by MS4.

The initial design and a description of its components is the following: Based on the requirements presented above, the ONOS SDN controller [5] was selected to programmatically control the network topology, and Mininet–WiFi [9] was selected to define the network topology. The tool SUMO [6] was chosen for the graphical representation of the selected scenarios. As displayed in Figure 2, these components were installed in a Virtual machine, which allowed to have a duplicated setup for the different teams working on the recreation of scenarios. The key properties of each of them is briefly explained in the following.

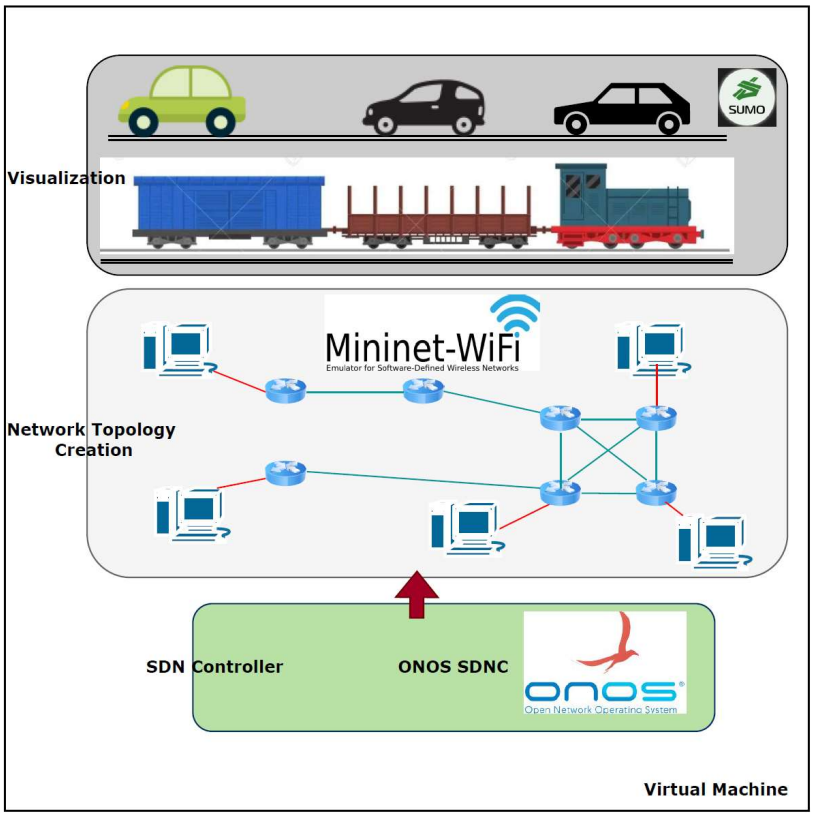

Figure 2: Emulation Platform Components

3.1.1 ONOS SDN Controller

The ONOS software-defined network controller is an open-sourced SDN and network function virtualisation (NFV) controller. A simplified programmatic interface makes ONOS an ideal platform for operators searching to build innovative and advanced network services. ONOS has the ability to configure and control the network by programming the functionality and reducing network protocol implementation requirements. The ONOS cloud controller integrates intelligence, enabling end-users to easily create new network applications without having to change the data plane [5].

3.1.2 Mininet–WiFi

Mininet–WiFi [9] is a software-defined network emulator. It is a branch of Mininet [4] embedded with additional functionalities such as the ability to define and configure Wi-Fi access points and nodes with moving capability based on Linux wireless driver and simulation driver 80211_hwsim [30]. Using Mininet–WiFi, users can define different network topologies, where host/nodes can be defined with multiple wireless interfaces. Along with these, Mininet–WiFi supports defining radio parameters such as operating frequency channel, propagation model, coverage range, and transmission power (Tx-Power). The network topologies developed with Mininet–WiFi have the potential to be controlled programmatically, based on OpenFlow protocol versions 1 through 5. Since it works on the Linux wireless driver 80211_hwsim, it does not have the ability to emulate 5G-based connectivity. Therefore, it can be observed that Mininet–WiFi fulfils the requirements that are taken into consideration for this practical work. Only Requirement 8 is not covered by this selected tool. We discuss this in the Conclusions (Section 3.11). Mininet–WiFi installation files and processes are available at [9,24].

3.1.3 SUMO

Simulation of Urban MObility, commonly known as SUMO [6,25,26], is an open-sourced traffic simulator used to design and visualise the mobility of vehicular networks. SUMO supports features such as multimodal and continuous mobility of selected nodes/stations. Using SUMO, users can define the speed and quantity of selected nodes (cars, train, tram, bicycle, etc.). Based on the user's interest, the simulation area can be extracted directly from the open street map, where users can select the intended simulation area and download the simulation map files. Further, an additional feature can be added that shows the map area with assigned access points and nodes in a graphical manner. The authors of [27,28] used SUMO for visualization, modelling, and defining nodes in traffic routes. Therefore, SUMO is considered to fulfil Requirement 9 mentioned previously.

3.1.4 Integration of Initial components

Figure 2 represents the test setup overview. All the selected tools that are considered to emulate the railway and road coexistence scenarios are installed on a virtual machine. The considered network topology is created using Mininet–WiFi for road and railway coexistence scenarios. To control the functionality of the network topology, an SDN application is developed, installed, and activated for the ONOS SDN controller. The SDN application is developed to support the moving of end nodes and the inter-cell handover of created nodes in a defined virtual space. It also has the ability to differentiate the data traffic based on VLAN tagging. Detailed information about this developed SDN application is elaborated in Section 8. SUMO is integrated with Mininet–WiFi to graphically represent the movement of network nodes on an open street map.

3.2 Final Selected Tools to Generate Data Traffic to Validate the Scenarios

Table 2 shows the selected tools to generate the different kinds of data traffic compliant with the real case scenario.

Table 1: Selected tools to generate different kinds of data traffic for compliance with real case scenario

Scenario	Considered Tool to Demonstrate the Scenario	Tool Information
• Voice communication for Operational Purpose, • Standard Data Communication	Iperf3	Iperf3 can send UDP and TCP packets from one host to another.
• Critical Data Communication • Very Critical Data Communication • Messaging	Scapy	Using Scapy, we can define our data packets and send them to the network. Using Scapy, messaging and critical data communication is demonstrated
• Critical Video Communication for Observation Purpose • Very Critical Video Communication Associated with Train Safety	VLC Player	To demonstrate video transmission from train or car to the assigned server, a VLC player is used.
• Measure Network Quality of Services (QoS)	MTR	MTR tool has the capability to measure the latency, packet loss, and jitter of the network.

3.3 ONOS SDN Application for Data Traffic Slicing

The most significant task of this empirical work is to design and develop SDN applications capable of fulfilling the following objectives:

- Supports handover/moving capability of nodes/hosts.
- Differentiates the data traffic based on VLAN tagging/slicing.
- Is scalable to support any kind of network topology.

An SDN data-forwarding application has been created using ONOS JAVA APIs [5], which has been deployed in the ONOS controller with the aim of enabling network slicing and differentiating data traffic between railways and roads. The application ensures that only trains can communicate with other trains and assigned rail service servers, while cars can only communicate with other cars and assigned car service servers. In addition, the application manages the movement and handover of nodes between assigned access points/cells. The application uses the packet processor, an ONOS API that defines the header context of packets and activates the developed applications. The application creates two arrays, one containing the IP addresses of all cars, and the other containing the IP addresses of all trains. The application makes decisions on whether a data packet should be forwarded or dropped between the nodes, switches, and access points. Figure 3 illustrates the various steps involved in the developed ONOS application. Once the application is installed and activated, it initialises the packet processor and activates the two IP address arrays.

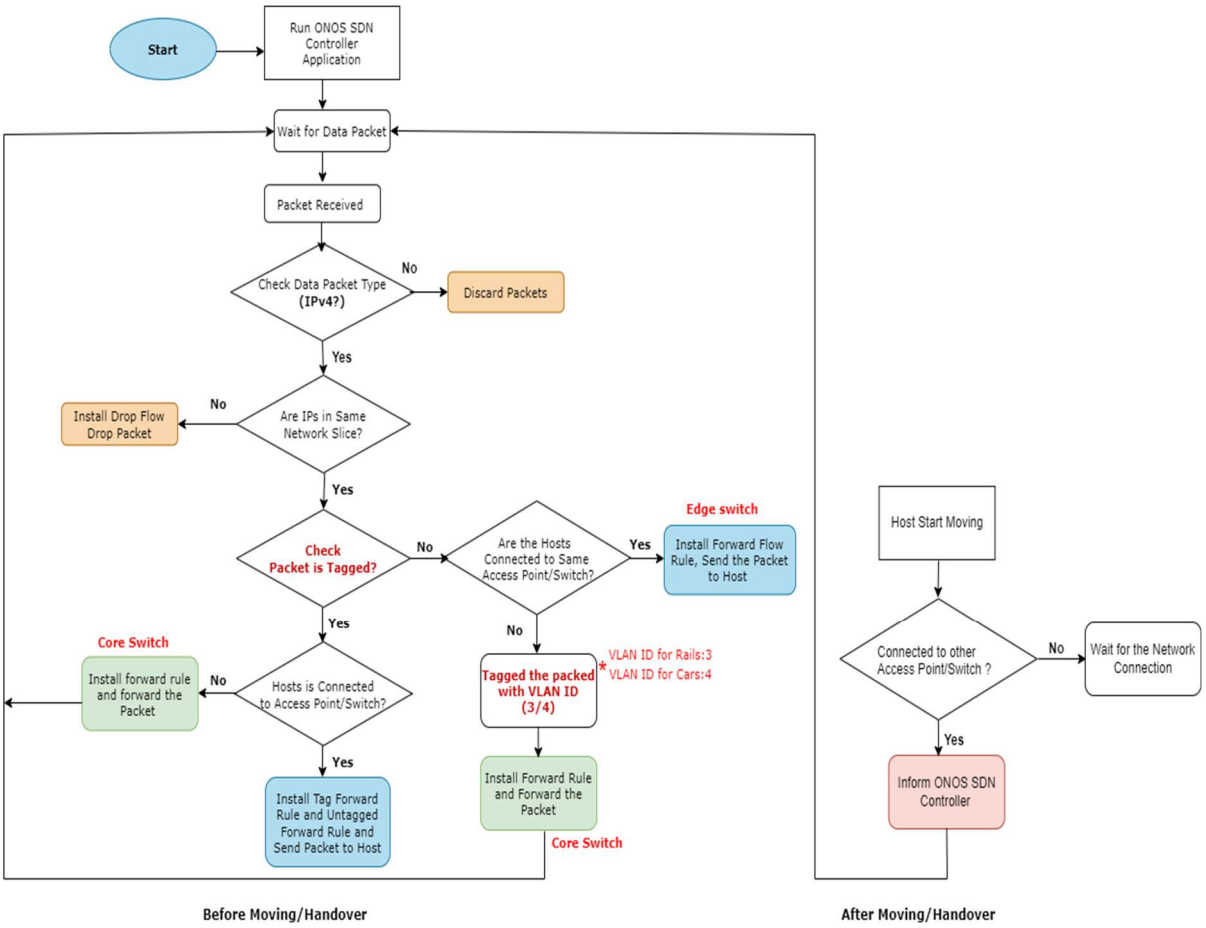

Figure 3: Flow Diagram of SDN Application

The developed ONOS application has been designed to support IPv4 data packets. When an IPv4 packet arrives at any access point or switch, the application checks its forwarding table/rules. If there are no forwarding rules for the source and destination IP pair at the current switch/access point, the packet is sent to the ONOS controller as an OpenFlow "PacketIn" message for processing. The application checks whether both the source and destination IPs belong to the same network slice (trains or cars), and if they do not, it installs a packet drop rule at the current switch/access point using the OpenFlow13 protocol. This disables any traffic between cars and trains, and their assigned service servers.

If the source and destination IP pairs belong to the same network slice, the application checks whether the data packet is tagged with a VLAN ID or not. If the packet is not tagged, and the source and destination hosts are connected to the same access point/switch, the application installs a forwarding rule using the OpenFlow13 protocol at the current access point/switch and forwards the data packet to the destination host/node. If the data packet is not tagged and the source and destination host/node are connected to the same access point/switch, the application tags the data packet with a VLAN ID based on the network slice. The application uses the number 3 as the VLAN ID to tag the data packets from/to railways/trains and the number 4 to tag the data packets from/to roads/cars. The application then installs the forwarding tag rule and forwarding untag rule using the OpenFlow13 protocol at the current access point/switch for the IP pair and forwards the data packet to the next switch. If the data packet is tagged with a VLAN ID and the source and destination host/node are

connected to the same access point/switch, the application installs the forward tag rule and untag rule for the given source destination IP pair at the current access point/switch. If the data packet is tagged with a VLAN ID and the host/node is not connected to the same access point/switch, the application installs the forward rule at the current access point/switch and forwards the data packet in the network.

If nodes move from one location to another and connect to the nearest assigned access points, the application informs the controller via a "PacketIn" message about the position of the nodes and the connected access points.

3.3.1 VLAN Tagging

VLAN tagging is a technique that enables the creation of several networks at Layer 2 of the core network [22]. It involves adding a VLAN ID to the data header as an extra element to the Ethernet header of a packet. This assigned tag can be utilised as a filtering criterion for the forwarding operation at switches/access points. By matching the tag of the data packet header, the VLAN tag determines which part of the network a data packet belongs to.

In the developed ONOS application, Figure 4 illustrates the use of VLAN tagging to label data packets. The network consists of edge switches S1, S2, S7, and S8, and core switches S3, S4, S5, and S6. Access points ap1, ap2, and ap3 serve as access network elements for cars, while ap4, ap5, and ap6 serve as access network elements for trains. When a data packet is transmitted from a car to the CarServer, access point ap1 adds a VLAN ID of 4 (cars' slice) to the data packet header and sends it to the next switch S3. S3 matches the VLAN ID and forwards the packet to S5, which also matches the VLAN ID and sends the data packet into the network. When the data packet reaches the edge switch S7, where the destination host CarServer is connected, S7 removes the VLAN ID and forwards the data packet to the CarServer.

Similarly, when a train's data packet is sent, access point ap4 tags the data packet with VLAN ID: 3 (trains' slice). Core network switches S4 and S6 match the VLAN ID of the data packet and send it into the network. When the data packet arrives at switch S8, it removes the VLAN ID and forwards the data packet to the host rail server.

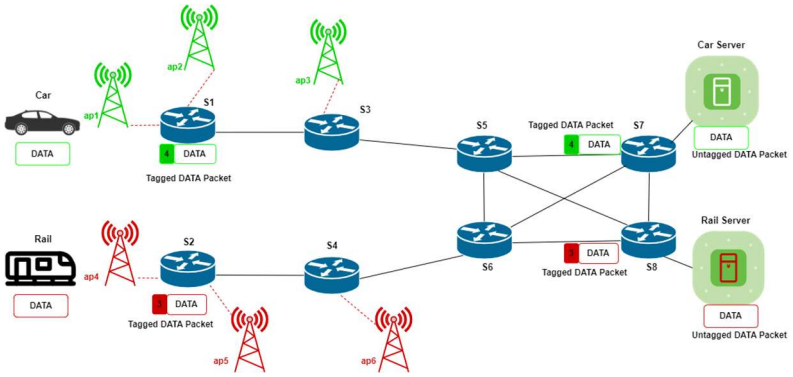

Figure 4: Data Packet Tagging

Tools described in Table 2 are used for the all the considered scenarios S1(5/6)1, S1(5/6)4, S2(5/6)1, S4(5/6)1 and S4(5/6)4 to generate the data traffic which can mimic the actual data traffic. Section 3.4

provides the detailed investigation test for the scenario S1(5/6)1 to validate the considered tools to emulate the coexistence scenarios of railways and roads. Rest of the considered scenarios are also emulated and validated with the considered tool. For the scenarios S1(5/6)4, S2(5/6)1, S4(5/6)1 and S4(5/6)4 test data are presented in the Section 71 Appendix of this documentation.

3.4 Implementation and Tests for Coexistence Scenario 1

As a reminder, **S1(5/6)1: Different Access Network and Different Core, Single Serving Technology, Track Parallel to Road**: This scenario is considered as the baseline scenario to investigate the coexistence of railway and road scenario telecommunication services infrastructure. In this scenario, both domains, i.e., railways and roads, have their own dedicated radio access network (RAN) and dedicated core. Considered access points and cores work on a single technology/radio frequency. Along with this, railway tracks are kept parallel to roads.

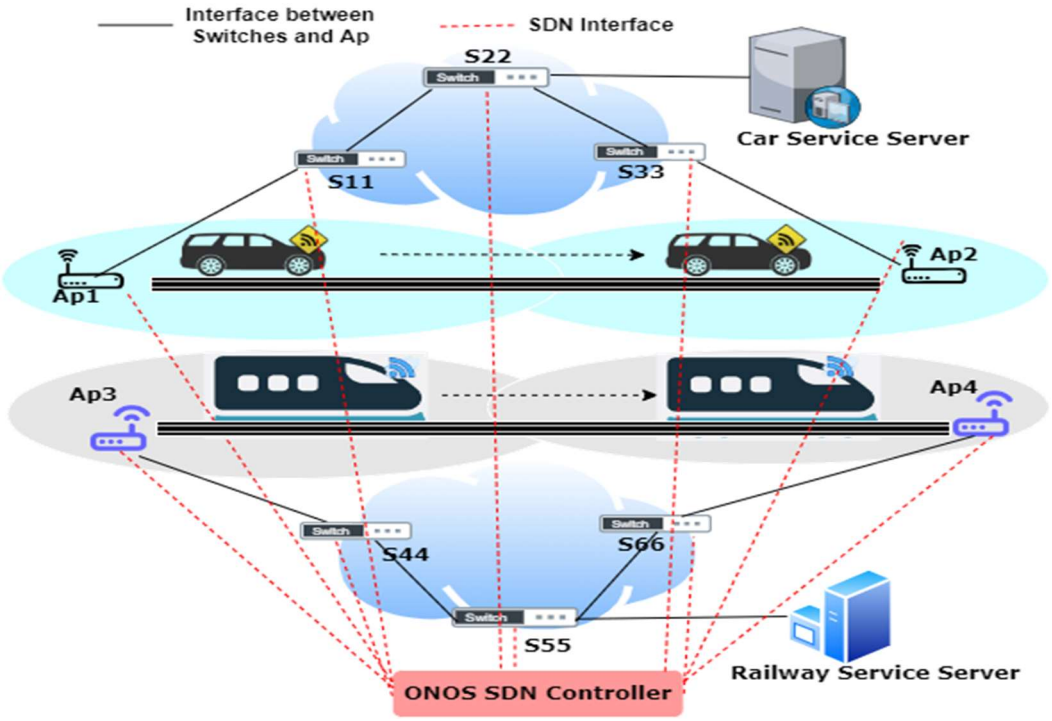

Figure 5: S1(5/6)1 Different Access Network & Different Core, Track parallel to Road

3.4.1 Topology

A Python script is utilised to generate a network topology with Mininet-WiFi where trains and cars are allocated separate access networks. Both domains' haves their own core network, and railways have parallel tracks to roads. Figure 5 showcases the S1(5/6)1 scenario, where the forwarding elements of the topology are programmatically managed by an ONOS SDN controller. The network switches and

access points are devices based on SDN and are managed and controlled through the OpenFlow protocol.

The network topology for the S1(5/6)1 scenario, created with Mininet-WiFi, is illustrated in Figure 6. Hosts Car1, Car2, and Car3 are used to represent cars, while Tra1, Tra2, and Tra3 represent trains. Access points ap1 and ap2 are allocated for roads, with access point ap1 being connected to switch S11 and access point ap2 to switch S33. Switch S22 is connected to both S11 and S33. The host "CarServer" is defined as the road service server and is linked to switch S22. Access points ap3 and ap4 are assigned for railways, with access point ap3 being linked to switch S44 and access point ap4 to switch S66. Switch S55 is connected to both S44 and S66. The "RailServer" host is defined as the railway service server and is linked to switch S55. Nodes Car1, Tra1, and Tra3 are configured with the capability to move to emulate moving cars and trains in this scenario. Figure 7, which is created by Mininet-WiFi, displays the positions of nodes and access points before handover/moving.

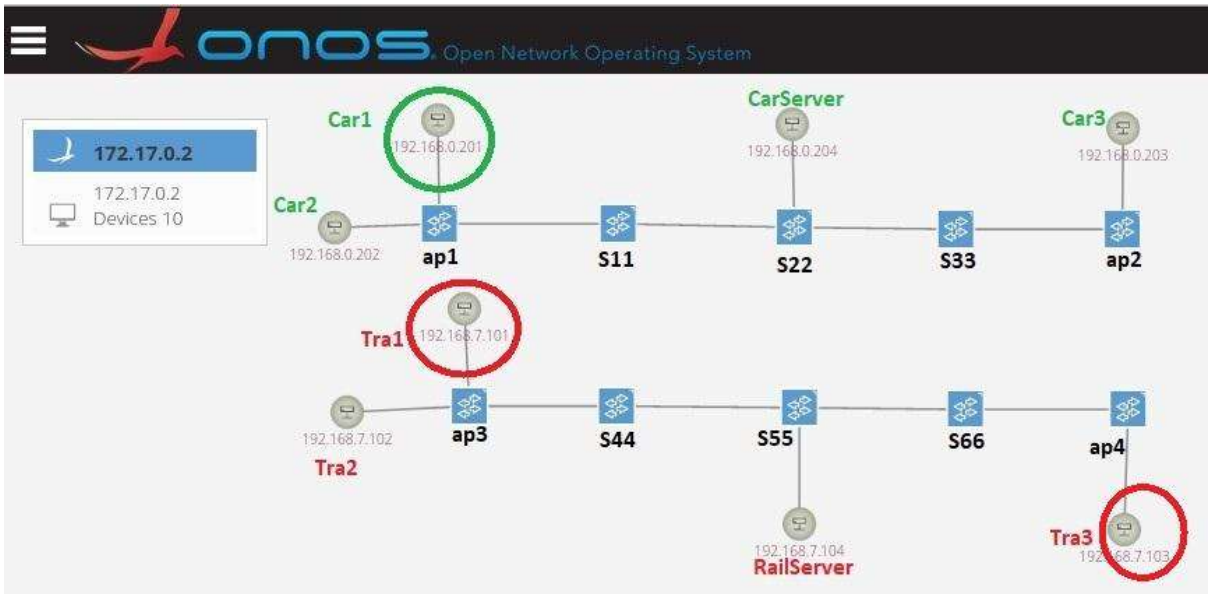

Figure 6: S1(5/6)1 Different Access Network & Different Core, Track parallel to Road: ONOS Screenshot (Before Handover)

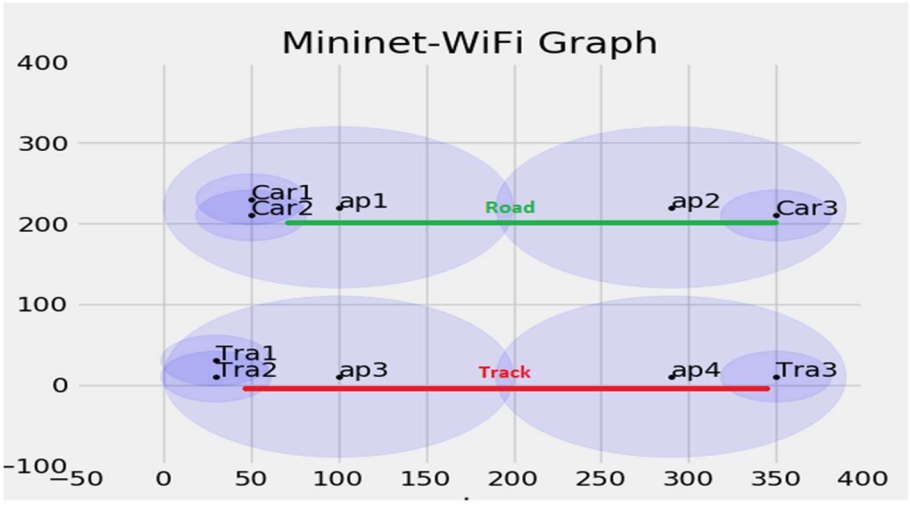

Figure 7: : S1(5/6)1 Hosts and Access Points: Mininet-WiFi Graph

3.4.2 Handover/Moving

To perform a handover/mobility analysis, Car1, Tra1, and Tra3 nodes have been configured with mobility capability. After 60 seconds, Car1 initiates movement towards access point ap2, Tra1 starts moving towards access point ap4, and Tra3 moves towards ap3. During their movement, Car1 pings Car2, while Tra1 pings Tra2. Figure 8 shows the results of the ping test, indicating that when Car1 reaches the edge of access point ap1 and enters the coverage range of access point ap2, it switches its connection from ap1 to ap3. Similarly, when Tra1 enters the coverage range of access point ap4, it switches its connection from ap3 to ap4.

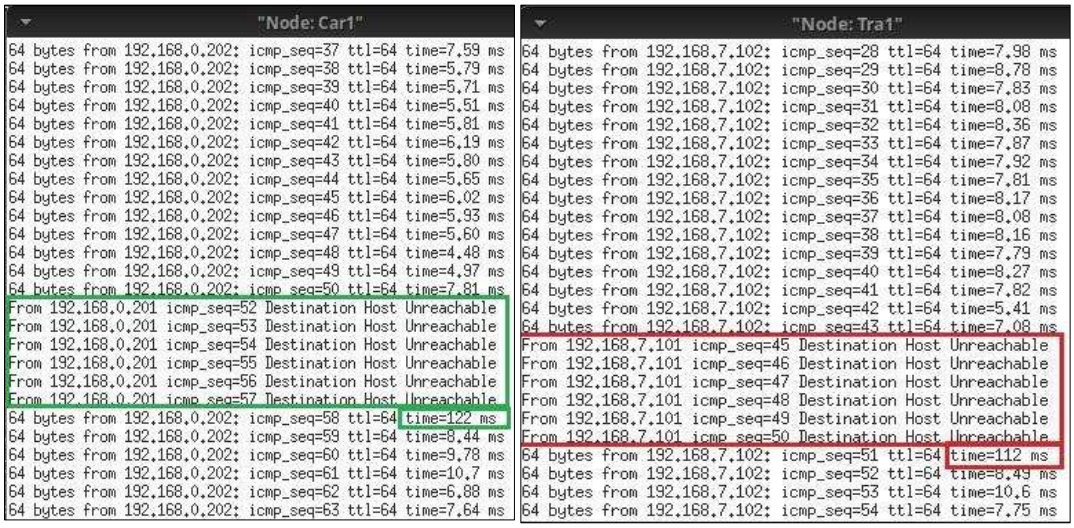

Figure 8: Checking Connectivity During Moving

To test the network connectivity and handover between assigned access points, both selected nodes (Car1 and Tra1) are pinging their respective service servers. As shown in Figure 8, when Car1 and Tra1 cross the coverage range of their previously connected access points, they automatically switch to the nearest access point (ap2 for Car1 and ap4 for Tra1). The figure also indicates that there is no packet loss during the handover process.

To further verify the handover and mobility functionality of the nodes, the command "Car1 iw dev Car1-wlan0 link" is executed for Car1, and "Tra1 iw dev Tra1-wlan0 link" is executed for Tra1, both before and after the nodes' movement. Figure 9 shows that Car1 was initially connected to ap1, but after the handover, it successfully switched to ap2. Similarly, Figure 10 shows that Tra1 was initially connected to ap3, but after the movement, it successfully switched to ap4. Figure 11 displays the topology after the nodes' movement, indicating that Car1 is now connected to ap2, and Tra1 is connected to ap4.

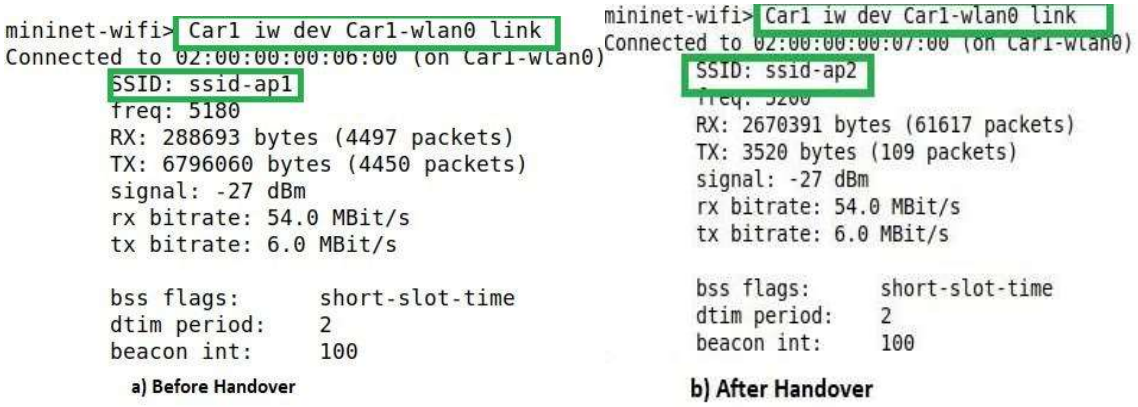

Figure 9: Connected Access Point for Car1 Before and After Handover/Moving

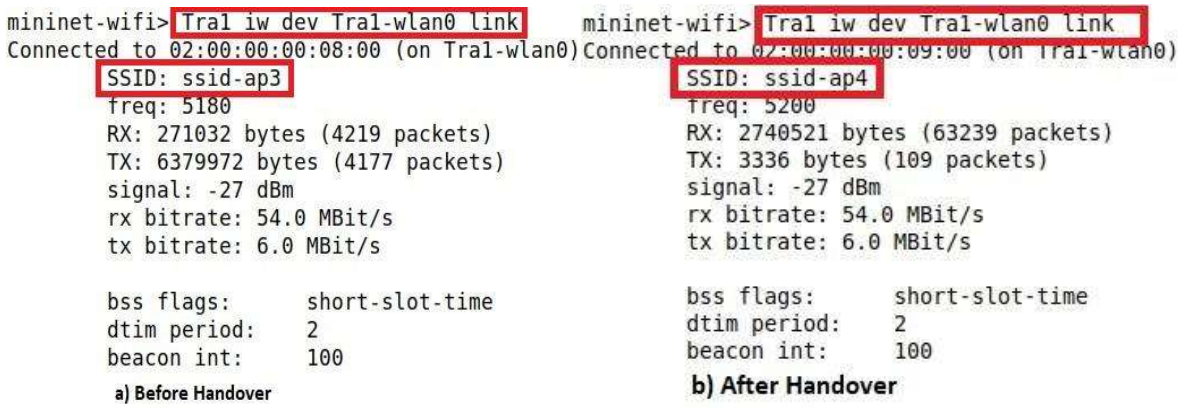

Figure 10: Connected Access Point for Tra1 Before and After Handover/Moving

Figure 7 illustrates the initial position of nodes and access points for scenario S1(5/6)1 before the movement, while Figure 12 displays their positions after the movement and handover. A comparison of these two figures indicates that Mininet-WiFi has the capability to effectively simulate moving and handover scenarios for coexisting railway and road environments. Although there is a delay in the handover process, there is no recorded data loss. This delay occurs due to the network joining process carried out by the nodes/stations during the handover.

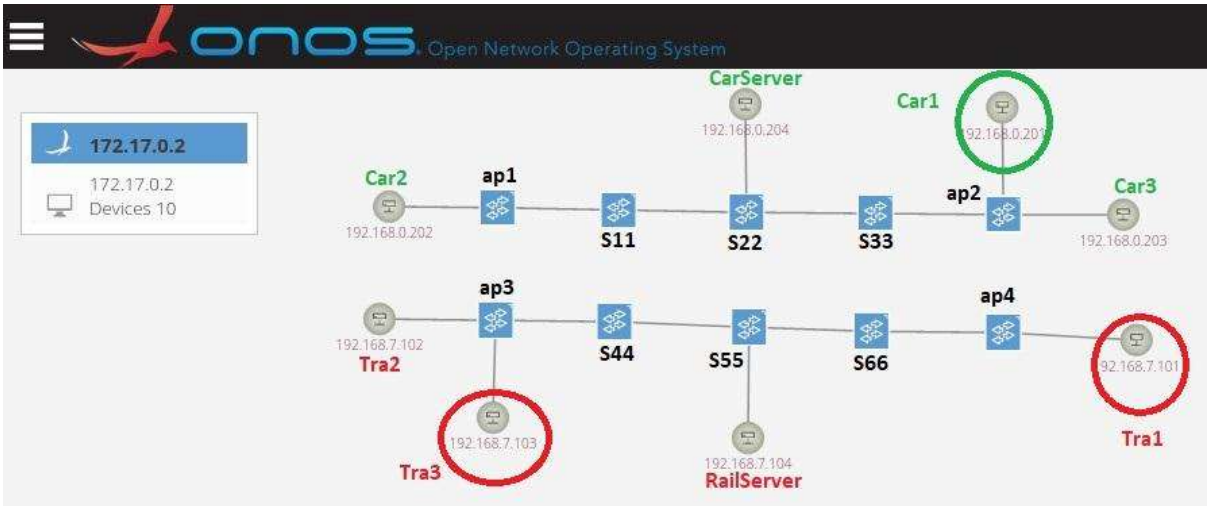

Figure 11: S1(5/6)1 Different Access Network & Different Core, Track parallel to Road: ONOS Screenshot (After Handover)

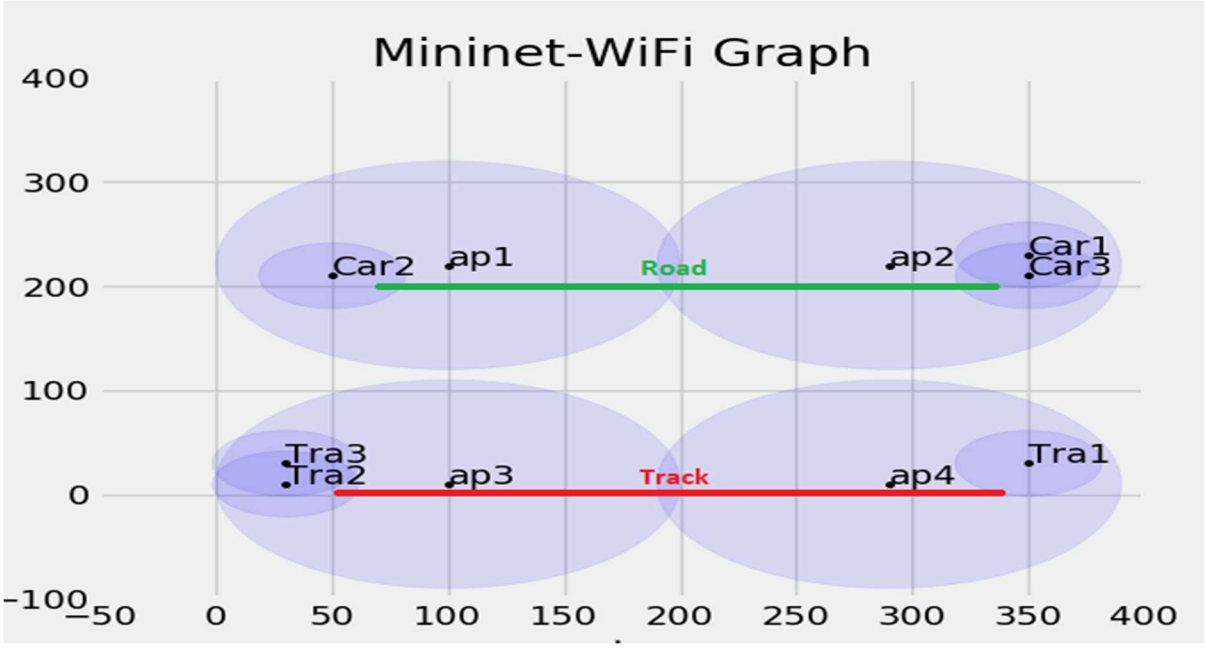

Figure 12: S1(5/6)1 Hosts and Access Points: Mininet-WiFi Graph (After Handover)

3.4.3 Reachability Test and Data Traffic Differentiation

The test was conducted for all nodes and hosts connected to the network topology S1(5/6)1, and the results are presented in Table 3. Based on the table, it is evident that Cars can only communicate with other Cars and the assigned road service server, which is CarServer. On the other hand, Trains can only communicate with other Trains and the designated railway service server, which is RailServer.

Table 2: Reachability Test

Src/Dst	Car1	Car2	Car3	CarServer	Tra1	Tra2	Tra3	RailServer
Car1	ü	ü	ü	ü	X	X	X	X
Car2	ü	ü	ü	ü	X	X	X	X
Car3	ü	ü	ü	ü	X	X	X	X
CarServer	ü	ü	ü	ü	X	X	X	X
Tra1	X	X	X	X	ü	ü	ü	ü
Tra2	X	X	X	X	ü	ü	ü	ü
Tra3	X	X	X	X	ü	ü	ü	ü
RailServer	X	X	X	X	ü	ü	ü	ü

3.4.4 TCP and UDP Data Transmission

The purpose of conducting this test is to showcase the standard data communication between cars and CarServer, as well as between trains and RailServer. Figure 14 illustrates the transmission of UDP data packets, while Figure 14 illustrates the transmission of TCP data packets from Tra1 to RailServer. In this case, Tra1 is functioning as a client, while RailServer is configured as a listening server.

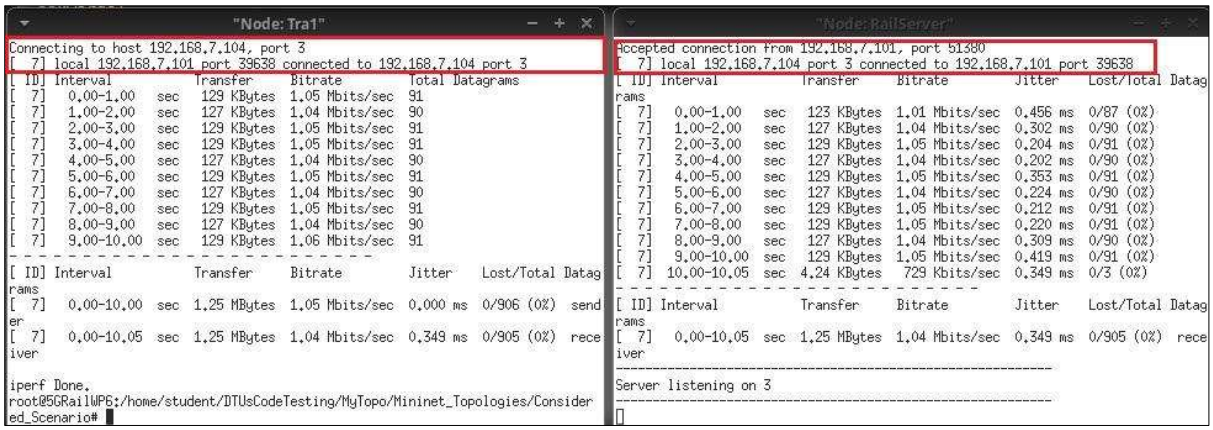

Figure 13: UDP Data Packet Transmission from Tra1 to RailServer

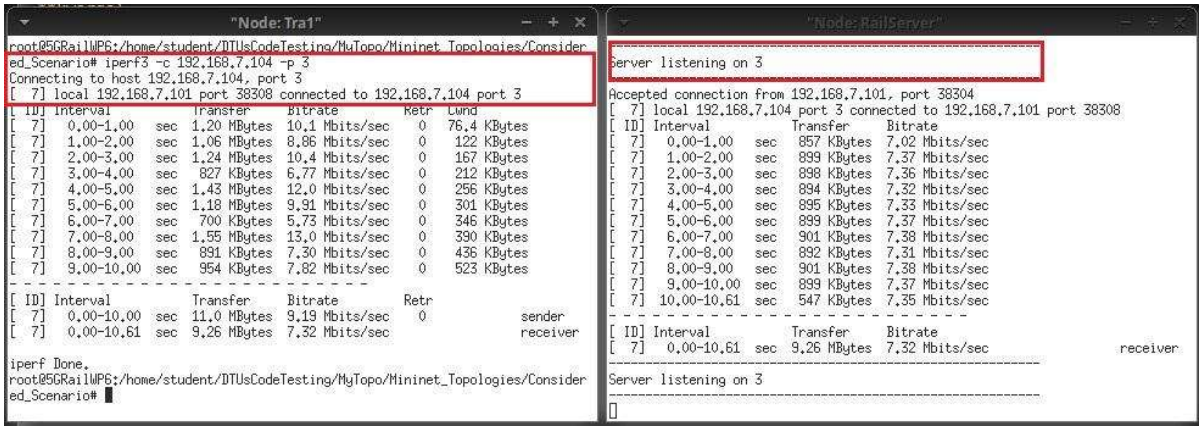

Figure 14: TCP Data Packet Transmission from Tra1 to RailServer

3.4.5 Link Capacity Test

The bandwidth between two network links is being measured using the iperf tool in this test. The command "iperf <Host1> <Host2>" is used to measure the bandwidth between the hosts. The measurement of link capacity between Car1 and CarServer and Tra1 and RailServer is shown in Figure 15. The obtained bandwidth measurement is sufficient for transmitting and receiving messages, voice, and video data in coexistence scenarios for both roads and railways.

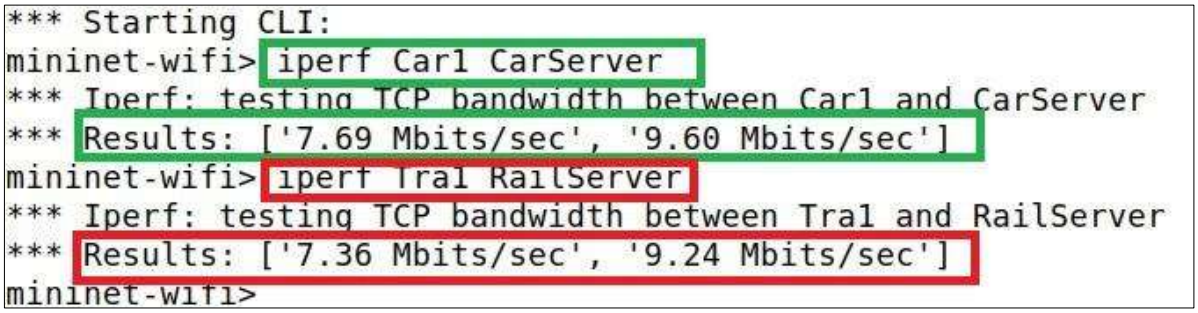

Figure 15: Link Capacity Test

3.4.6 Latency Test and Network Jitter Test

To measure losses, latency, and network jitter, the MTR tool is utilised. To perform the latency test, 100 UDP and TCP data packets are transmitted from Car1 to CarServer and Tra1 to RailServer. For this network topology, the latency ranges from 4.8 to 5.1 milliseconds, as displayed in Figures 16 and 17. Figures 18 and 19 represent that the network jitter ranges from 4.6 to 4.8 milliseconds.

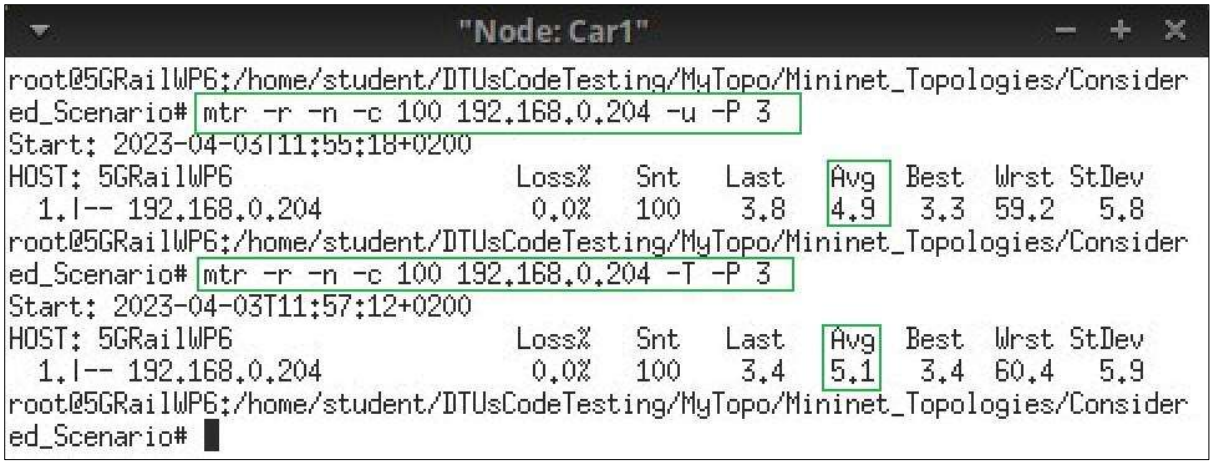

Figure 16: Latency Test from Car1

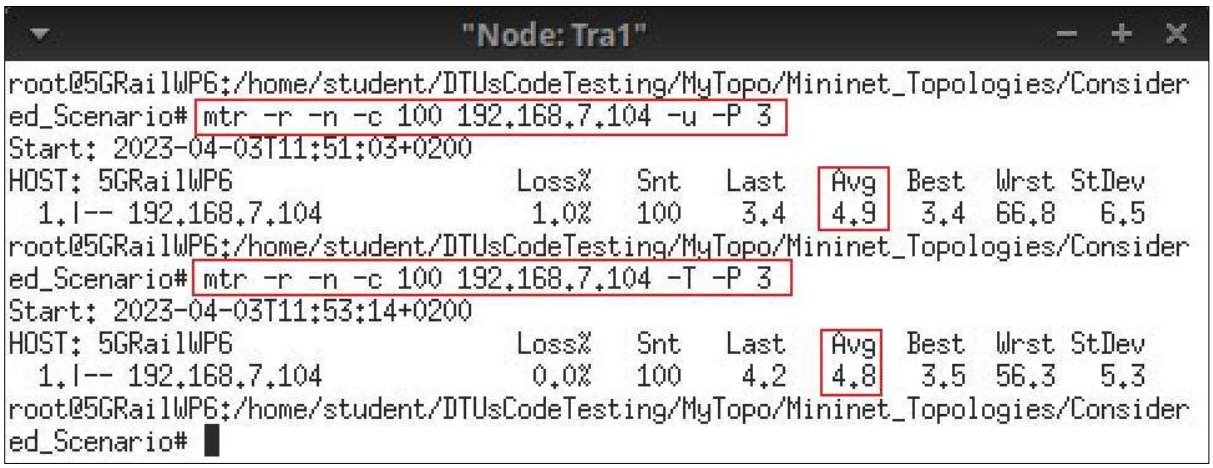

Figure 17: Latency Test from Tra1

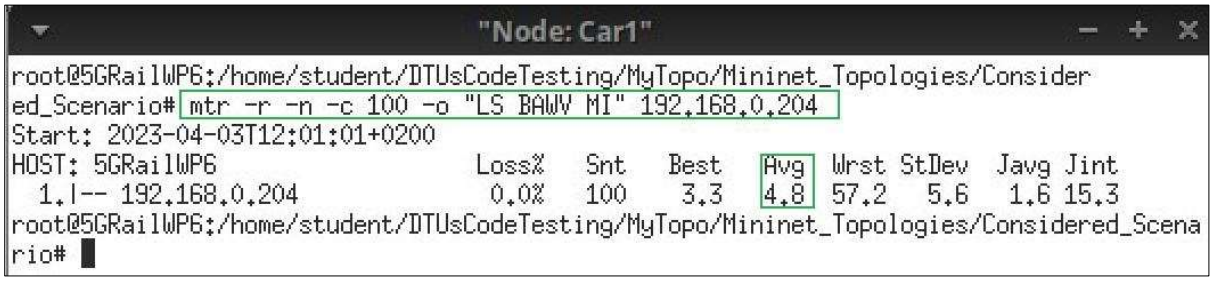

Figure 18: Jitter Test from Car1

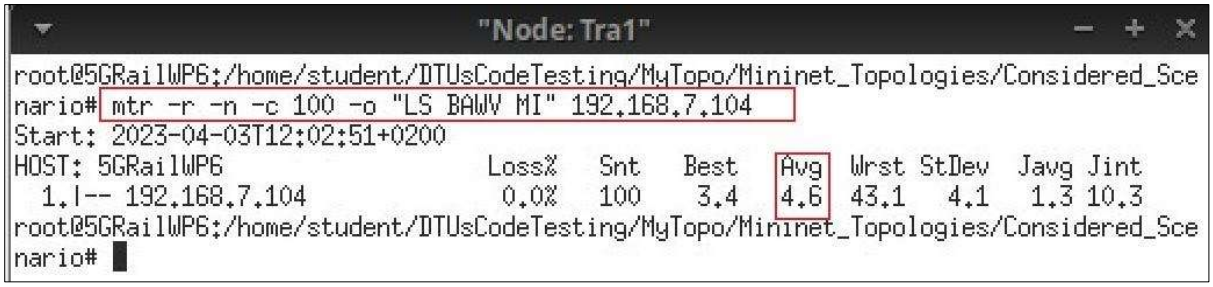

Figure 19: Jitter Test from Tra1

3.4.7 Sending a Critical Message to the Assigned Server

The Scapy tool is an efficient way to send customised messages and information from any node or station to a designated service server. It offers great flexibility for users to create and transmit data packets in different scenarios.

For example, if a user wants to send a message from Car1 to CarServer, they can create an ICMP data packet with the customised message "Msg: Car1 is running with Speed 60 Km/hr" using the Scapy Python API within a Python script, as demonstrated in Figure 20. The user can then capture the data packet using the Wireshark tool, as depicted in Figure 21.

Similarly, if a user wants to send a message from Tra1 to RailServer, they can use Scapy to create and transmit a data packet with the customised message "Tra1 is running on Time", as shown in Figure 22. The data packet can be captured using the Wireshark tool at RailServer, as illustrated in Figure 23.

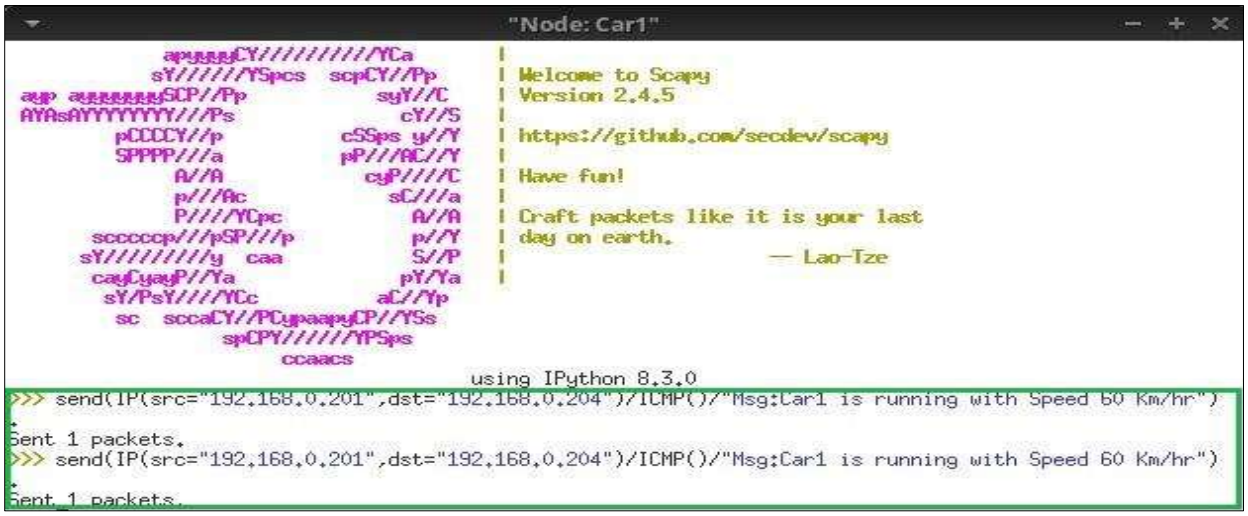

Figure 20: Scapy: Message Creation from Car1 to CarServer

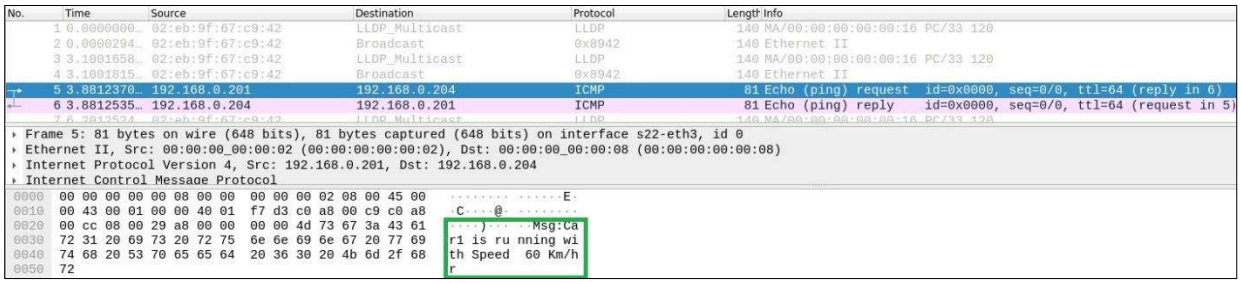

Figure 21: Wireshark: Scapy Packet with a Message

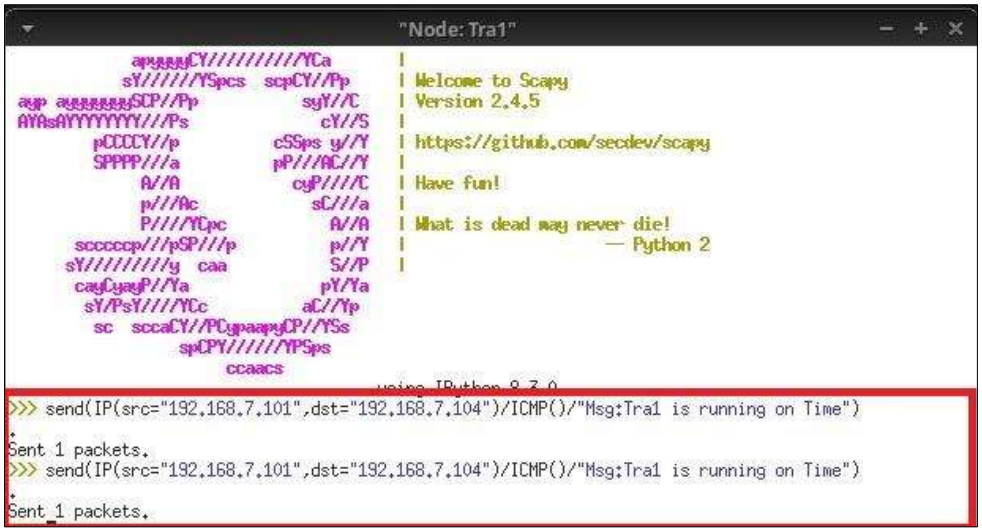

Figure 22: Scapy: Message Creation from Tra1 to RailServer

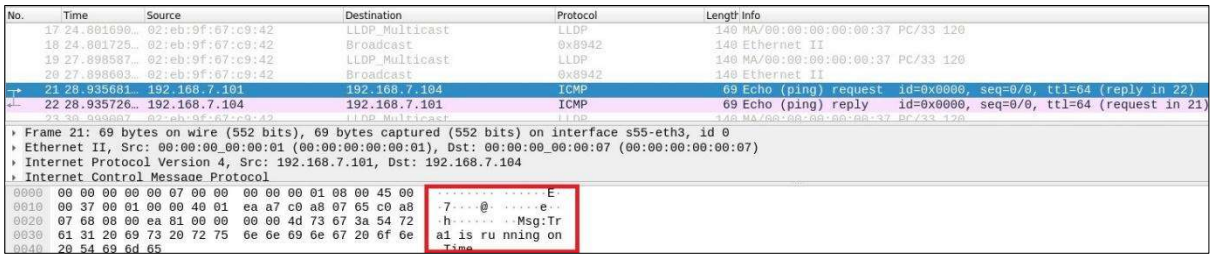

Figure 23: Wireshark: Scapy Packet with a Message

3.4.8 Video Transmission Test

A detailed description is given in the Journal paper [21] to demonstrate the video streaming test from one node to another node in Mininet-WiFi network emulator. Figure 24 depicts Train1 as the streaming node while RailServer is the receiving node. The use of VLC player enables the demonstration of video transmission between hosts in a railway and road coexistence scenario. This test provides valuable insights into the functionality and performance of the system.

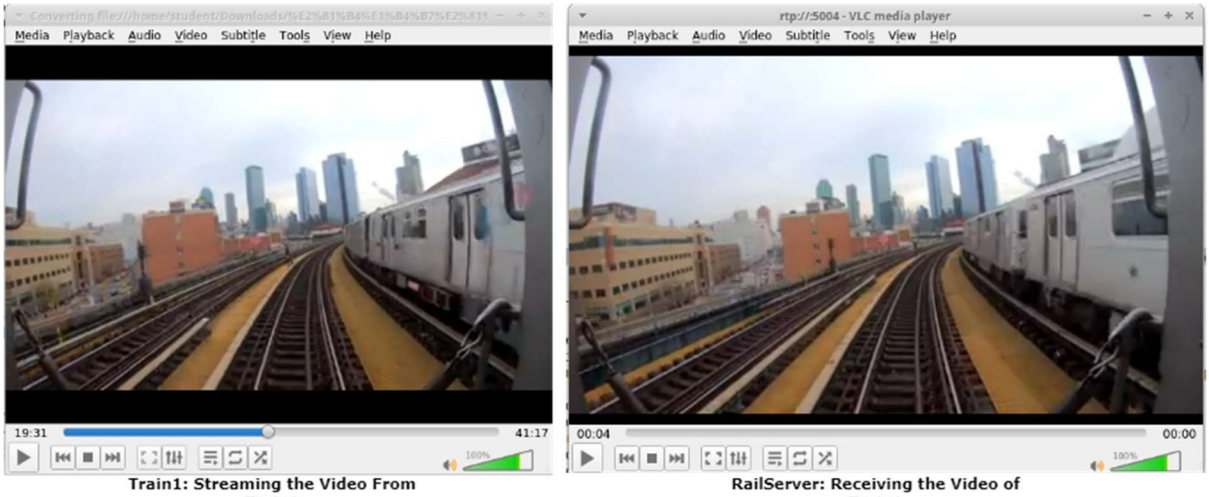

Figure 24: Video Data Transmission and Reception

3.5 Implementation and tests for Coexistence Scenario 2

As a reminder, **S1(5/6)4: Different Access Network and Different Core, Single Serving Technology, Track Perpendicular to Road:** In this considered scenario, the network parameters are similar to scenario S1(5/6)1, i.e., railways and roads have dedicated radio access networks with dedicated cores, but in this scenario railway tracks are perpendicular to roads.

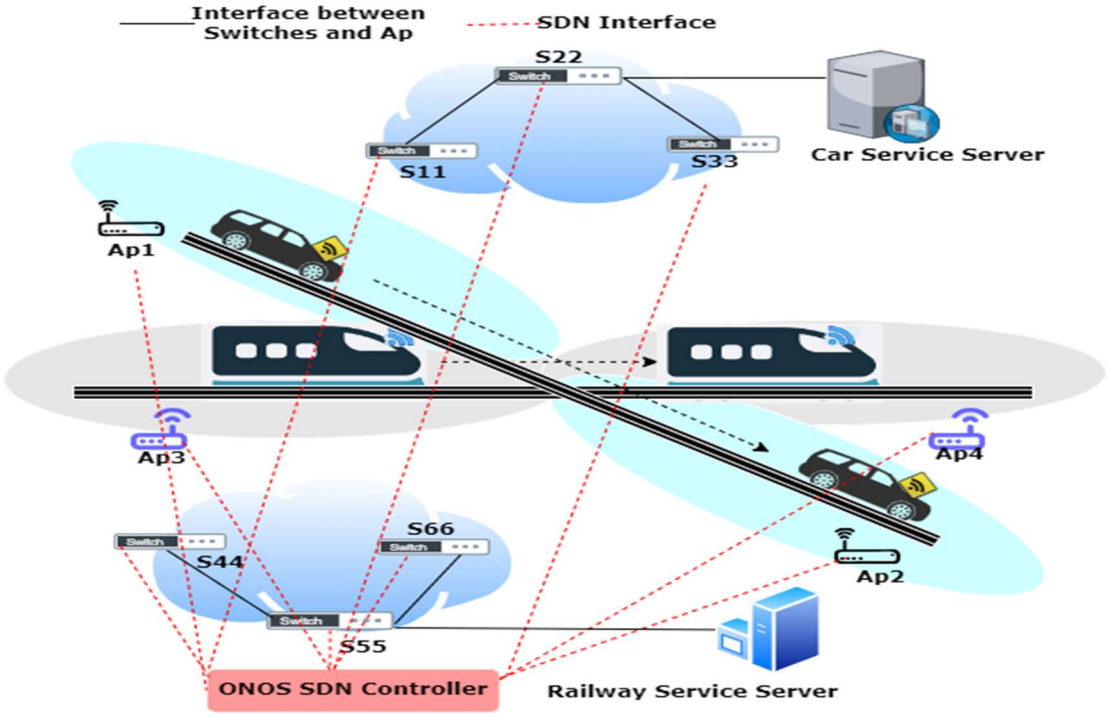

Figure 25: S1(5/6)4 Different Access Network & Different Core, Track Perpendicular to Road

3.5.1 Topology

A Mininet-WiFi network topology is generated using a Python script to allocate separate access networks for trains and cars. Both domains have their own core networks, with railways having perpendicular tracks to roads. Figure 25 illustrates the S1(5/6)4 scenario, where the ONOS SDN controller programmatically manages the forwarding elements of the topology. The network switches and access points are SDN-based devices that are managed and controlled through the OpenFlow protocol.

The S1(5/6)4 scenario's network topology, created with Mininet-WiFi, is depicted in Figure 26. Hosts Car1, Car2, and Car3 represent cars, while Tra1, Tra2, and Tra3 represent trains. Access points ap1 and ap2 are allocated for roads, with access point ap1 connected to switch S11 and access point ap2 to switch S33. Switch S22 is connected to both S11 and S33. The road service server, "CarServer," is connected to switch S22. Access points ap3 and ap4 are designated for railways, with access point ap3 connected to switch S44 and access point ap4 to switch S66. Switch S55 is connected to both S44 and S66. The railway service server, "RailServer," is connected to switch S55. Nodes Car1 and Tra1 are configured to move, to simulate moving cars and trains in this scenario. Figure 27, generated by Mininet-WiFi, displays the positions of nodes and access points before handover/moving.

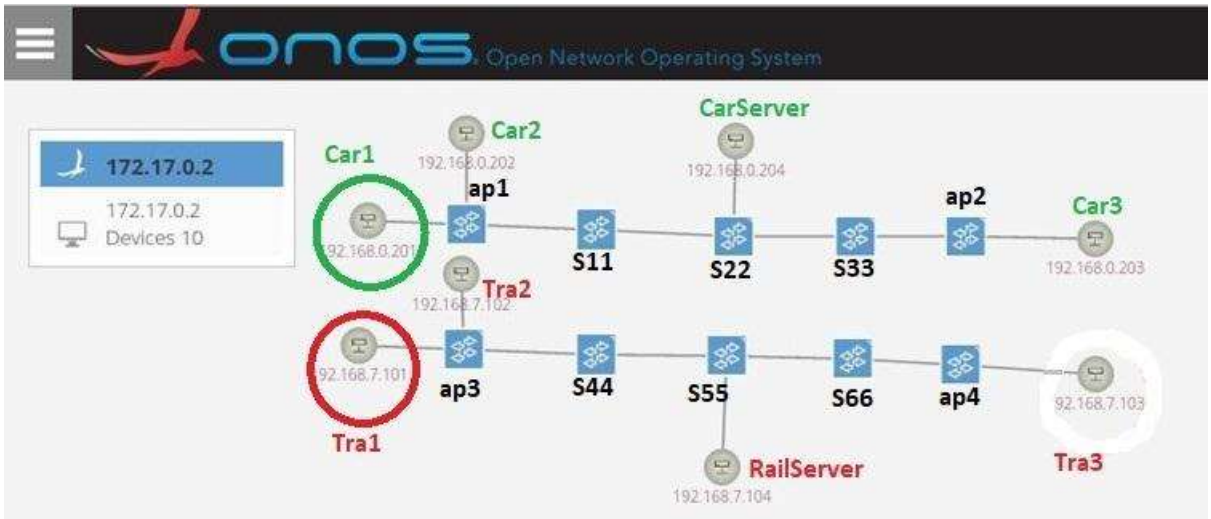

Figure 26: S1(5/6)4 Different Access Network & Different Core, Track Perpendicular to Road: ONOS Screenshot (Before Handover)

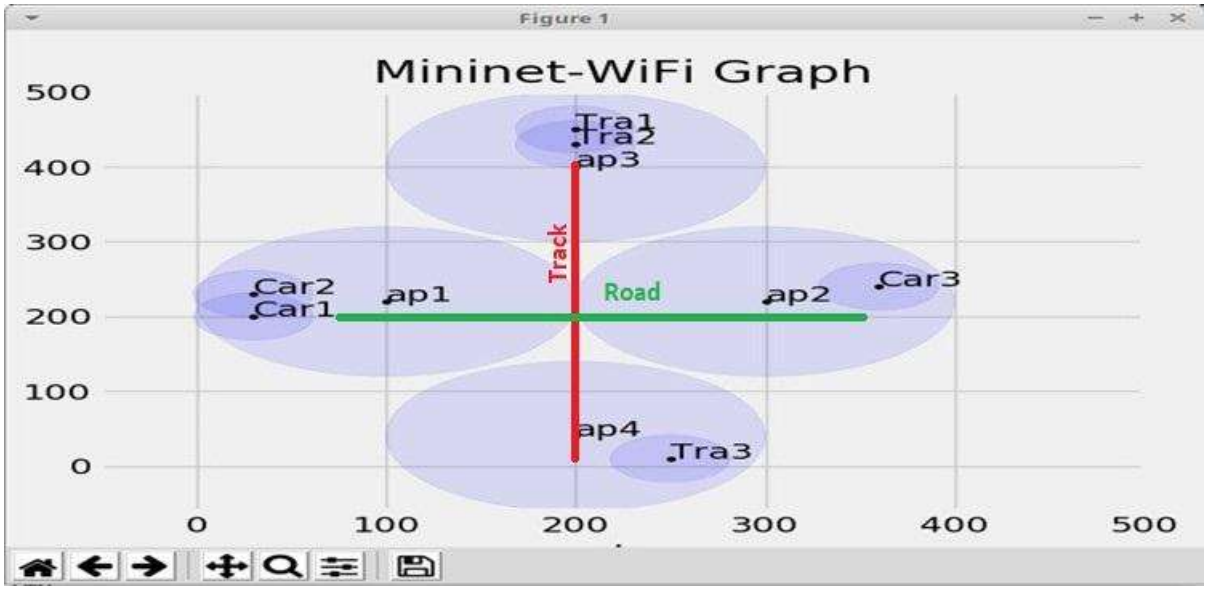

Figure 27: S1(5/6)4 Hosts and Access Points: Mininet-WiFi Graph (Before Handover)

To validate the considered tool to emulate the S1(5/6)4 coexistence scenario, the test results related to Handover/Moving, Reachability test, Data Traffic Differentiation test, UDP, TCP, Link Capacity, Latency, Jitter and sending a message using Scapy application are presented in the Appendix Section 8.1 of this documentation.

3.6 Implementation and tests for Coexistence Scenario 3

As a reminder, **S2(5/6)1: Different Access Network and Shared Core, Single Serving Technology, Track Parallel to Road:** In this scenario, railway and road domains have different radio access networks, and both domains share backhaul and core network infrastructure. In this considered scenario, railway tracks are perpendicular to roads.

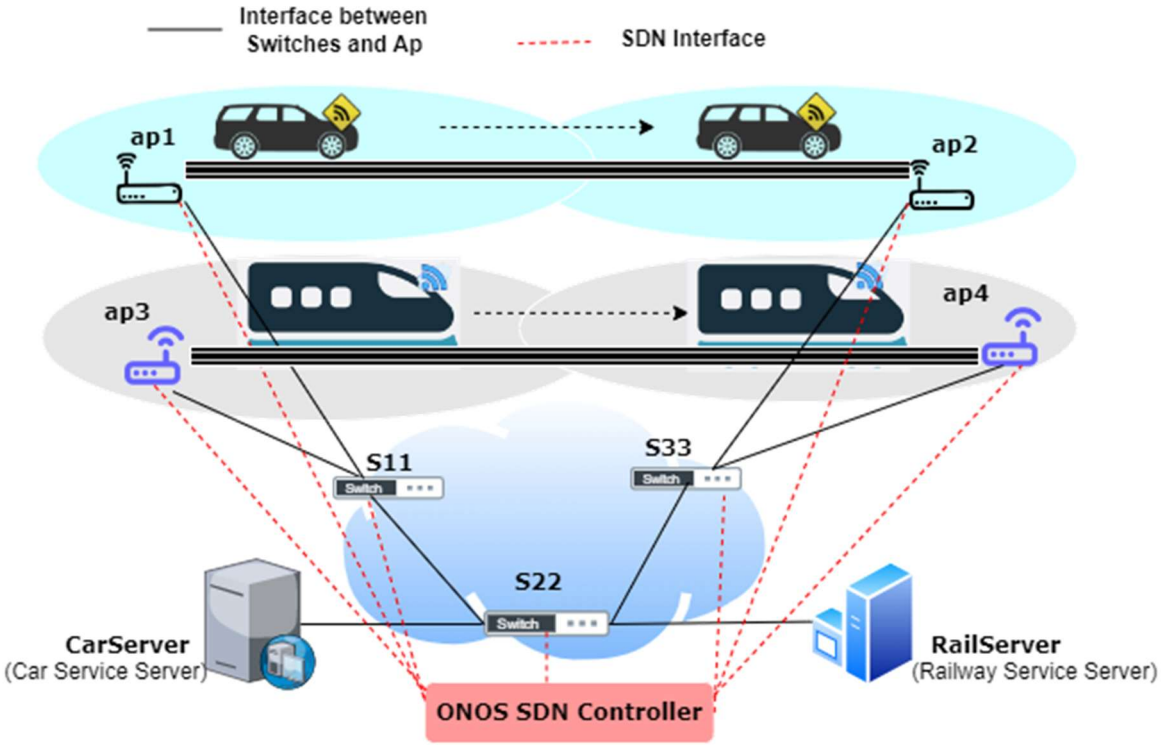

Figure 28: S2(5/6)1 Different Access Network & Shared Core, Track Parallel to Road

3.6.1 Topology

A Python script is used to create a network topology with Mininet-WiFi that separates trains and cars into different access networks, while the core network is shared between the two domains. Railways have parallel tracks to roads in this topology. Figure 28 illustrates the S2(5/6)1 scenario, where the forwarding elements of the topology are programmatically managed by an ONOS SDN controller. The network switches and access points are SDN-based devices that are operated and controlled via the OpenFlow protocol.

The S2(5/6)1 scenario's network topology, created with Mininet-WiFi, is shown in Figure 29. Hosts Car1, Car2, Car3, and Car4 represent cars, while Tra1, Tra2, Tra3, and Tra4 represent trains. Access points ap1 and ap2 are allocated for roads, while ap3 and ap4 are designated for railways. Access points ap1 and ap3 are connected to network switch S11, and ap2 and ap4 are linked to switch S33. Switch S22 is connected to both S11 and S33. The "RailServer" host is defined as the railways service server, and the "CarServer" is defined as the road service server; both servers are connected to switch S22. In this straightforward network topology, switch S22 serves as the core network switch, while S11 and S33 are the edge switches. Nodes Car1 and Tra1 are configured to move, to simulate moving cars and trains in this scenario. Figure 30, generated by Mininet-WiFi, displays the positions of nodes and access points before handover/moving.

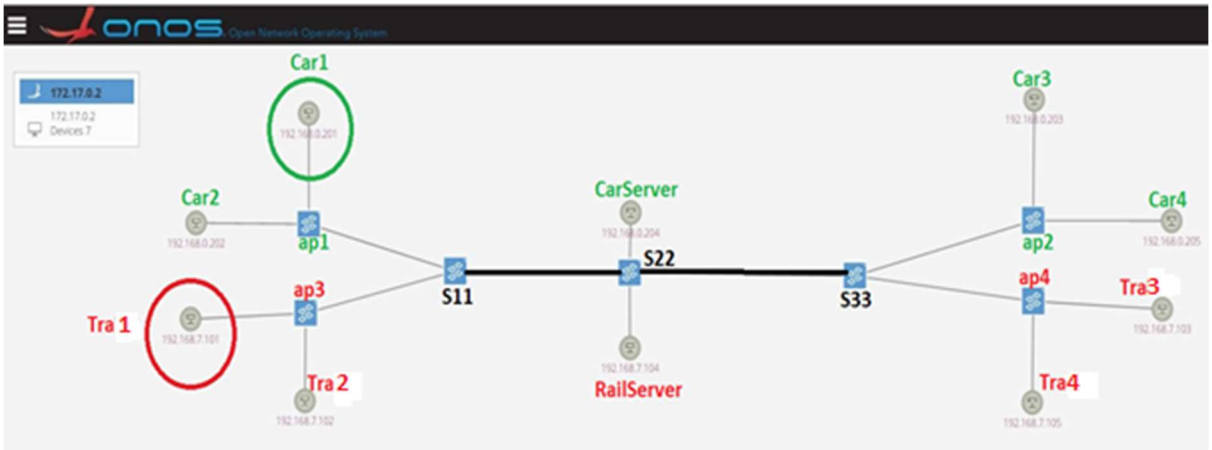

Figure 29: S2(5/6)1 Different Access Network & Shared Core, Track Parallel to Road Topology: ONOS Screenshot

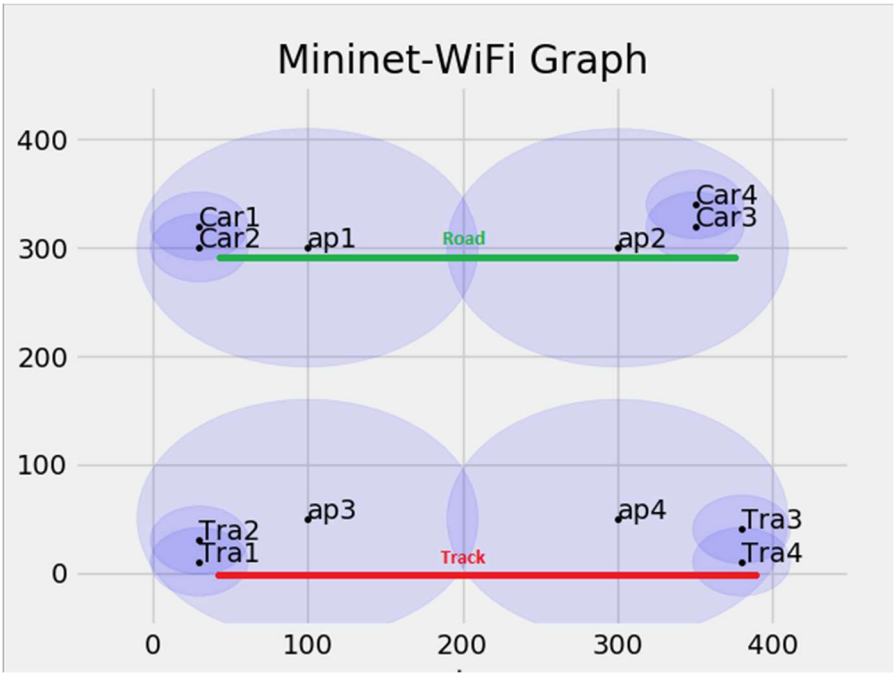

Figure 30**: S2(5/6)1 Hosts and Access Points: Mininet-WiFi Graph**

In order to validate the considered tool for emulating the S2(5/6)1 coexistence scenario, the test results related to Handover/Moving, Reachability test, Data Traffic Differentiation test, UDP, TCP, Link Capacity, Latency, Jitter and sending a message using Scapy application are presented in the Appendix Section 8.2 of this documentation.

3.7 Implementation and tests for Coexistence Scenario 4

As a reminder, **S4(5/6)1: Shared Access Network and Shared Core, Single Serving Technology, Track Parallel to Road:** In this scenario, railway and road domains share the radio access network along with backhaul and core network infrastructure. Railway tracks are parallel to roads.

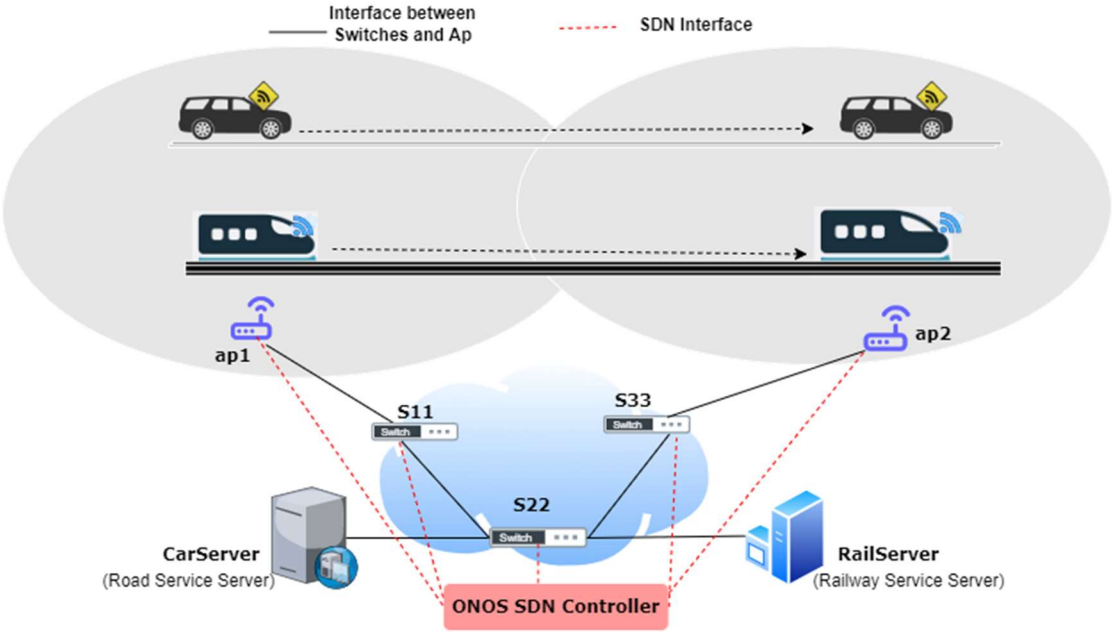

Figure 31: S4(5/6)1 Shared Access Network and Shared Core, Track Parallel to Road

3.7.1 Topology

A Python script is used to generate a network topology with Mininet-WiFi. In this topology, railways run parallel to roads and both domains share the access points and core. Figure 31 depicts the S4(5/6)1 scenario where an ONOS SDN controller programmatically manages the forwarding elements of the topology. The network switches and access points are SDN-based devices and are operated and controlled via the OpenFlow protocol.

Figure 32 shows the network topology for the S4(5/6)1 scenario, created with Mininet-WiFi. Cars are represented by hosts Car1 and Car2, while trains are represented by Tra1 and Tra2. Access points ap1 and ap2 are shared by both the domains. Access points ap1 is connected to network switch S11, while ap2 is connected to switch S33. Switch S22 is connected to both S11 and S33. The "RailServer" host is designated as the railways service server, and the "CarServer" is defined as the road service server, both connected to switch S22. Nodes Car1 and Tra1 are configured to move, simulating moving cars and trains in this scenario. Figure 33, generated by Mininet-WiFi, displays the positions of nodes and access points before moving/handover.

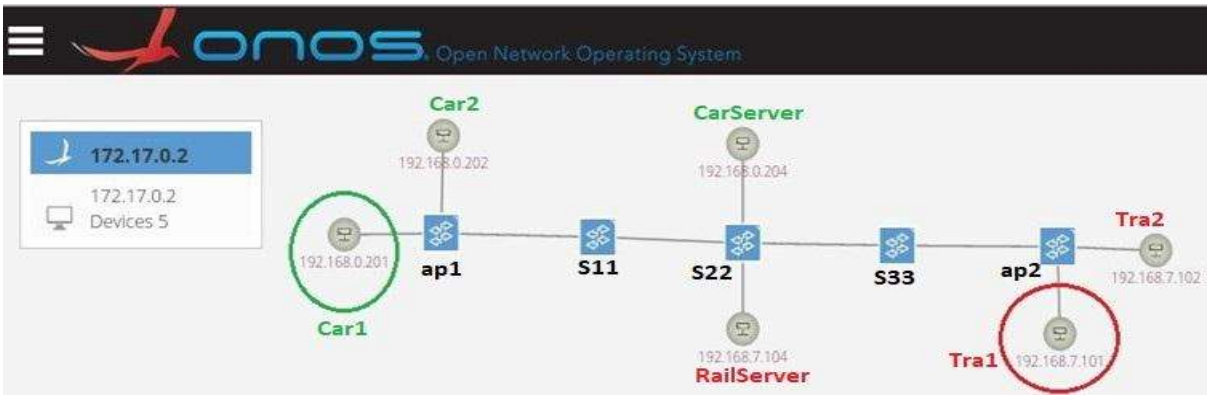

Figure 32: S4(5/6)1 Shared Access Network and Shared Core, Single, Track Parallel to Road: ONOS Screenshot (Before Handover)

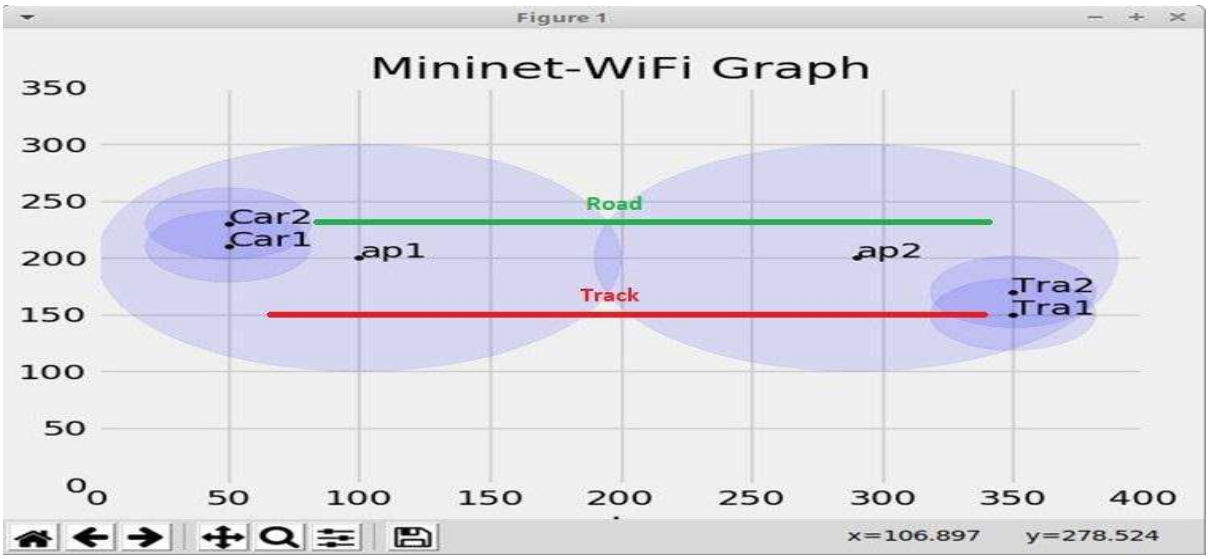

Figure 33: S4(5/6)1 Hosts and Access Points: Mininet-WiFi Graph (Before Handover)

In order to validate the considered tool for emulating the S4(5/6)1 coexistence scenario, the test results related to Handover/Moving, Reachability test, Data Traffic Differentiation test, UDP, TCP, Link Capacity, Latency, Jitter and sending a message using Scapy application are presented in the Appendix Section 8.3 of this documentation.

3.8 Implementation and tests for Coexistence Scenario 5

As a reminder, **S4(5/6)4: Shared Access Network and Shared Core, Single Serving Technology, Track Perpendicular to Road:** In this considered scenario, the network deployment infrastructures are similar to those of scenario S4(5/6)1, but in this case railway tracks are kept perpendicular to roads.

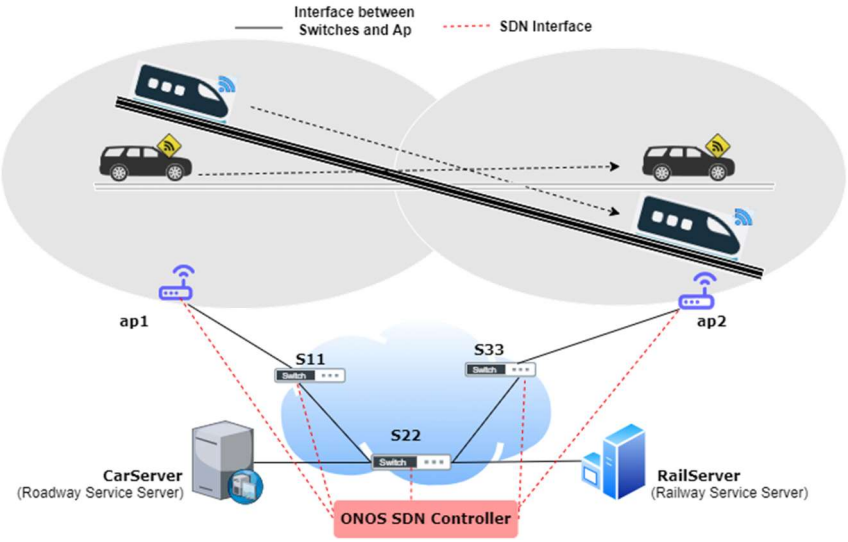

Figure 34: S4(5/6)4 Shared Access Network and Shared Core, Track Perpendicular to Road

3.8.1 Topology

A Mininet-WiFi Python script is used to create a network topology where railways run perpendicular to roads, and the access points and core network are shared between the two domains. Figure 34 illustrates the S4(5/6)4 scenario, where an ONOS SDN controller manages the forwarding elements programmatically. The network switches and access points are SDN-based devices that are operated and controlled through the OpenFlow protocol.

The Mininet-WiFi generated network topology for the S4(5/6)4 scenario is presented in Figure 35. Hosts Car1, Car2, Car3, Tra1, Tra2 and Tra3 represent cars and trains, respectively. Access points ap1 and ap2 are shared between the domains, with ap1 connected to network switch S11 and ap2 connected to switch S33. Switch S22 is linked to both S11 and S33. The "RailServer" host serves as the railways service server, while the "CarServer" is defined as the road service server. Both servers are connected to switch S22. Nodes Car1 and Tra1 are configured to move, simulating the movement of cars and trains in this scenario. Figure 36, generated by Mininet-WiFi, shows the nodes and access points' positions before moving/handover.

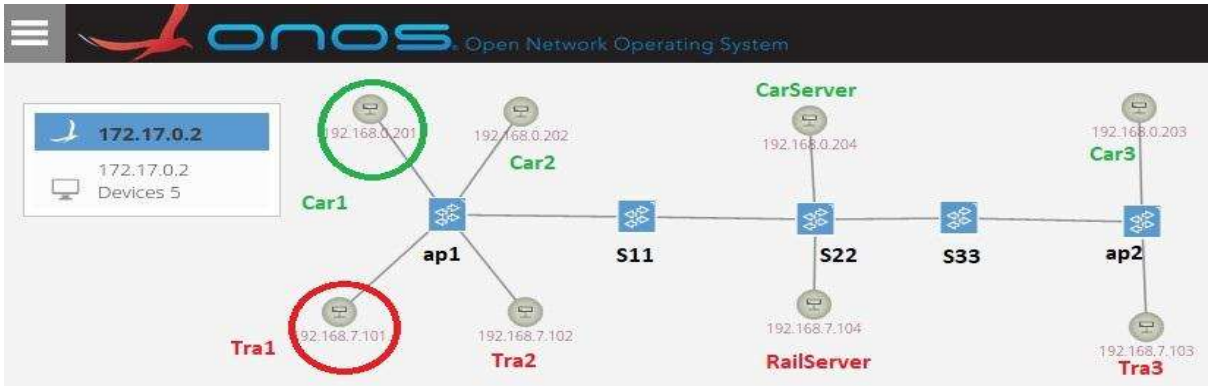

Figure 35: S4(5/6)4 Shared Access Network and Shared Core, Track Perpendicular to Road: ONOS Screenshot (Before Handover)

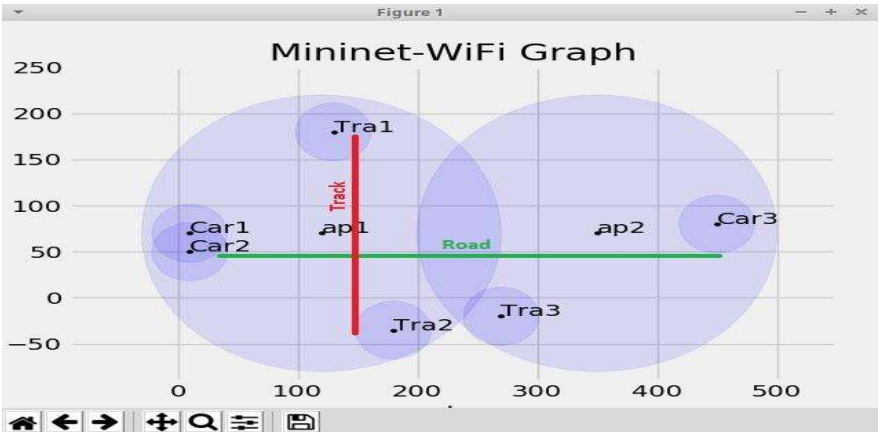

Figure 36: S4(5/6)4 Hosts and Access Points: Mininet-WiFi Graph (Before Handover)

In order to validate the considered tool for emulating the S2(5/6)1 coexistence scenario, the test results related to Handover/Moving, Reachability test, Data Traffic Differentiation test, UDP, TCP, Link Capacity, Latency, Jitter and sending a message using Scapy application are presented in the Appendix Section 8.4 of this documentation.

3.9 SUMO Integration

This section provides the information for the integration of SUMO with Mininet-WiFi, which provides a graphical representation of scenarios where railways and roads coexist. Figure 37 illustrates the steps involved in generating a SUMO map from OpenStreetMap and integrating it with Mininet-WiFi. This integration allows for a visual representation of the scenarios, enhancing the understanding of the system.

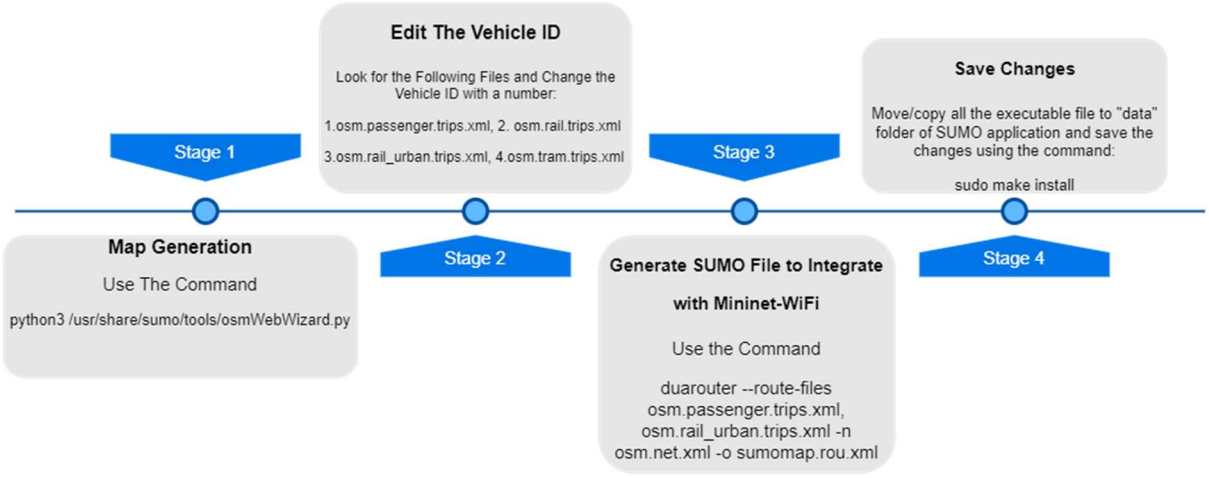

Figure 37: Steps to Integrate SUMO with Mininet-WiFi Topology [21]

To create a visual representation of the coexistence of railways and roads, users can design a customised map using Google Maps. To begin, download the desired Google Map file and run the command "python3 /usr/share/sumo/tools/osmWebWizard.py" from the desired folder. This will open the "OSM Web Wizard for SUMO" page, where users can select the "Generate Scenario" option to choose the location of the map by entering the city or place name or GPS coordinates. For the

purpose of demonstrating SUMO integration, we selected the location "Puente De Santiago", which has parallel tracks and roads. We selected the desired map area using the "Select Area" option and specified the parameters to generate vehicle traffic using the "Through Traffic Factor" parameter given at the SUMO web page. This parameter allows users to define the number of vehicles that will depart and arrive at the simulation boundary area. Figure 38 shows the selected map area and the chosen parameters for generating the vehicle traffic. By creating such visual representations, we can better understand the coexistence of railways and roads and their impact on traffic flow.

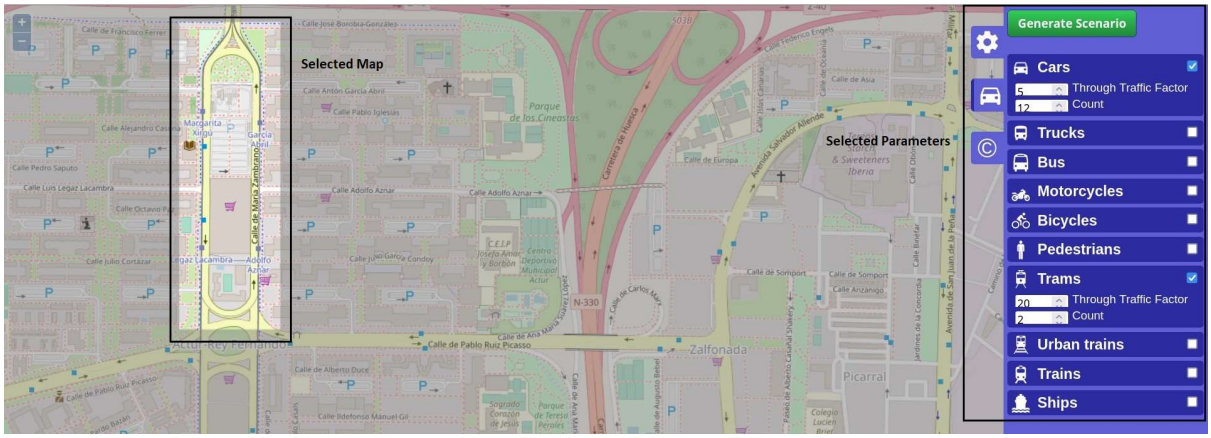

Figure 38: Selected location: Puente de Santiago

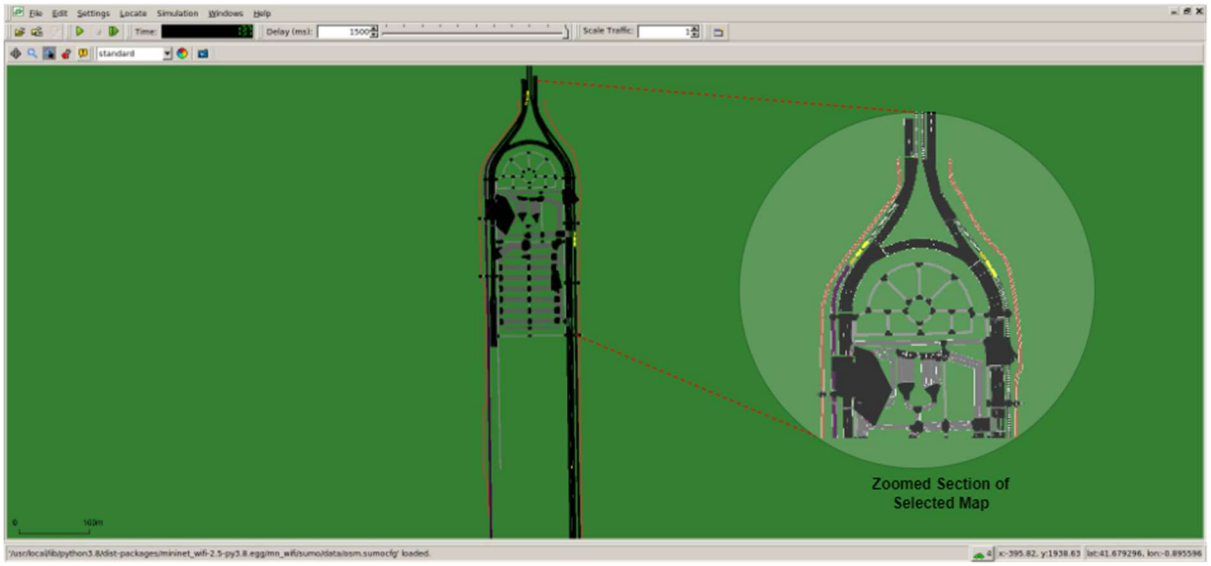

Figure 39: Simulation: Puente de Santiago

To generate the network topology integrated with SUMO maps, a Python script is used. The entire Python script “SUMO_Aug17.py” for network topology creation with SUMO map integration is available at [29]. After executing the “SUMO_Aug17.py” script, a network emulation will be created based on the SUMO map displayed in Figure 39. The location of the assigned access points and the movement of Cars and Trains are shown in Figure 40, where the access points are labelled as ap1 to ap17, Car2 is denoted as C2, and Train 77 is denoted as T77.

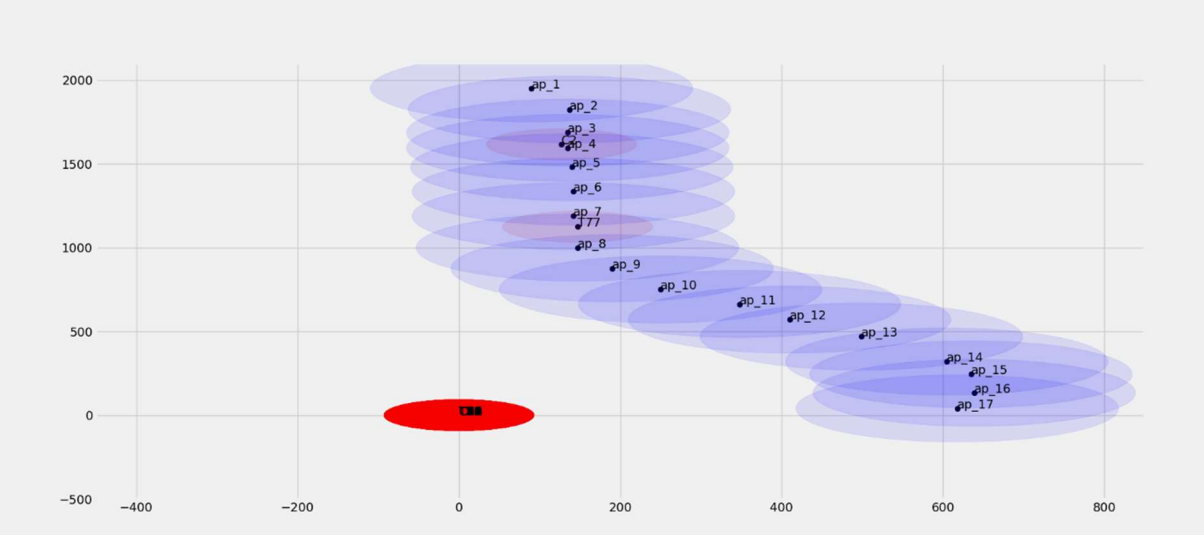

Figure 40: Movement of Train and Cars on SUMO Map

As a Car or Train moves out of the coverage range of one access point and enters the coverage range of another access point, it should automatically connect to the nearest access point. However, in some cases, the automatic connection to the nearest access point may not be established. If such an issue occurs, you can establish the connection manually by using the command: “<node name> iw dev <node name>-wlan0 connect <SSID name>”. Here, "<node name>" refers to the name of the node (i.e., Car or Train) that needs to connect, and "<SSID name>" refers to the SSID name of the access point it needs to connect to. For instance, if you want to connect a node named "Car1" to an access point with the SSID "ap1", you can use the following command: “Car1 iw dev Car1-wlan0 connect ap1”. This command will force the node to connect to the access point with the SSID "ap1" using the "wlan0" interface.

3.10 Critical Analysis of limitations

This first emulation environment has enabled us to implement the various coexistence scenarios identified, and to simulate the different railway applications. The implementation of these scenarios and applications can be reused and exported to any other environment. However, some elements of this platform could be improved to cover a wider range of scenarios:

- The radio access technology used in the network emulation is limited to Wi-Fi, and it does not include any 5G radio access or 5G architectural elements.
- The handover process between the Access Points in the network is based on an ad-hoc solution that relies on a self-built SDN (Software-Defined Networking) application at the network level. This means that standard 5G-compliant handover mechanisms are not supported.
- The implementation of new data processing architectures (Edge Computing) is not taken into account in this first experimental environment. However, these architectures could prove to be relevant to guarantee the smooth running of rail services.
- Additionally, the control of mobility and entities in the SUMO simulation tool is limited. Specifically, it is not possible to determine the initial location or trajectories of vehicles, which

makes it extremely challenging to generate iterations over a similar scenario/case under the same conditions. However, manual configuration can be done, but it requires a lot of effort and is prone to errors.

3.11 Conclusions

The emulator presented in this section has been used to implement the various coexistence scenarios and rail applications identified. The validation tests carried out in sections 3.4 to 3.9 demonstrate that users can develop different network topologies with nodes having moving capabilities and wireless access points for railway and road coexistence scenarios. By replicating Cars and Trains in a virtual space, moving hosts can simulate real-world scenarios. The developed ONOS SDN application can differentiate data traffic based on VLAN tags and handle handover scenarios. SUMO is a visualisation tool capable of representing the simulation of a scenario in a graphical way.

During the emulation of the Mininet-WiFi network with SUMO, there may be instances where nodes do not automatically connect to the nearest Wi-Fi access point when they move from one access point to another. This could be due to coverage range problems in the vehicle's path, which is determined solely by SUMO integration since the SUMO map is extracted from OpenStreetMap. To simulate real-world data traffic for railway and road coexistence scenarios, tools like iperf3, Scapy, and VLC player are considered. Iperf3 is used to demonstrate standard data communication by sending and receiving UDP and TCP packets between nodes, while Scapy is used to show messaging and critical data communication. To demonstrate the video transmission from one network node to another, VLC player is used. The MTR tool is used to measure network parameters such as latency, packet loss, and jitter.

The tools used in this study have shown great potential for emulating scenarios of coexistence between railway and road services, thereby providing a valuable framework for further exploration and analysis of this complex environment. This environment forms a high-performance basis on which bricks can be added to emulate a larger number of use cases. This is presented in the next section.

4 IMPROVED DEMONSTRATION SANDBOX

In this section, we present an emulation environment, called Emu5GNet, extending the solution proposed in the previous section. The objective of this platform was to offer an extended simulation/emulation environment allowing the implementation of more complex and realistic scenarios, including Edge Computing architectures, 5G E2E architectures in line with the work carried out in the other work packages of the 5GRAIL project, and cross-border scenarios.

4.1 Requirements Revision

We identified the following criteria for the development of this Emu5GNet platform:

1. The platform should enable the implementation of realistic end-to-end 5G networks. This implies the implementation of a 5G core network and the simulation/emulation of 5G access networks;
2. The platform must allow the deployment of real railway applications and must therefore include a platform offering the possibility to implement complex services (virtual platform);
3. The platform must allow the deployment of new data processing architectures that are increasingly considered today: Edge Computing architectures. The platform must therefore offer, in a realistic way, the possibility to deploy services at the edge of the network and to move these services;
4. The platform must allow the implementation of complex network architectures, including the possibility of managing the mobility of trains/vehicles on a small scale (inter-cell handover) and large scale (inter-core handover);
5. This platform must reproduce the operation of the 5G architecture developed in the framework of the 5GRAIL project. To do this, we have used the following document: “5GRAIL_20230320_R_PU_D1.1_RV4.0_UIC_Test_Plan”.

4.2 Sandbox Design and Initial Implementation

To implement a complete 5G network, while taking into account the limitations of the existing emulation platforms and the identified, we considered the integration of different tools:

- Mininet-WiFi - Containernet: Containernet [32] is a fork of Mininet-WiFi using Docker containers as hosts. It allows the emulation of Wi-Fi networks (e.g., 802.11ac and 802.11p) managed by an SDN controller in a flexible environment (containers). This platform can be linked to the SUMO simulator [26], an open-source software for microscopic traffic simulation (vehicles, trains, pedestrians, etc.) to implement complex scenarios;
- VIM-EMU: This platform [33] (SONATA project [34]) enables to locally prototype, deploy and evaluate network services. VIM-EMU, as Containernet, is based on Docker containers to enable quick and efficient deployment of services. This platform represents an efficient way to deploy and manage NFV functions and Edge servers using and deploying real orchestrators and services;

- UERANSIM: This tool [35] implements 5G User Equipment (UE) and 5G RAN (gNodeB) for both SA and NSA architectures. It represents an interesting building block for the implementation of a 5G communication architecture including the wireless segment. It is designed to support a large number of simultaneous communications and to evolve with the advances of 5G;
- Open5GS: This tool provides a C-language implementation for 5G Core [36]. It implements all 5G Core functions and can be used to deploy a complete 5G communication architecture. It is designed to be interconnected with 5G RAN platforms including UERANSIM.

An integration process was required to enable the implementation of Emu5GNet:

- VIM-EMU and Containernet - Mininet-WiFi compatibility: The VIM-EMU data centres are switches that have been modified to behave like edge servers (including CPU and storage models). Containernet was not designed to handle such nodes and was unable to recognise them. It was, therefore, necessary to integrate these two environments to enable the VIM-EMU data centres to be used in the Containernet - Mininet-WiFi platform;
- Open5GS and UERANSIM integration in Mininet-WiFi: These tools have not been designed to be integrated into a larger emulator. An integration and dockerisation work of the 5G Core (Open5GS) and the 5G RAN (UERANSIM) was required to deploy end-to-end 5G communications in Mininet-WiFi - Containernet;
- VIM-EMU improvement: VIM-EMU is designed to deploy network functions and edge services in wired networks. The extension of VIM-EMU was necessary to allow the placement and migration of edge services for wireless 5G networks. This work involved adding new Application Programming Interfaces (APIs) to VIM-EMU and specifying new information about deployed services/functions: CPU, memory, etc.

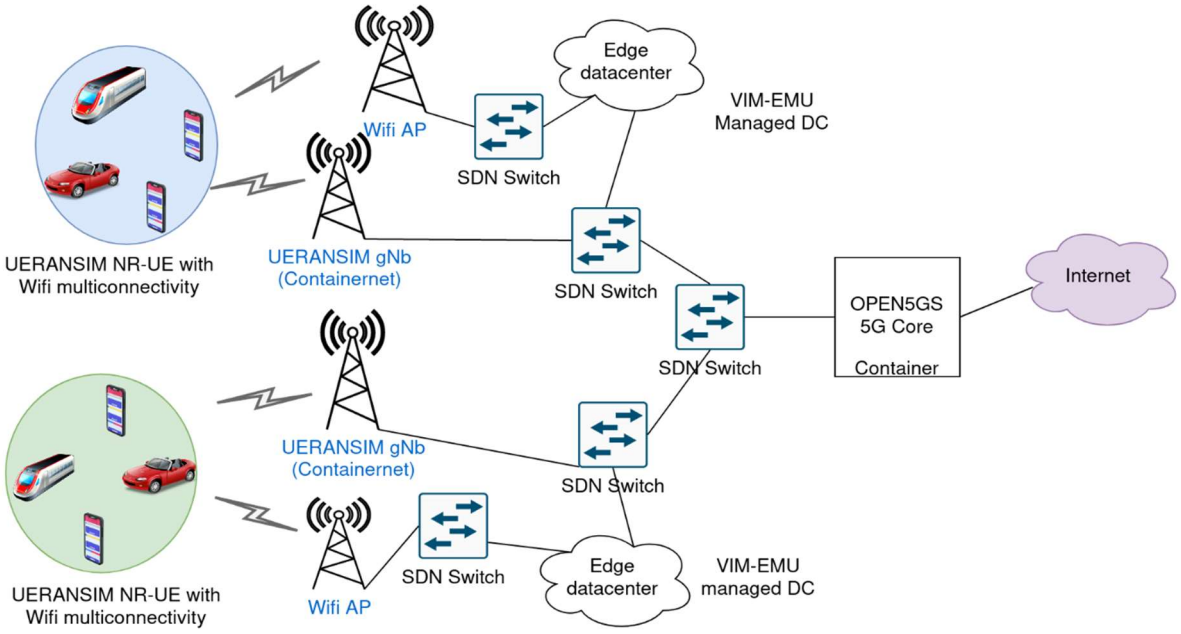

Figure 41: Overview of the Emu5GNet Architecture

Emu5GNet architecture (cf. Figure 41) is designed to deploy different types of nodes/elements:

- 5G UE and gNodeB: 5G UEs and 5G gNodeB can be instantiated in Emu5GNet using UERANSIM. These 5G UEs and gNB can be both mobile and fixed. UEs are implemented as specific Mininet-WiFi hosts and gNB as specific docker containers;
- 5G Core: A core 5G network can be deployed in Emu5GNEt using Open5GS. This 5G Core, implemented as a docker container, manages all the Emu5GNet UEs and gNodeB. In more complex scenarios the deployment of different network cores could easily be addressed;
- Wi-Fi Access Points and hosts: With Mininet-WiFi, Wi-Fi access points and Wi-Fi hosts can easily be deployed in Emu5GNet. These nodes correspond to existing Mininet-WiFi - Containernet nodes and it was not necessary to modify them to enable their implementation;
- Edge Data Centres and Orchestrators: The deployment of edge data centres and orchestrators, as Docker containers, is possible in Emu5GNet using VIM-EMU. The edge orchestrator can manage the migration of edge services between the available servers (cf. section 3.A). All nodes (Wi-Fi, 5G NR) can connect to these servers.

For Wi-Fi communications, the Emu5GNet architecture is managed by a central SDN controller (compatibility with the first platform developed within the framework of this project). Wi-Fi hosts and 5G UEs can be integrated into the same Mininet host (multi-RATs device). Wi-Fi Access Points and gNBs can be connected to the same edge server. Moreover, all the nodes are connected to the Internet, increasing the number of deployable applications on these devices (e.g., streaming). Finally, all devices can be mobile (SUMO) to implement enhanced scenarios.

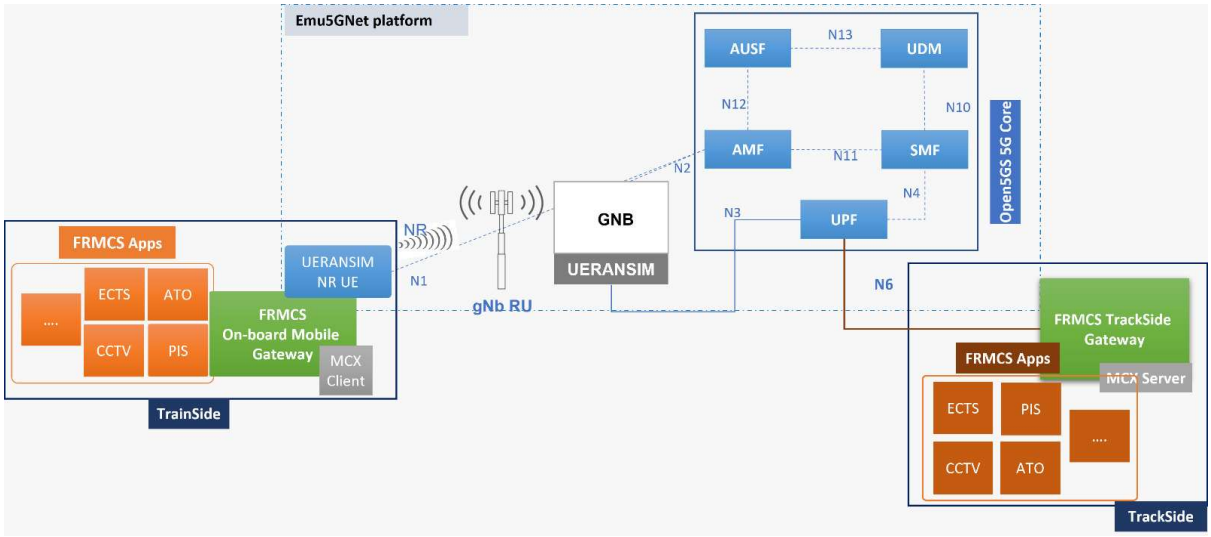

Figure 42: Emu5GNet Architecture in the 5GRAIL Framework

The Figure 42 presents a higher level view of the architecture implemented in the Emu5GNet platform. In particular, it highlights the elements developed at the TrackSide, TrainSide and 5G network core. A comparison with the architecture presented in the document 5GRAIL_20230320_R_PU_D1.1_RV4.0_UIC_Test_Plan allowed us to confirm the compatibility of the proposed architecture with the architecture considered more globally in the framework of the 5GRAIL project. This allows us to validate the relevance of this platform.

4.3 Edge Computing Implementation and Initial Evaluation

4.3.1 Edge Computing Implementation

The deployment of Edge Servers to enable the evaluation of current data processing architectures was another requirement identified for the Emu5GNet platform. The implementation relies on the tools integrated in the Emu5GNet platform (cf. Section 4.1): Mininet-WiFi, VIM-EMU, UERANSIM, Open5GS. In particular, VIM-EMU was used to implement:

- Computing nodes: Integrating the VIM-EMU platform, Emu5GNet can be used to deploy realistic computing servers. These servers are implemented as Docker Containers that can integrate real railway services. These servers can be configured to act both as 1) Cloud Computing Servers and 2) Edge Computing Servers. To do so, latency, reliability and computing capabilities parameters can be updated at any point in time;
- Computing nodes orchestrator: Managing a set of servers necessarily implies the implementation of an orchestration solution. To do so, Emu5GNet includes scripts and APIs that allow jointly managing all the deployed servers: start, stop, move a service, etc. This allows users to quickly master the platform for the deployment of new solutions to define optimised edge solutions for railway networks.

Figure 43 offers a basic view of the deployment of Edge and Cloud Servers and Edge Servers orchestration.

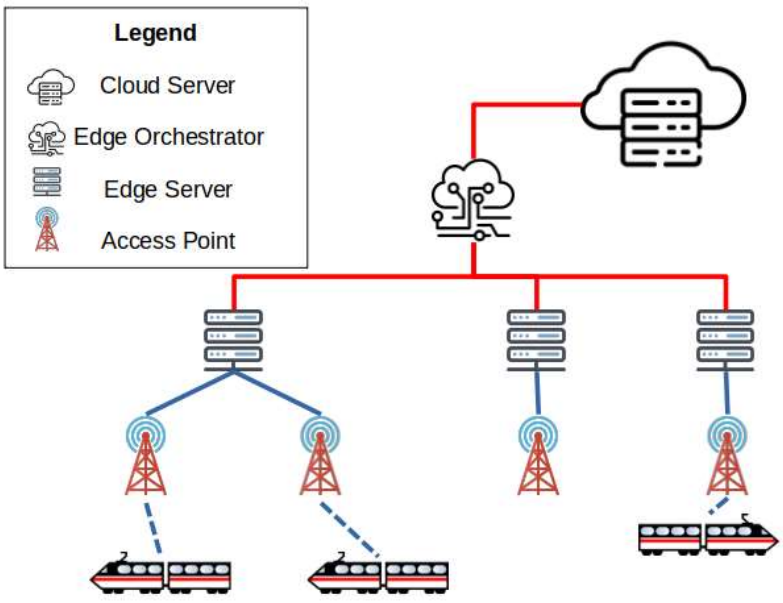

Figure 43: Overview of the Edge Computing Paradigm

Based on this tool and the implementation of different features, Emu5GNet includes the possibility to configure a broad set of parameters for Edge Computing Architecture's Implementation and Evaluation:

- Edge nodes deployment: The idea of the platform is to allow the evaluation of an extensive panel of Edge Computing architectures. Therefore, different types of nodes (mobile/fixed) can be considered for the deployment of Edge servers. For example, edge nodes could be deployed at the base station level as well as at the train level. Note that Cloud servers can also be deployed;
- Edge nodes capabilities: The deployed platform is designed to allow dynamic management of all parameters related to the deployed edge servers: bandwidth allocated to a given server, computing power, available storage space, etc. This allows the implementation of heterogeneous processing architectures that can take into account the various possible deployments: at the base station level, at the terminal level, in the core network, etc.
- Edge nodes orchestration: Different orchestrators can be used concurrently to manage the edge servers deployed in Emu5GNet. This can allow not only the definition of different domains but also the implementation of concurrent strategies for the management of edge servers and the resources available within these servers;
- Network performance level: The network performance level, both wired and wireless, can have an impact on the capacity of edge servers. Indeed, high latency could lead to the inability of some edge servers to handle critical rail services. Emu5GNet allows setting the network performance level through different approaches: 1) by deploying different radio access networks (Wi-Fi or Cellular) and core networks (SDN or 5G) and 2) by directly setting the performance level of the communication links: packet loss, latency, bandwidth, etc.

4.3.2 Initial Evaluation

To demonstrate the potential of the Emu5GNet platform, we considered a simple use case (cf. Figure 44): on a 3km^2 map, we considered a variable number of cars (random trip) and a dosen of trains generating constant volumes of data (iperf command) and connected to 17 access points (5G + Wi-Fi 802.11n) uniformly distributed on the map. Three edge data centres are distributed on the map. Data generated by vehicles is automatically transmitted to the nearest data centre. This scenario is presented in more detail on the Emu5GNet Github Page [37].

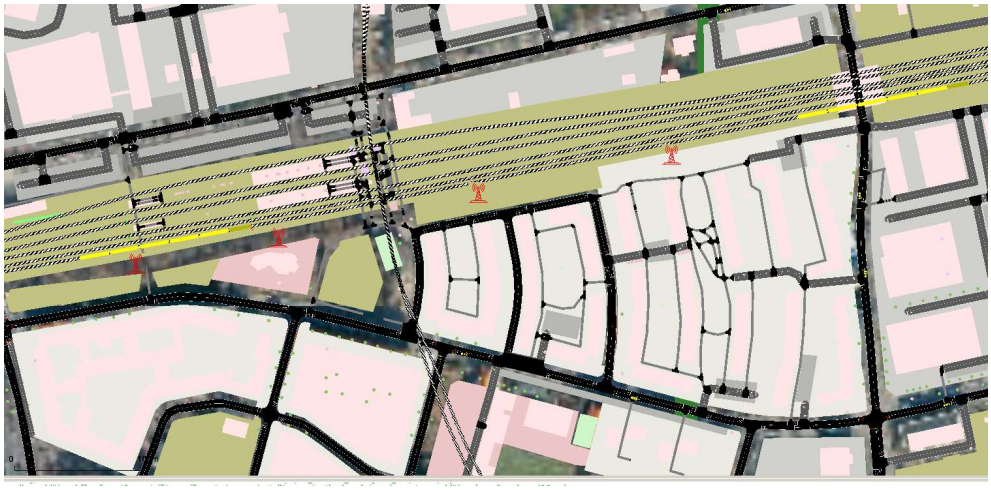

Figure 44: SUMO-based emulation

More precisely, we considered the following scenario: each train simultaneously launches a variable number of services considered as critical applications (corresponding to iperf commands) and these services link to an Edge or Cloud server (acting as an iperf server and that could host, more broadly, any kind of server/application). The experiment lasted 60 seconds and a maximum of 135 simultaneous services were counted. The latency associated with the Cloud server is calculated based on the average latency data measured for Google Cloud servers in France. Different articles, such as [38], provide useful pieces of information regarding Cloud servers' latency measurement.

In this evaluation, we aimed to demonstrate that Emu5GNet can be used for: 1) Radio Access Technologies comparison, 2) the definition of optimal architectures, 2) service placement and 3) service migration. We also wanted to highlight the fact that the platform can provide different types of results related to: 1) latency, 2) service placement failure rate or 3) percentage of services hosted on one of the deployed edge servers.

Regarding RATs, we compared, for end devices, the level of performance of the two available RATs (802.11n and 5G NR). To do so, we assessed the maximum throughput allowed by each of these technologies and the associated latency (Round Trip Time) in the context of a data transmission from the terminal equipment (trains, cars) to the nearest edge servers. The obtained results (cf. Figure 45) shows that, both in terms of latency (RTT 10x lower on average) and maximum throughput (6x higher on average), 5G NR technology performs better than 802.11n. This use case demonstrates that the proposed environment easily enables the simultaneous evaluation of the performance of different RATs.

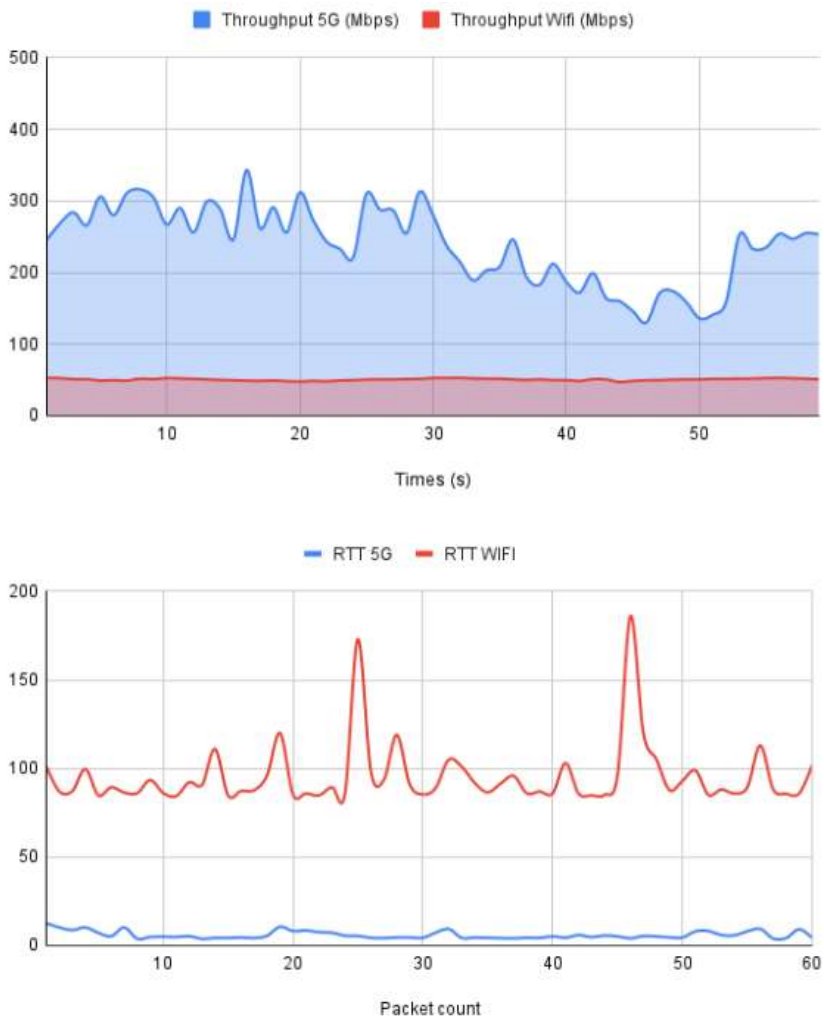

Figure 45: Evaluation for multi-RATS

Regarding Edge Computing, three different curves are presented in Figure 46 :

- On the top left, based on the average latency times measured for Cloud servers, the idea was to highlight the potential benefits of deploying Edge servers in terms of latency. Over the duration of the experiment (60s), this curve compares the latency required to transmit data from a train to a) a Cloud server and b) an Edge server. Such measurements could also be used to define optimal, multi-tier Edge Computing architectures to optimise both the cost and performance level offered to rail services;
- On the top right, the idea was to demonstrate that the Emu5GNet platform can be relevant for the definition of placement strategies for Edge services. We have therefore basically compared two placement strategies, one uniform (services are distributed fairly by the orchestrator among the different servers) and the other non-uniform (non-equitable distribution). The curve displayed here allows us to see the percentage of services currently running on one of the Edge servers compared to the total number of services currently running in the infrastructure. Related to these placement strategies, many other curves could be obtained: energy overhead generated by a non-optimal placement strategy, estimation of the computational latency associated with each edge computing server, etc.

- In the bottom centre, the idea was to demonstrate that the Emu5GNet platform can also be used for the evaluation of Edge service migration policies. To do so, we compared two simple approaches: a) a first approach in which services are moved to the nearest edge server based on train mobility and b) a second approach in which deployed services are maintained end-to-end on the first server attached to the train. We have defined a maximum communication delay between the train and the edge server used (10s) and considered that the service placement is a failure when the latency exceeds this limit. Indeed, for a critical service, exceeding the defined maximum latency might not be acceptable. Thus, this curve allows us to see that not migrating edge services leads necessarily to an increased latency and a higher percentage of placement failures. Emu5GNet could therefore be used to define an optimal service migration strategy. In this context, many parameters could be evaluated such as the time needed for the migration of a service, the additional cost (computation, communication) caused by the migration strategy, etc.

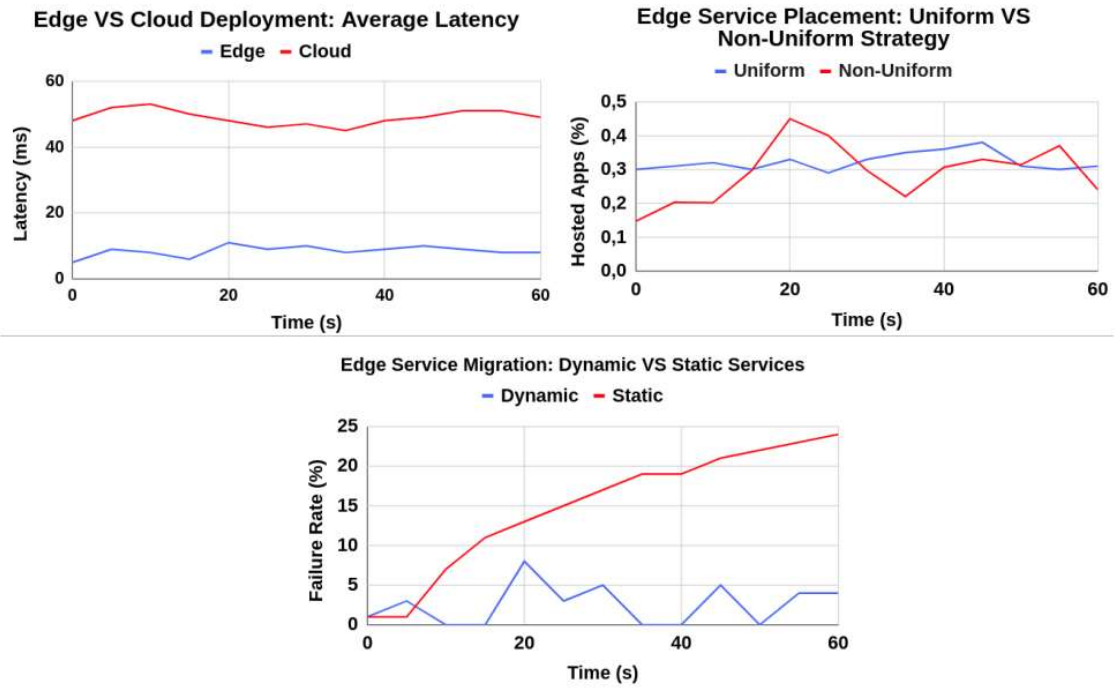

Figure 46: Evaluation for Edge Computing

Thus, this simple experiment shows that the Emu5GNet platform can be used for a wide range of implementations. Various RATs, server/service placement and migration strategies could be implemented and compared with this platform. The results obtained could allow us to identify appropriate strategies for specific applications and optimisation objectives: cost, performance, energy, etc. The potential applications of Emu5GNet are therefore numerous.

4.4 Cross-border Scenario Implementation

The last element we wanted to implement in the Emu5GNet platform is the possibility to emulate cross-border management and, in particular, roaming between two 5G cores, based on the E2E

architecture defined in other WP ("5GRAIL_20230320_R_PU_D1.1_RV4.0_UIC_Test_Plan"). Indeed, this feature seems to offer the possibility to simulate and validate a large panel of scenarios that could be relevant in the 5GRAIL project.

Figure 47 presents a simplified vision for managing this cross-border scenario. Two 5G cores (Open5GS - part of Emu5GNet) are deployed and each manages 5G gNBs (EURANSIM - part of Emu5GNet). In each of the 5G cores the main functions are deployed (AUSF, UDM, AMF, SMS, UFP) and the UPF function of each core is used to transmit data to the FRMCS TrackSide Gateway.

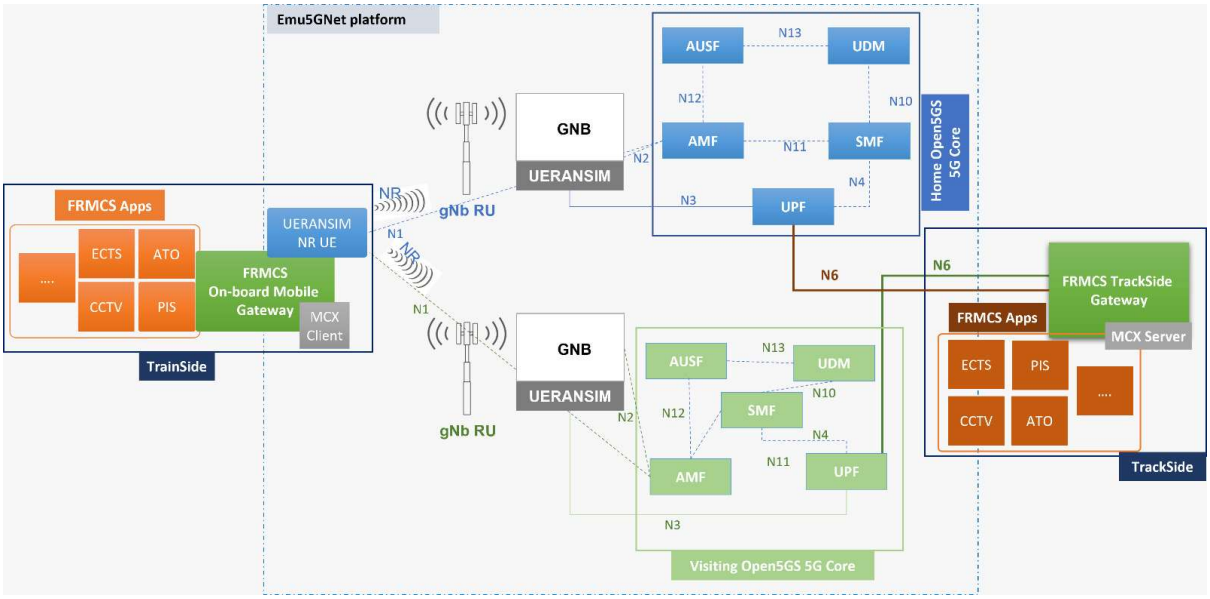

Figure 47: Roaming Implementation in the 5GRAIL Framework

The solution chosen in the framework of this platform for the implementation of roaming is the home routing roaming approach [39]. This approach has the advantage of being a well-known solution and can be implemented in the considered 5G multicore environment. It offers a direct connection between the train and the FRMCS application (via the UPF) via a traffic redirection to the home network, which guarantees low end-to-end delays. The scenario chosen in the demonstration presented in Figure 48 corresponds to a train moving at a border between France and Belgium (the border is represented on the figure by a vertical red line). gNB base stations are deployed on both sides of the border. They are managed by two separate network cores interconnected by a home routing roaming approach.

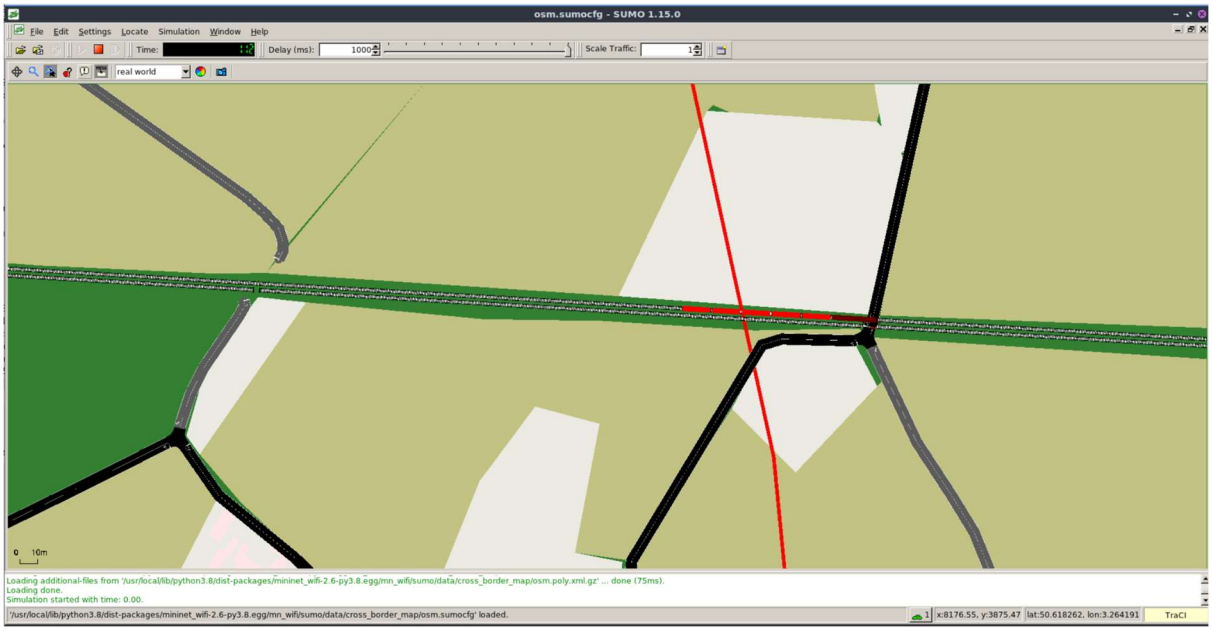

Figure 48: Implemented scenario with Emu5GNet and SUMO

Figure 49 shows the latency that was measured during the 60s experiment at the train level (NR-UE interface). The deployed service corresponds to a video streaming service at the train level and uploaded to the TrackSide server. We can see on this figure that the average transmission latency between the UE and the server is about 10ms with a peak at the time of roaming: 1s. This experimentation was carried out under specific conditions (network cores deployed on the same hardware machine within virtual machines), but in a customisable environment that could include 1) variable quality of service parameters that could evolve over time, thus reproducing the degradation of the communication channel, and 2) the implementation of different roaming solutions beyond this project implementation (home routing roaming). This environment could therefore be interesting for the implementation of new roaming solutions.

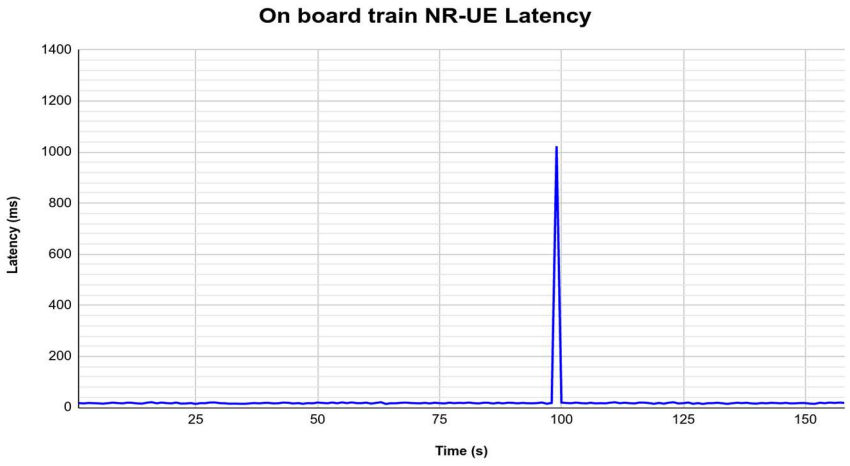

Figure 49: Latency evaluation in roaming scenario

4.5 Conclusions

In this section, a new simulation/emulation platform was presented: Emu5GNet. This platform aims at offering a more realistic simulation environment thanks to the implementation of 1) a 5G network core (Open5GS), 2) a 5G RAN (UERANSIM) based on a virtualised environment (Docker Containers) and 3) the implementation of the various FRMCS elements (TrainSide and TrackSide).

This platform also aims to cover a wider range of applications than the first environment implemented and presented in Section 3 by including the possibility to simulate complex data processing architectures (Edge Computing architectures with VIM-EMU) allowing the deployment of complex applications with significant constraints (latency, bandwidth). It also allows the simulation of cross-border scenarios thanks to the implementation of a roaming solution (home routing roaming). Finally, different access technologies can be considered and emulated to interconnect UEs (trains, cars) to the network.

The implemented evaluations allowed to demonstrate the relevance of this platform for the different scenarios identified (cross-border, edge computing, multi-RATs, multi-UEs). They also demonstrated that complex solutions for managing these scenarios could be implemented and evaluated in a realistic environment using this open source and documented tool: edge servers deployment, RATs selection, services migrations, etc. This tool therefore opens up the prospect of many applications.

5 CROSS DOMAIN SERVICE IDEA AND DEMONSTRATION

5.1 Introduction

One specific assumption in the initial work on coexistence between roads and railways in WP6 was that applications were specific to each domain. As a result, it was assumed that there were no common services, even in coexistence scenarios.

As our work-package progressed, we discovered a specific case where having a common aspect for a service could be of great interest. This case was related to a level-crossing scenario where the railway tracks intersected with road lanes perpendicularly. During this scenario, when a train was approaching the level crossing, the flow of cars on the perpendicularly traversing road lanes was stopped by a movable barrier to ensure the safe passage of the train. However, dangerous situations frequently arose when a car was stuck across the railway track in the level-crossing area. To mitigate this issue, we proposed the implementation of an "emergency" messaging system that could communicate with both a Car-Emergency service and a Train-Emergency system. In doing so, any potential collision between the two could be avoided by notifying the approaching train of the situation. This approach could significantly enhance the safety of both the railway and road users in such scenarios.

In the current railway signalling system (ERTMS [11]), a specific voice service, known as the Railway Emergency Call (REC), has been enabled by the supporting telecommunication network (GSM-R [12]). A revised version of the REC, called "enhanced Railway Emergency Call" (eREC), has also been introduced [13]. This domain-specific communication service is primarily related to emergency situations. The eREC is based on the Voice Group Call Service (VGCS) with additional pre-emption features. When a user initiates an eREC call, the request is received by the serving GSM-R network, which creates a group call (VGCS) to all entities in the same area, as well as to the area-dispatcher/controller. The call is established even if no resources are initially available for it. The network disconnects any other, lower-priority call to free resources and prioritises the eREC communication. Once each endpoint accepts the eREC, the call is open for all participants, and everyone can receive the contents of the call and participate in it. Strict rules ensure that only participants with relevant information to the eREC can contribute to it. The eREC is an enhanced version of the REC as it allows the definition of the REC area differently than the cell of the REC initiator. This feature enhances safety in areas where cells overlap, minimizing the risk of potential accidents.

In the roadway environment, messages are exchanged between vehicles and the roadside infrastructure to enable communication. The European Telecommunications Standards Institute (ETSI) has defined two main message types: Cooperative Awareness Message (CAM) and Decentralized Environmental Notification Message (DENM) [14]. CAM messages are periodic messages used to transmit vehicle status information such as location, speed, and identifier. DENM messages are asynchronous messages used for the transmission of specific information, such as emergency information in the vehicle environment. They can be used to indicate obstacles on the road, lane changes, or sudden slowdowns. In addition to direct communications between vehicles (Vehicle-to-Vehicle) that enable quick reception of DENM messages by terminals located in the same area, roadside infrastructure is also used to transmit these messages to a wider area. This is achieved through the GeoNet protocol and Geo-Multicasting, which determine the geographical area to which the message should be transmitted based on its type and optimize its distribution to ensure that it is received by all relevant entities.

Aim of this experiment was to showcase the common emergency services for railways and roads coexistence scenario through the use of a level crossing situation. The selected scenario involved railway tracks perpendicular to roads, as illustrated in Figure 50. For this purpose, a Python script was written in Mininet-WiFi. The network topology was created to emulate the level-crossing scenario using the Mininet-WiFi emulator. Open Network Operating System (ONOS) SDN controller was used to programmatically control the forwarding element of the network topology. Access points and switches were SDN-based devices, which were controlled via the OpenFlow protocol. Rail Emergency Server (RailEmer) and Car Emergency Server (CarEmer) were defined to handle the exchange of emergency messages between the server and nodes. In contrast, RailServer and CarServer were created to manage the normal message exchange between the nodes and the server. The network topology allowed for shared access networks between railways and roads, with both entities having a common core.

Figure 50: Cross-domain Emergency Service Scenario

The network topology developed to demonstrate and emulate the common emergency services for railways and roads coexistence scenario is depicted in Figure 51. The nodes Tra1, Tra2, Car1 and Car2 are connected to access point Ap1. Nodes Tra3 and Car3 are connected to access point Ap2. The access points are further connected to switches S11 and S44, respectively. Switch S22 connects to the Rail Emergency Server (RailEmer) and Car Emergency Server (CarEmer), while switch S33 connects to RailServer and CarServer, which are responsible for managing the normal message exchange between the nodes and the server.

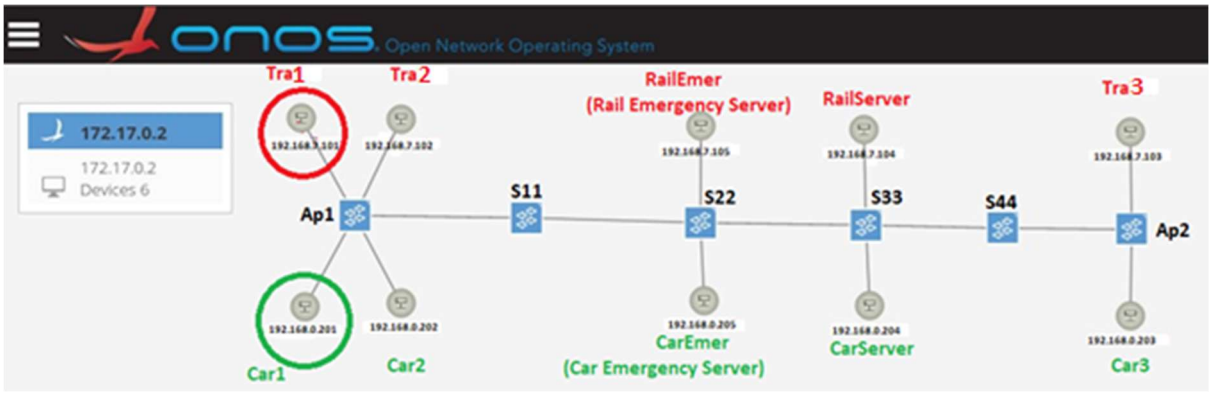

Figure 51: Considered Scenario: Shared Access Network and Shared Core, Track Perpendicular to Road: ONOS Screenshot Shared Emergency Service and Network-based Implementation

This proof of concept aims to demonstrate how a common emergency service can be implemented and triggered in both railway and road domains. Although not a complete implementation of the eREC mechanism, this work showcases how emergency services can be deployed in a coexisting scenario. The "start-eREC" message can be initiated by cars or trains in the level crossing area. The implementation of the emergency service is based on Software Defined Network (SDN) technology, which allows for the identification and forwarding of emergency data packets to the respective emergency servers designated for railways and roads. For example, if an emergency message is sent from a train to the Rail Emergency Server, it should also be forwarded to the Road Emergency Server to ensure that all relevant parties receive the message. Similarly, if an emergency message is sent from a car to the Car Emergency Server, it should also be forwarded to the Rail Emergency Server.

When a User Datagram Protocol (UDP) data packet arrives at an access point or switch, the device checks its forwarding rules. If there are no rules for the source and destination IP pairs, the data packet is sent to the SDN controller as an OpenFlow PacketIn message. The developed SDN application checks the destination port of the UDP data packet, and if it is 135, the packet is marked as an emergency packet and tagged with VLAN ID 5. Port 135 is designated for receiving emergency data packets via UDP. The SDN application duplicates the emergency packet and changes the destination IP address of the duplicate packet. If a Car initiates the emergency message, the original packet is sent to the Car Emergency Server, and the duplicate is sent to the Rail Emergency Server. If a Train initiates the emergency message, the original packet is sent to the Rail Emergency Server, and the duplicate is sent to the Car Emergency Server.

If the arrived UDP data packets have a destination port other than 135, they are marked as non-emergency packets. Data packets sent to or from Train, Rail Server, and Rail Emergency Server are assigned VLAN ID 3, while data packets sent to or from Car, Car Server, and Car Emergency Server are assigned VLAN ID 4.

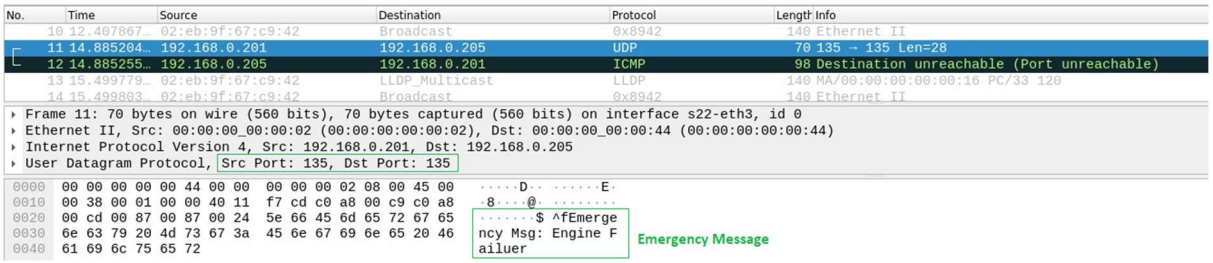

Figure 52: Emergency Data Packet from Car1 to Car Emergency Server: Data Packet Captured at CarEmer

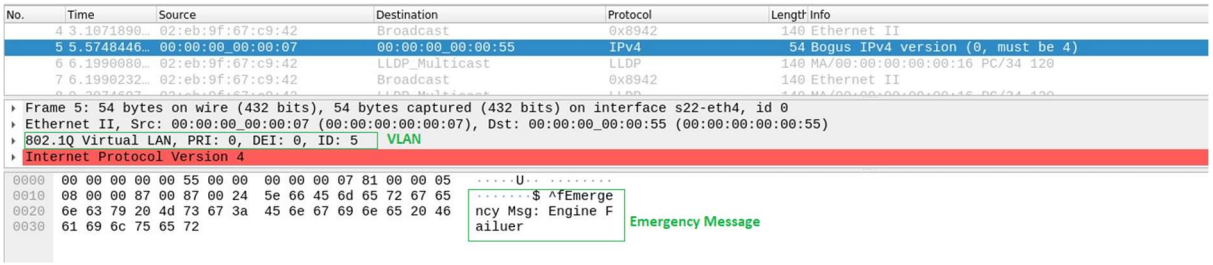

Figure 53: Duplicate Emergency Data Packet Sent to Rail Emergency Server: Data Packet Captured at RailEmer

Figure 52 shows the emergency data packet captured at Car Emergency Server (CarEmer) and Figure 53 shows the emergency data packet captured at Rail Emergency Server using Wireshark tool. When an emergency message is sent from Car1 to Car Emergency Server, the developed SDN application differentiate the data packet and mark this data packet as “Emergency Data Packet” and tagged this data packet with VLAN 5. After duplicating it, it sent the original data packet to Car Emergency Server and duplicated emergency data packet to Rail Emergency Server (RailEmer), as shown in Figure 53. Similarly, when an emergency message is sent from Tra1 to Rail Emergency Server (RailEmer), an SDN application is able to send this message to Rail Emergency Server as well as Car Emergency Server, as shown in Figure 54 and Figure 55.

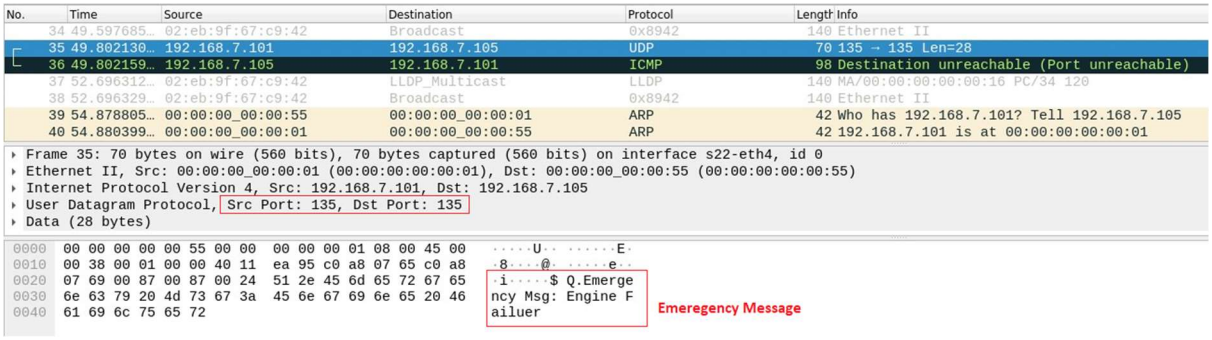

Figure 54: Emergency Data Packet from Tra1 to Rail Emergency Server: Data Packet Captured at RailEmer

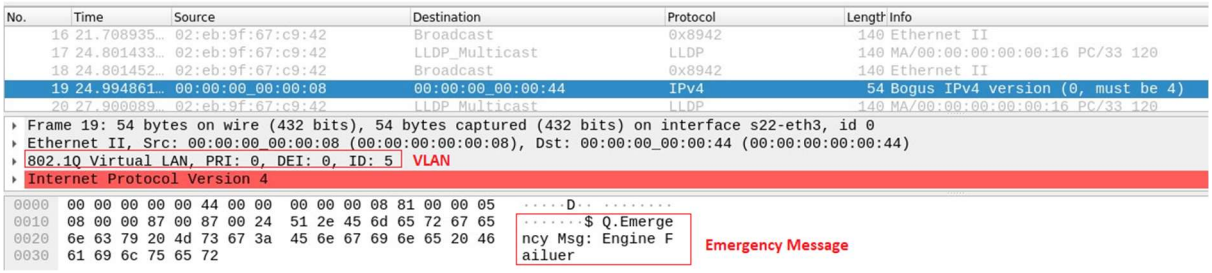

Figure 55: Duplicate Emergency Data Packet Sent to Car Emergency Server: Data Packet Captured at CarEmer

In conclusion, the SDN-based application developed in this project successfully demonstrated the capability to send emergency messages to both the Rail Emergency Server (RailEmer) and Car Emergency Server (CarEmer) in a scenario of railways and roads coexistence. This proof of concept shows that cross-domain emergency services can be implemented to serve both domains.

5.2 Shared Emergency Service and Application-based Implementation

Following this implementation at the network level, our idea was to use this approach to propose a solution at the application level: 1) integrating this network-based solution to optimise latency, 2) compatible with existing solutions both in the literature and in industry, 3) capable of guaranteeing the proper functioning of very low latency applications, and 4) relying on the data processing architecture deployed in the second emulator platform designed.

To do so, we considered the use of MQTT (Message Queue Telemetry Transport) [15], one of the most widely used solutions at the application layer. It has many advantages, including lightness, portability and reliability. It is based on an asynchronous Publish-Subscribe (Pub/Sub) pattern. In this model, messages are transmitted by senders (publishers) to a broker (car/train server potentially) that manages the delivery of these messages to recipients (subscribers).

To be able to simultaneously manage a large number of devices (trains/cars) and guarantee low latency communications, the distribution of the MQTT broker (car/train server) has been proposed in numerous studies, both academic and industrial [16, 17], especially using an Edge Computing architecture. In this case, a cluster of MQTT brokers is deployed on different physical machines and

interconnected by the network. Each broker manages a given number of devices (trains/cars), generally located in the same geographical area.

SDN-based MQTT clusters have been proposed in different studies [18, 19]. This has demonstrated the relevance of this approach both in terms of the network's ability to respond to cluster evolutions (failures, integration of a new broker, etc.), and in terms of performance (latency, bandwidth usage). However, there are several limitations to this work today: 1) the existing solutions aim to replace the MQTT architecture rather than to improve it, which makes them incompatible with the solutions deployed today on a large scale, 2) the implemented mechanisms for message distribution to the various brokers imply delays that are incompatible with low-latency applications, and 3) the studies focus on the distribution of messages between the network's brokers, and not to the end clients (subscribers), which is actually the objective of MQTT.

That is why during this phase we designed a new SDN-based distributed MQTT broker: SoD-MQTT based on the network layer solution that we proposed earlier. It aimed at demonstrating the potential benefits of such an approach.

5.2.1 SOD-MQTT Architecture

The SoD-MQTT architecture is shown in Figure 56. Compared to SDN-based architectures for MQTT clusters, the proposed solution has the following characteristics:

- The idea of Local and Remote MQTT Broker: Because each MQTT Broker (car/train server) manages a specific geographical area, the Local Broker is the broker managing the area where a device (train/car) is located. Remote brokers correspond to all other brokers in the cluster (other servers). This classification allows an efficient management of the devices subscription (cf. Section 5.2.2) and can also allow a message to be broadcast to a specific area/to a specific number of brokers, an idea that has not been considered yet (cf. Section 3.C). It can be noted that communications are possible between SDN switches depending on different geographical areas;
- The standardization of exchanges between MQTT brokers and SDN controllers: The implementation of a REST API (REpresentational State Transfer) is proposed to allow the integration of the SDN technology in the existing MQTT architecture. This API can be used for the management of brokers (addition, deletion), subscribers (addition, deletion), and the publication of messages. It represents a global interface between MQTT Brokers and SDN Controller. This could allow us to integrate this approach in currently deployed solutions without impacting the MQTT protocol. It can be noted that this approach could also be applied to existing studies;
- The use of OpenFlow Group Tables at the SDN switch level for data dissemination within MQTT clusters: These Group Tables [20] are a feature of the OpenFlow protocol allowing multicast management in an SDN system. We propose to use them to calculate optimal communication paths to MQTT clients (publishers/subscribers) and thus optimise the distribution of information but also to manage communication within the MQTT cluster. The idea is to be able to simultaneously transmit data to different MQTT brokers, without using a Root Broker. This is an important evolution compared to existing solutions.

SoD-MQTT, from an architectural point of view, presents three main evolutions: 1) the possibility to identify different types of MQTT brokers, 2) the standardization of exchanges between brokers and controllers, and 3) a complete use of OpenFlow Group Tables. The implementation of the proposed system is described in detail in Sections 0 and 5.2.2.

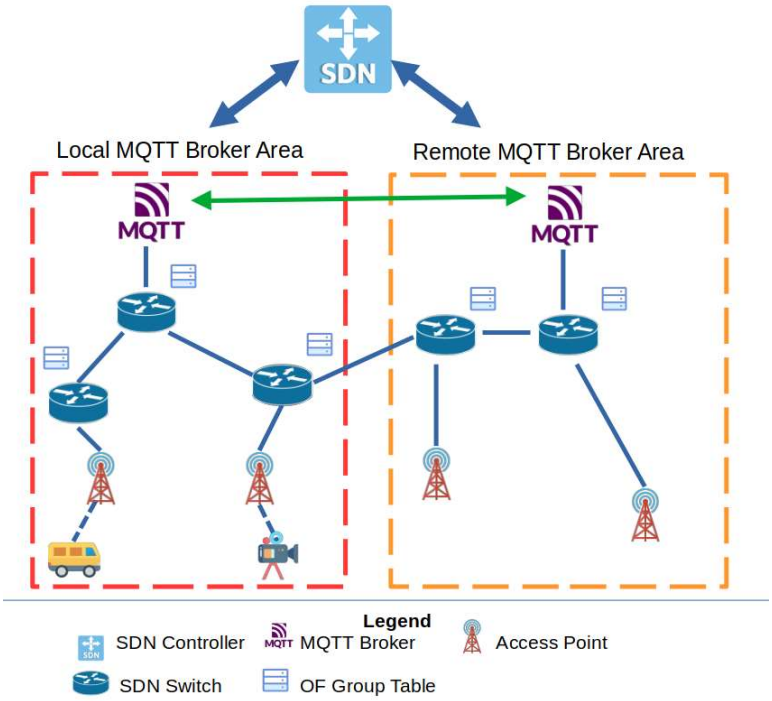

Figure 56: High-Level View of SoD-MQTT

SOD-MQTT Brokers Management (Inter-Edge Servers Management)

In existing SDN-based Distributed MQTT architectures, cluster management relies on sequential distribution to the different brokers. A Root Broker that receives a message from a publisher will transmit it to a set of neighbouring brokers that will then transmit it to their neighbours. With SoD-MQTT, a different process is used. It is based on the OpenFlow Group Tables.

When a new broker is added to an existing cluster, the procedure is shown below (cf. Figure 57):

- The broker (or cluster operator) asks the SDN controller via the Rest API to be added to a given cluster. If there are multiple clusters, each cluster is identified by a unique ID. The broker is also identified by an ID that will allow it to be associated with a given geographical area;
- The SDN controller calculates the optimal path within the core network to reach this broker. This path aims to: a) minimize latency and b) limit the number of flow rules deployed at the SDN switch level;
- The controller sends the corresponding flow rules (entries) to the SDN switches. These rules are integrated into the group tables and must allow data to be transmitted simultaneously to the various brokers in a cluster (multicast);
- The controller adds the newly arrived broker to its database, indicating the cluster it is being attached to. This will allow us to identify it later.

Note that this simple way of adding an MQTT broker to an existing cluster can be applied to different scenarios. In the case where:

- The cluster does not exist: the controller starts by creating a new cluster in its database and associates it with this broker. A new rule/entry is also created in the Group Table of the concerned SDN Switches to allow the integration of new brokers to this cluster in the future;
- A MQTT cluster is organized into a set of sub-clusters: The objective here may be to implement local/geographic services and to optimise the use of the available bandwidth, avoiding the transmission of messages to all the members of an MQTT cluster. In this case, the SDN controller stores not only a list of MQTT clusters and brokers but also a list of sub-clusters. The sub-clusters are managed in the same way as the clusters. The procedure for transmitting a message to a given cluster or subcluster is defined in section 5.2.2.

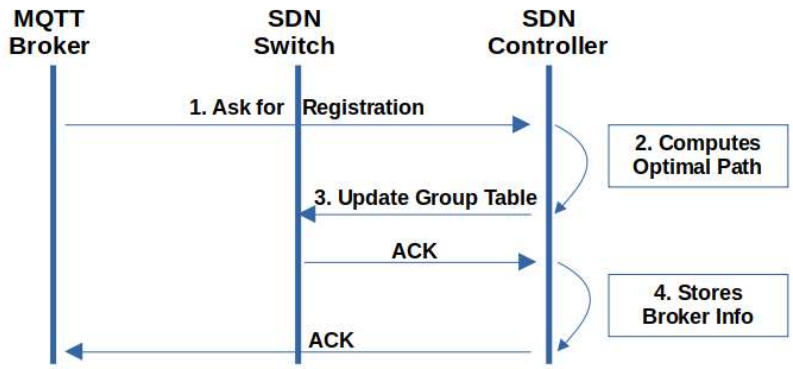

Figure 57: Exchanges between Brokers and Controllers in SoD-MQTT

The MQTT broker registration process with SoD-MQTT, therefore, ensures that data can be transmitted simultaneously to all the brokers associated with a given publication. This aims to minimise the information transmission delays in the network. When a Broker wants to be de-registered, it simply sends a request to the SDN controller. The flow rules corresponding to this Broker will be removed from the Group Tables of the SDN switches and the controller will delete it from its database. If this broker was the only broker of a given cluster, this cluster will also be deleted.

5.2.2 MQTT messages management in SoD-MQTT

The MQTT messages can be classified into two main types: messages related to the activities of publishers (nodes emitting messages) and those related to the activity of subscribers (nodes receiving messages).

To distinguish these types of messages and to offer a more fine-grained management protocol integrating, for example, the idea of sub-clusters (cf. Section 5.2.1), the proposed solution is based on a simple idea: select a specific TCP port when the message is transmitted by an MQTT client (publisher/subscriber). This approach offers an important advantage: it is compatible both with the SDN technology, which enables traffic differentiation based on the TCP port indicated in the transmitted packet, and with existing MQTT solutions, which enable the TCP ports to be defined at

both the MQTT broker and the client level. This approach can therefore be integrated into the MQTT architectures currently deployed without requiring major changes.

In our system, a reserved TCP port SUB_PORT is intended for actions related to subscribers: new subscription, Keep-Alive Message, etc. The objective is that the messages sent by this subscriber are only transmitted to the MQTT Broker managing the geographical area in which this subscriber is located. This avoids the transmission of these messages to all the brokers in the cluster and thus limits unnecessary message transmission. This point is not considered in existing works and represents an interesting advance in terms of optimization of the network performance. If the message sent to the MQTT broker is a new subscription request, it is sent to the SDN controller to deploy the communication paths used to reach the IoT device. Group Tables are attached to each topic and each broker and are used to manage the distribution of messages within the network, similar to cluster management (cf. Section 0).

Topics (and therefore published messages) can also be attached to specific TCP ports. In a basic case, considered in the literature, a message sent by a publisher is transmitted to all the brokers on the network. In this case, the port PUB_PORT_0 is used and seems sufficient if we do not want to consider other scenarios. Therefore, two ports are defined: SUB_PORT and PUB_PORT_0. In more complex scenarios, we can consider that messages must be broadcast in a given geographical area (sub-cluster). Considering X sub-clusters, the PUB_PORT_{1-X} ports can be set. When the message is generated by a publisher, depending on the topic, it will be attached to a port according to a predefined list aimed at providing access to the various sub-clusters.

Thus, three main scenarios can be identified:

- Transmission of the message to the Broker managing the geographical area in which the device (car/train) is located: This corresponds to most of the actions related to the subscriber (Local Broker): only the closest MQTT broker is concerned, so the message is not duplicated. This can also correspond to a scenario in which a sub-cluster consists of only one broker. In this case, the message sent by the publisher is only transmitted to a single broker;
- Transmission of the message to a set N of Brokers (N lower than the total number of brokers): This is the classic case of transmission of a message from a publisher to an MQTT subcluster. The message will then be transmitted to all the subscribers concerned;
- Transmission of the message to all brokers: this is the classic case when a message is sent by a publisher. This message is transmitted to the whole MQTT cluster to be then broadcast to interested subscribers. This can also correspond to an unusual case where an MQTT client would like to subscribe to a topic that is not yet registered: the local broker transmits the information to all the brokers in the cluster so that they can record this new topic (potential new service at a given point in time).

SoD-MQTT, besides being based on a new architecture, allows a high level of flexibility for message distribution.

5.2.3 Evaluation

In this section, we aim to evaluate the performance level of SoD-MQTT. Besides enabling integration into currently widely deployed MQTT architectures, SoD-MQTT defines new message management mechanisms. Therefore, in this section, we seek to verify that:

- Real-time message transmission within the MQTT cluster, enabled by SoD-MQTT, can be beneficial;
- The mechanisms proposed in SoD-MQTT for the management of MQTT clients allow to optimise the network management.

To do so, within Emu5GNet (cf. Section 3), we used:

- Mininet-WiFi: a Mininet-based emulator of software-defined networks for wireless environments [4]. This tool is commonly used for the evaluation of new mechanisms/architectures related to SDN technology. This solution has the advantage of allowing the deployment of actual services and the parameterisation of the network performance level;
- Eclipse Mosquitto: This is an open-source message broker that implements multiple MQTT versions (v5.0 in our case) and which offers the advantage of being lightweight and deployable on devices with limited capabilities. In our implementation Mosquitto is complemented by Paho MQTT, also developed by Eclipse, which is a Python library allowing to easily connect an MQTT client to a broker;
- Iperf3: This is an open-source tool that can be used to collect latency and bandwidth statistics for both TCP and UDP. So, we integrated it into our architecture.

Beyond the tools used, other elements seem to us also notable in the evaluation environment that we have put in place:

- An emulated 5G Core using Emu5GNet and a performance level corresponding to 5G Networks;
- A realistic number of brokers within the MQTT cluster, based on existing work, we therefore assumed a variable number of clusters between 3 and 12;
- A large number of data to provide relevant results (10,000 messages emitted).

A last important element is the choice of the solutions that we have selected and compared to SoD-MQTT within the framework of this experimentation. There are two solutions:

- HbH: a solution corresponding to the basic solutions described in the literature in which messages are transmitted hop by hop within the MQTT cluster (broker after broker);
- Tree: a more advanced solution in which the transmission of data within the cluster is organised in a tree (each broker transmits messages to two clusters).

In this first evaluation, we tried to determine the amount of time required to transmit a piece of information to all the brokers of a cluster in the case of the SoD-MQTT, Tree, and HbH approaches. We simply measured the time elapsed between a) the moment when a message is sent by an MQTT

Client (publisher) and b) the moment when this same message is received by all the brokers of a cluster).

What can be seen in Figure 58 is that the proposed approach allows significant gains ranging from 200% in the case of a cluster composed of 3 nodes to nearly 400% in the case of a cluster composed of 12 nodes. This is due to the use of multicast and OpenFlow group tables which had not been considered until now. This approach seems relevant to reduce the broadcast latency within an MQTT cluster as it guarantees a constant broadcast time.

Such a process of parallel sending of messages to the different brokers of a cluster could be problematic if some actions would imply synchronisation between the different brokers. In this case, a sequential distribution could seem relevant. However, in the deployed framework, the need for synchronisation does not exist when the messages are multicast to all the brokers. The only objective is to distribute the messages to all the subscribers of the network. Therefore, this solution seems only beneficial in the considered case and could be applied to the different solutions defined in the literature.

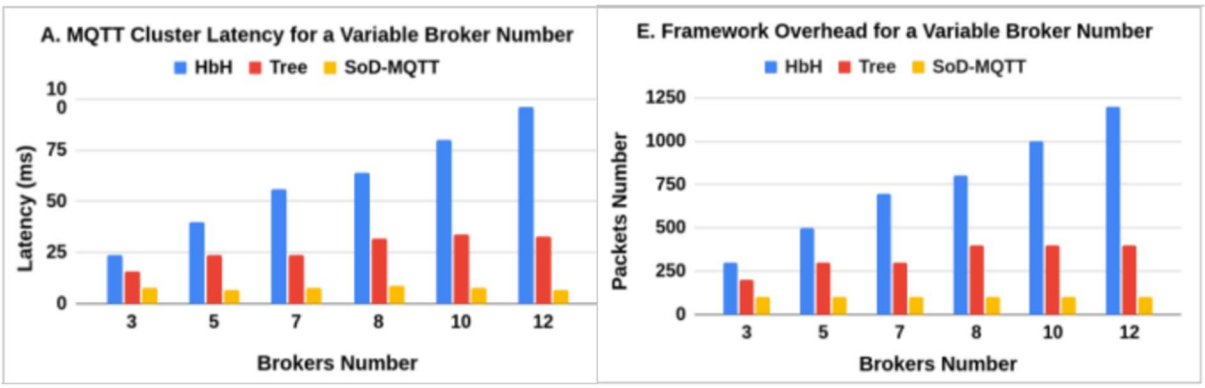

Figure 58: SoD-MQTT Evaluation

In the second evaluation, we aimed to evaluate the overhead associated with the different frameworks considered: HbH, Tree, and SoD-MQTT. In other words, for a message generated by an MQTT client and destined for all MQTT brokers, we wanted to determine the number of messages that would pass through the network. To do this we considered a base of 100 messages sent and tried to estimate how many messages would be regenerated/retransmitted by an MQTT broker.

As can be seen in Figure 59, with the SoD-MQTT approach, messages are directly broadcast to all MQTT brokers. Therefore, we can consider that no message is recreated/reissued. In contrast, in the HbH and Tree approaches, message transmissions are sequential and the number of messages passing through the network is therefore multiplied (2 to 12 times). The induced overload for the network is therefore important. Similarly, the overhead for the MQTT brokers is also existing since they may have to participate in the retransmission of messages. The overload is therefore both in terms of network capacity and computing capacity.

We can also add that SoD-MQTT also integrates the possibility of transmitting a topic only to a subset of the cluster. This solution could be applied to other existing frameworks to improve their performance level. Moreover, in the study carried out here only published messages are considered.

However, messages related to subscriptions (Keep Alive, for example) could have an important impact in terms of load when they are transmitted to all the brokers of an MQTT cluster. A problem eliminated by SoD-MQTT which manages the local distribution of these messages.

5.3 Conclusions

In this section, we have proposed a new emergency service that could be used to manage the coexistence of trains and cars. This type of service could be particularly relevant at level crossings if a problem is encountered. It could allow critical information to be transmitted simultaneously to rail and road services.

For the implementation of this emergency service, we considered two different implementations. The first one is based on a network-level implementation and on the use of SDN technology, particularly highlighted in Section 3 of this report. The development of a VLAN tagging mechanism could be used, as it has been demonstrated, to guarantee the joint dissemination of information to both servers (trains/car)

The second proposed solution is based on the implementation of a solution at the application layer. The objective of this design is threefold: 1) to rely on the proposed solution at the network level to optimise the use of available network resources, 2) to provide a solution that can be integrated into existing architectures currently used in research and industry, and 3) to optimise these solutions through the implementation of new architectures. As it has been demonstrated through experimentation, this solution, based on the MQTT protocol, meets the different objectives and could therefore be a relevant option.

In conclusion, the approaches proposed in this section appear to be a relevant way to implement new emergency services for both the rail and road environments.

6 CONCLUSIONS

The Future Railway Mobile Communication System (FRMCS) will be the 5G worldwide standard for railway operational communications, conforming to European regulation as well as responding to the needs and obligations of rail organisations outside of Europe. The work on functional & technical requirements, specification & standardisation in 3GPP as well as regarding harmonised spectrum solutions is currently led by UIC, in cooperation with the whole railway sector. In this context, the 5GRAIL project aims at verifying the first set of FRMCS specifications and standards (FRMCS V1) by developing and testing prototypes of the FRMCS ecosystem. The validation of the latest available railway-relevant 5G specifications will be achieved through trials covering significant portions of railway operational communication requirements and including the core technological innovations for rail expected from 5G release 16 and pre-release 17.

In this context, the main objective of WP6 is the evaluation of the coexistence of rail and road automotive communication use cases. The possible synergies allowed by FRMCS between both vertical industries based on a situation implying common use cases will be evaluated. The objective of deliverable D6.1 was the identification and definition of possible rail and road coexistence scenarios.

In continuity, the aim of this deliverable D6.2 was to implement some of the scenarios identified in D6.1 in a realistic network emulator framework. The different steps towards the implementation of a complete emulator are presented in this deliverable. This is divided into two steps. First, an identification of all the requirements that must be taken into account by the emulator. Then, the implementation of a first emulator which allowed to validate and evaluate all the scenarios selected from D6.1. Thereafter, the definition of a second emulator, which is more complex, allows the implementation of a large set of scenarios, including multi-radio access technologies, edge computing, and roaming. The evaluations presented for these two emulators demonstrate their relevance and their potential. Finally, an emergency service (network-level implementation and application-level implementation) aiming at ensuring the coexistence of trains and cars at level crossings has also been presented in this deliverable.

The simulation/emulation environments implemented in this WP6, and more specifically in task 6.2, open the door to many new opportunities. Indeed, they could be used in the context of many applications such as 1) the definition of optimal Edge architectures by analyzing the resources required to run given applications in a given traffic context, 2) the analysis and comparison of different solutions for cross-border scenario management enabled by the implementation of different 5G cores and their interconnection and 3) the definition and validation of new services such as the emergency service presented in this document.

7 REFERENCES

[1] [Online] Available : https://www.riverbed.com/products/npm/riverbed-modeler.html

[2] [Online] Available : https://omnetpp.org/

[3] [Online] Available : https://www.nsnam.org/

[4] [Online] Available : http://mininet.org/

[5] [Online] Available : https://opennetworking.org/onos/

[6] [Online] Available : https://www.eclipse.org/sumo/

[7] [Online] Available : https://openairinterface.org/

[8] [Online] Available : https://wiki.linuxfoundation.org/networking/netem

[9] [Online] Available : https://mininet-wifi.github.io/get-started/

[10] [Online] Available : https://mininet-wifi.github.io/mac80211_hwsim/

[11] Smith, P., Majumdar, A., & Ochieng, W. Y. (2012). An overview of lessons learnt from ERTMS implementation in European railways. Journal of Rail Transport Planning & Management, 2(4), 79-87.

[12] Sniady, A., & Soler, J. (2012, November). An overview of GSM-R technology and its shortcomings. In 2012 12th International Conference on ITS Telecommunications (pp. 626-629). IEEE.

[13] Siergiejczyk, M. (2016). Railway Radio Communication Systems in the Context of an Emergency Call. In Challenge of Transport Telematics: 16th International Conference on Transport Systems Telematics, TST 2016, Katowice-Ustroń, Poland, March 16–19, 2016, Selected Papers 16 (pp. 223-234). Springer International Publishing.

[14] Santa, J., Pereñíguez, F., Moragón, A., & Skarmeta, A. F. (2014). Experimental evaluation of CAM and DENM messaging services in vehicular communications. Transportation Research Part C: Emerging Technologies, 46, 98-120.

[15] Light, R. A. (2017). Mosquitto: server and client implementation of the MQTT protocol. Journal of Open Source Software, 2(13), 265.

[16] Longo, E., & Redondi, A. E. (2023). Design and implementation of an advanced MQTT broker for distributed pub/sub scenarios. Computer Networks, 224, 109601.

[17] Kawaguchi, R., & Bandai, M. (2019, January). A distributed MQTT broker system for location-based IoT applications. In 2019 IEEE International Conference on Consumer Electronics (ICCE) (pp. 1-4). IEEE.

[18] Fawwaz, D. Z., Chung, S. H., Ahn, C. W., & Kim, W. S. (2022). Optimal distributed MQTT broker and services placement for SDN-edge based smart city architecture. Sensors, 22(9), 3431.

[19] Park, J. H., Kim, H. S., & Kim, W. T. (2018). Dm-mqtt: An efficient mqtt based on sdn multicast for massive iot communications. Sensors, 18(9), 3071.

[20] Lara, A., Kolasani, A., & Ramamurthy, B. (2013). Network innovation using openflow: A survey. IEEE communications surveys & tutorials, 16(1), 493-512.

[21] Singh, R.; Soler, J.; Sylla, T.; Mendiboure, L.; Berbineau, M. Coexistence of Railway and Road Services by Sharing Telecommunication Infrastructure Using SDN-Based Slicing: A Tutorial. Network 2022, 2, 670-706. https://doi.org/10.3390/network2040038

[22] [Online] Available : https://www.techopedia.com/definition/32105/vlan-tagging, What Does VLAN Tagging Mean?

[23] "Working Document for T6.2", Section 4. 5Grail.

[24] [Online] Available : https://usermanual.wiki/Pdf/mininetwifidraftmanual.297704656.pdf The User Manual.

[25] Behrisch, M.; Bieker, L.; Erdmann, J.; Krajzewicz, D. SUMO—Simulation of urban mobility: An overview. In Proceedings of the SIMUL 2011, The Third International Conference on Advances in System Simulation, ThinkMind, 23-29 October 2011 Barcelona, Spain 2011

[26] [Online] Available : https://sumo.dlr.de/docs/Tutorials/OSMWebWizard.html#getting_started OSMWebWizard

[27] Kendziorra, A.; Weber, M. Public transport, logistics and rail traffic extensions in sumo. In Simulating Urban Traffic Scenarios; Springer: Cham, Switzerland, 2019; pp. 83--95.

[28] Lopez, P.A.; Behrisch, M.; Bieker-Walz, L.; Erdmann, J.; Flötteröd, Y.P.; Hilbrich, R.; Wießner, E. Microscopic traffic simulation using sumo. In Proceedings of the 2018 21st International Conference on Intelligent Transportation Systems (ITSC), Maui, HI, USA, 4--7 November 2018; pp. 2575--2582.

[29] [Online] Available : https://github.com/DTU5GRail/5GRail_WP6/tree/main/DTU_Code/Considerd_Scenario/ DTU5GRail/5GRail_WP6

[30] [Online] Available : https://wireless.wiki.kernel.org/en/users/drivers/mac80211_hwsim mac80211_hwsim.

[31] Sylla, T.; Mendiboure, L.; Berbineau, M.; Singh, R.; Soler, J.; Berger, M.S. Emu5GNet: An Open-Source Emulator for 5G Software-Defined Networks. In Proceedings of the 18th International Conference on Wireless and Mobile Computing, Networking and Communications WiMob 2022, Thessaloniki, Greece, 10--12 October 2022.

[32] Peuster, M., Kampmeyer, J., & Karl, H. (2018, June). Containernet 2.0: A rapid prototyping platform for hybrid service function chains. In 2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft) (pp. 335-337). IEEE.

[33] Acar, U., Ustok, R. F., Keskin, S., Breitgand, D., & Weit, A. (2018, November). Programming tools for rapid NFV-based media application development in 5G networks. In 2018 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN) (pp. 1-5). IEEE.

[34] Soenen, T., Van Rossem, S., Tavernier, W., Vicens, F., Valocchi, D., Trakadas, P., ... & Lopez, D. (2018, April). Insights from SONATA: Implementing and integrating a microservice-based NFV service platform with a DevOps methodology. In NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium (pp. 1-6). IEEE.

[35] Gawor, A. (2022). Design and deployment of educational 5G platform (Doctoral dissertation, Instytut Telekomunikacji).

[36] Neto, F. J. D. S., Amatucci, E., Nassif, N. A., & Farias, P. A. M. (2021, November). Analysis for comparison of framework for 5G core implementation. In 2021 International Conference on Information Science and Communications Technologies (ICISCT) (pp. 1-5). IEEE.

[37] [Online] Available : https://github.com/tsylla/5grail-emu5gnet

[38] Charyyev, B., Arslan, E., & Gunes, M. H. (2020, December). Latency comparison of cloud datacenters and edge servers. In GLOBECOM 2020-2020 IEEE Global Communications Conference (pp. 1-6). IEEE.

[39] Mandalari, A. M., Lutu, A., Custura, A., Safari Khatouni, A., Alay, Ö., Bagnulo, M., ... & Fairhurst, G. (2018, October). Experience: Implications of roaming in europe. In Proceedings of the 24th Annual International Conference on Mobile Computing and Networking (pp. 179-189).

8 APPENDICES

In this section can be found the test results for the scenario S1(5/6)4, S2(5/6)1, S4(5/6)1 and S4(5/6)4 and are related to Section 3.

8.1 Implementation and tests for Coexistence Scenario 2

S1(5/6)4: Different Access Network and Different Core, Single Serving Technology, Track Perpendicular to Road: In this considered scenario, the network parameters are similar to scenario S1(5/6)1, i.e., railways and roads have dedicated radio access networks with dedicated cores, but in this scenario, railway tracks are perpendicular to roads.

8.1.1 Handover/Moving

To analyse handover/mobility, the nodes Car1 and Tra1 were configured with mobility capability. After a 60-second delay, Car1 initiated movement towards access point ap2, Tra1 started moving towards access point ap4. While in motion, Car1 pinged Car2, and Tra1 pinged Tra2. The results of the ping test, as shown in Figure A1, indicate that when Car1 reached the edge of access point ap1 and entered the coverage range of access point ap2, it successfully switched its connection from ap1 to ap3. Similarly, when Tra1 entered the coverage range of access point ap4, it successfully switched its connection from ap3 to ap4.

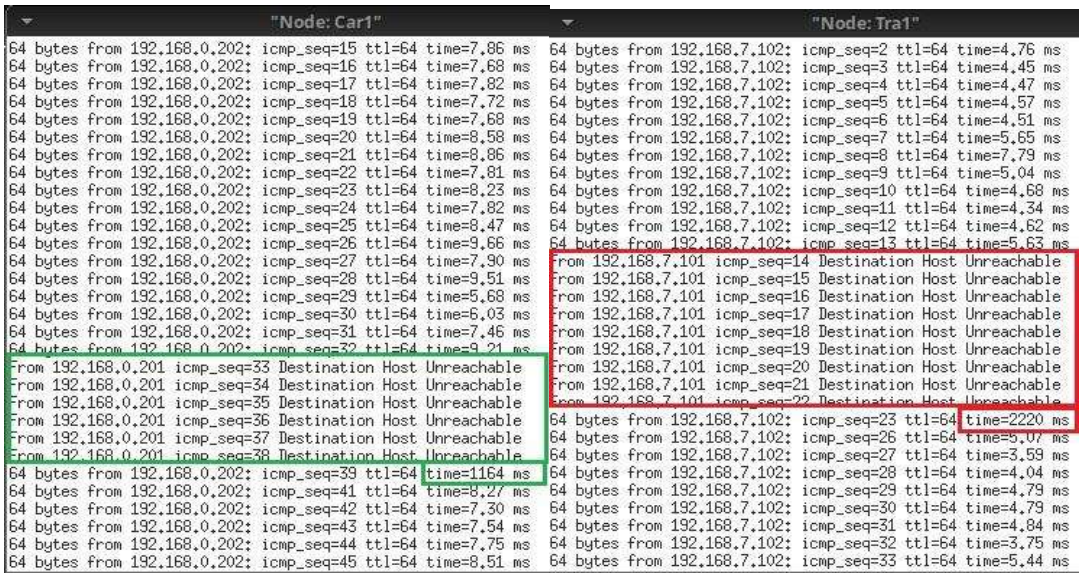

Figure A1: Checking Connectivity During Moving

In order to verify network connectivity and handover between assigned access points, both selected nodes (Car1 and Tra1) were pinging their respective service servers. The results, as shown in Figure A1, indicate that when Car1 and Tra1 cross the coverage range of their previously connected access points, they automatically switch to the nearest access point (ap2 for Car1 and ap4 for Tra1), with no packet loss recorded during the handover process.

To further confirm the handover and mobility functionality of the nodes, the commands "Car1 iw dev Car1-wlan0 link" and "Tra1 iw dev Tra1-wlan0 link" were executed for Car1 and Tra1, respectively, both before and after the nodes' movement. Figure A2 shows that initially, Car1 was connected to ap1, but after the handover, it successfully switched to ap2. Similarly, Figure A3 shows that initially, Tra1 was connected to ap3, but after the movement, it successfully switched to ap4. Figure 10 displays the topology after the nodes' movement, clearly indicating that Car1 is now connected to ap2, and Tra1 is connected to ap4.

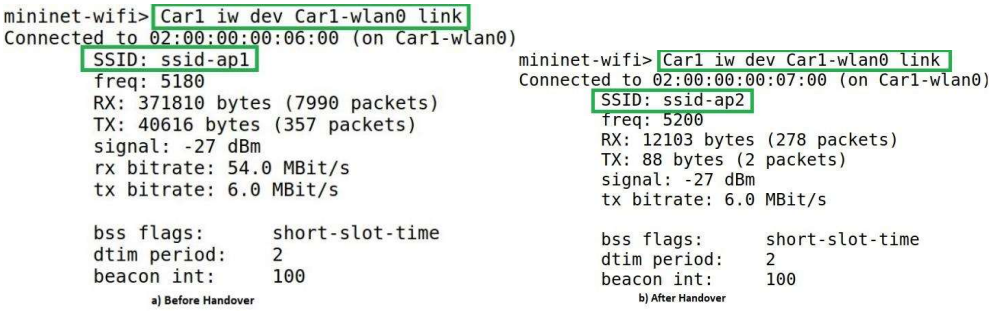

Figure A2: Connected Access Point for Car1 Before and After Handover/Moving

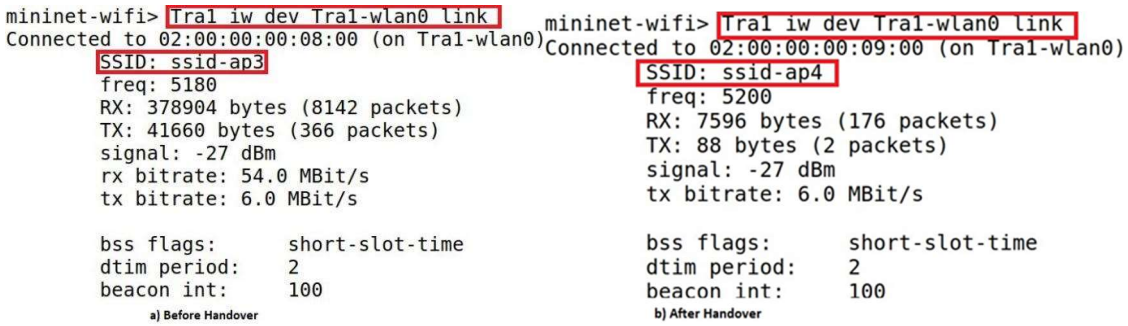

Figure A3: Connected Access Point for Tra1 Before and After Handover/Moving

The position of the nodes and access points for scenario S1(5/6)4 before and after the movement and handover are depicted in Figures 25 of Section 3 and 4, respectively. The comparison of these two figures confirms that Mininet-WiFi can successfully simulate handover and movement scenarios for coexisting railway and road environments. Although there is a delay in the handover process, there is no loss of data during the transition. The nodes and stations undergo a network joining process during the handover, which causes the delay.

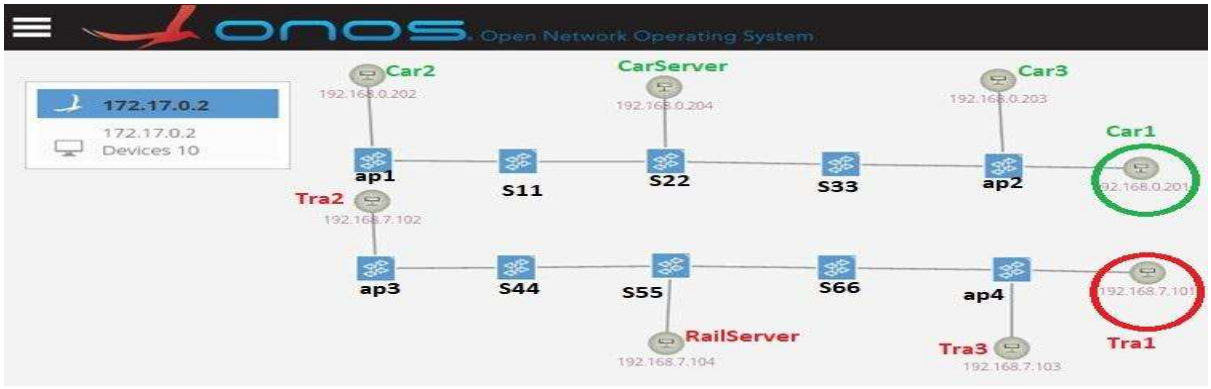

Figure A4: S1(5/6)4 Different Access Network & Different Core, Track Perpendicular to Road: ONOS Screenshot (After Handover)

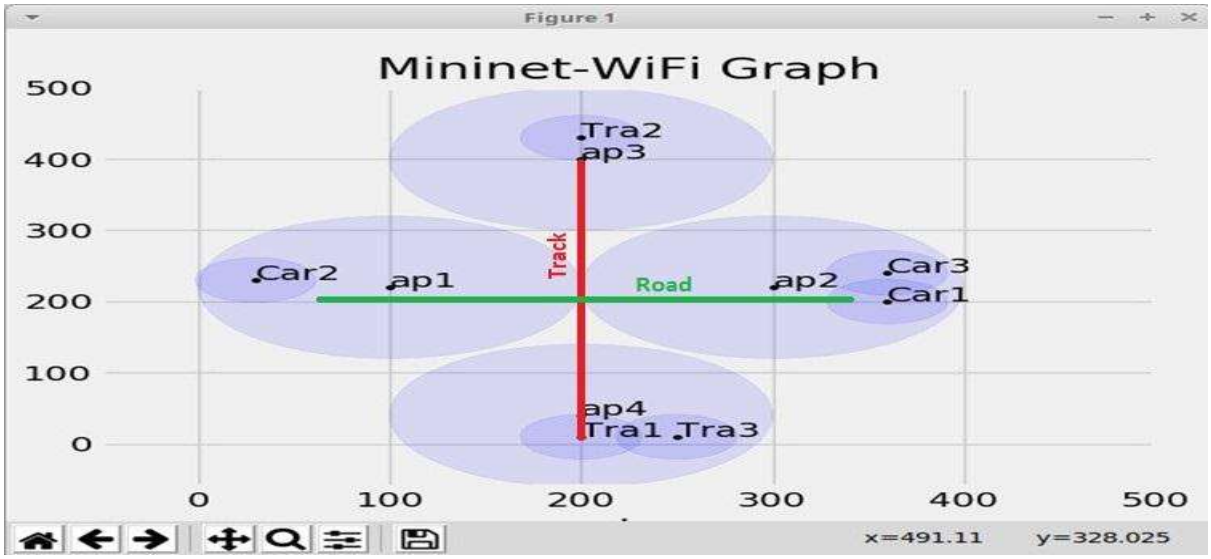

Figure A5: S1(5/6)4 Hosts and Access Points: Mininet-WiFi Graph (After Handover)

8.1.2 Reachability Test and Data Traffic Differentiation

For all nodes and hosts linked to the network topology S1(5/6)4, this test is conducted, and the findings are displayed in Table A1. According to the table, Cars can communicate solely with other Cars and the assigned road service server i.e., CarServer. Similarly, Trains can communicate only with other Trains and the designated railway service server, i.e., RailServer.

Table A1: Reachability Test

Src/Dst	Car1	Car2	Car3	CarServer	Tra1	Tra2	Tra3	RailServer
Car1	ü	ü	ü	ü	X	X	X	X
Car2	ü	ü	ü	ü	X	X	X	X
Car3	ü	ü	ü	ü	X	X	X	X

CarServer	ü	ü	ü	ü	X	X	X	X
Tra1	X	X	X	X	ü	ü	ü	ü
Tra2	X	X	X	X	ü	ü	ü	ü
Tra3	X	X	X	X	ü	ü	ü	ü
RailServer	X	X	X	X	ü	ü	ü	ü

8.1.3 UDP and TCP Transmission

The aim of this test is to demonstrate the typical data communication between cars and CarServer, as well as between trains and RailServer. Figure A6 shows the transmission of UDP data packets, while Figure A7 displays the transmission of TCP data packets from Tra1 to RailServer. In this scenario, Tra1 is operating as a client, while RailServer is configured as a listening server.

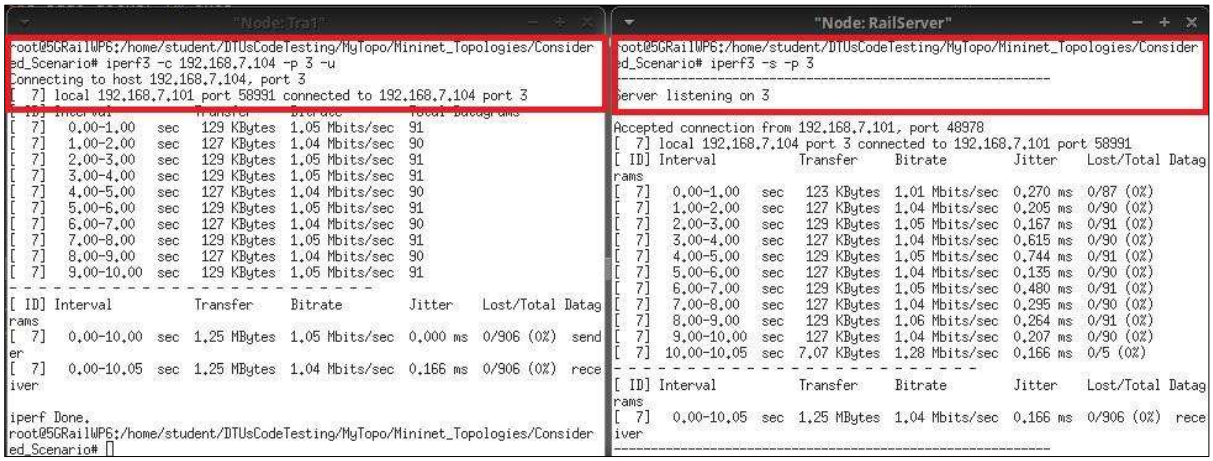

Figure A6: UDP Data Packet Transmission from Tra1 to RailServer

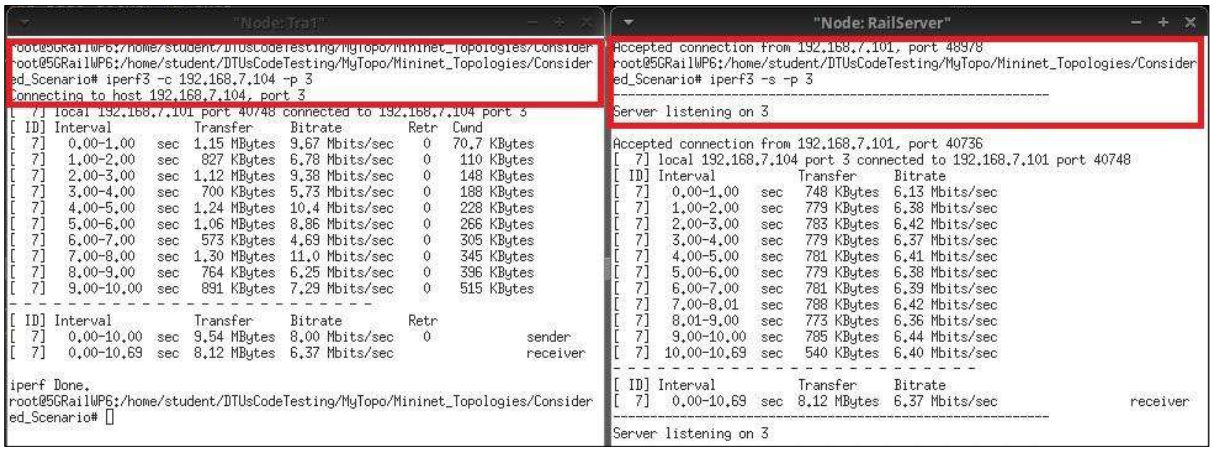

Figure A7: TCP Data Packet Transmission from Tra1 to RailServer

8.1.4 Link Capacity Test

The measurement of link capacity between Car1 and CarServer and Tra1 and RailServer is shown in Figure A8. In coexistence scenarios for both roads and railways, the bandwidth measurement obtained is enough to facilitate the transmission and reception of messages, voice, and video data.

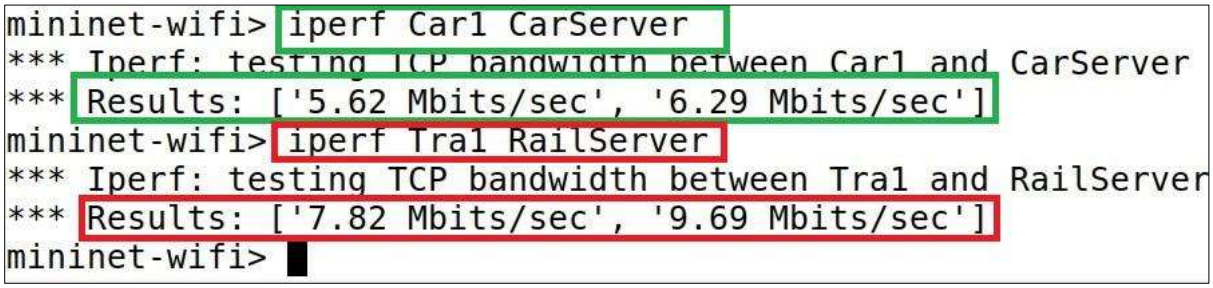

Figure A8: Link Capacity Test

8.1.5 Latency Test and Network Jitter Test

The MTR tool is used to measure losses, latency, and network jitter. To conduct the latency test, 100 UDP and TCP data packets are sent from Car1 to CarServer and Tra1 to RailServer. For this network topology, the latency ranges between 3.6 to 5.9 milliseconds, as demonstrated in Figures A9 and A10. Additionally, Figures A11 and A12 reveal that the network jitter ranges from 2.8 to 5.0 milliseconds.

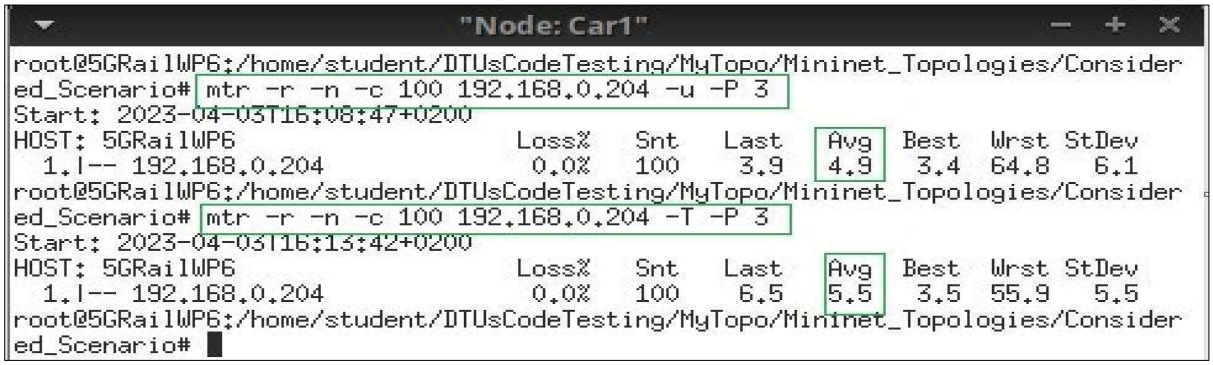

Figure A9: Latency Test from Car1

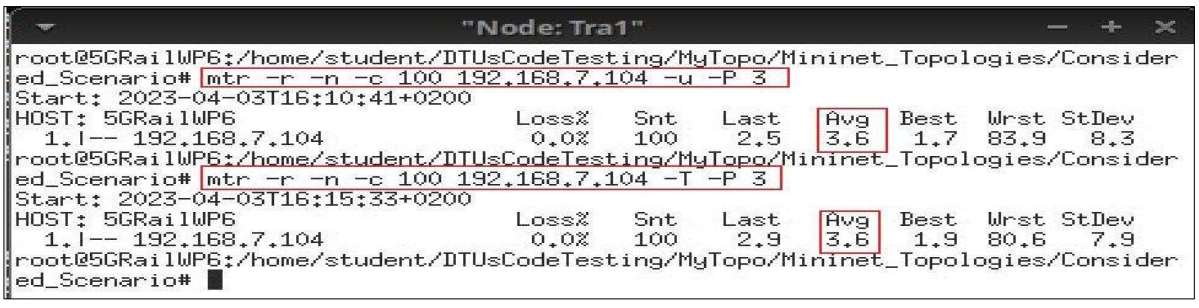

Figure A10: Latency Test from Tra1

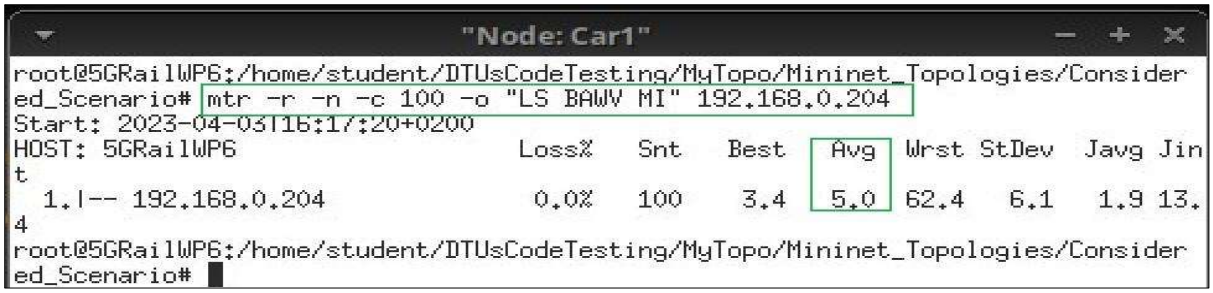

Figure A11: Jitter Test from Car1

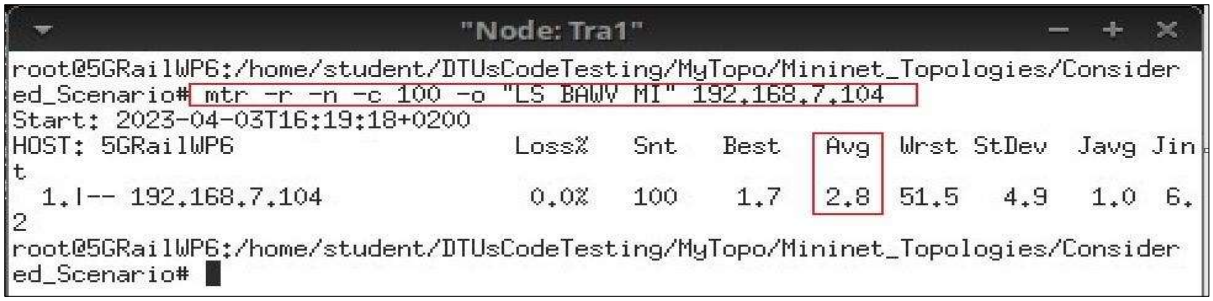

Figure A12: Jitter Test from Tra1

8.1.6 Sending a Message to the Assigned Server

The Scapy tool can be utilized in situations where it is necessary to transmit a message or information. By utilizing this tool, users can send a custom message from any node or station to the designated service server.

As an example scenario, let's say a message needs to be sent from Car1 to CarServer. To achieve this, an ICMP data packet can be created with the message "Msg: Car1 is running with Speed 60 Km/hr" and sent using the Scapy Python API through a Python script. Figure A13 shows the message creation using the Scapy. The Wireshark tool can then be used to capture this data packet, as depicted in Figure A14.

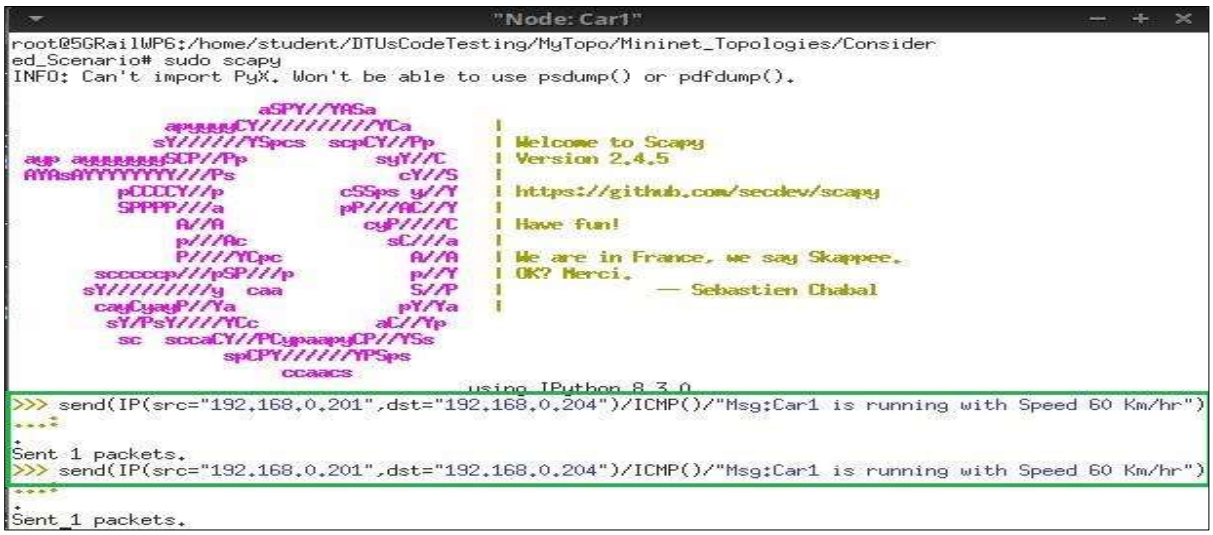

Figure A13: Scapy: Message Creation from Car1 to CarServer

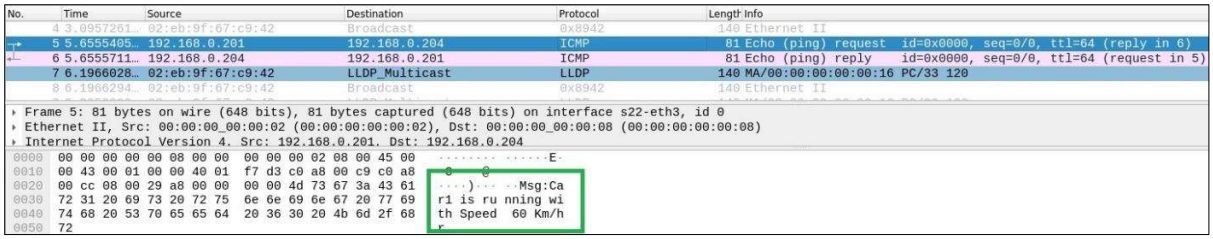

Figure A14: Wireshark: Scapy Packet with a Message

Similarly, using the Scapy a message is sent from Tra1 to RailServer with the message “Tra1 is running on Time” as shown in Figure A15. This data packet is captured at RailServer using the Wireshark tool shown in Figure A16.

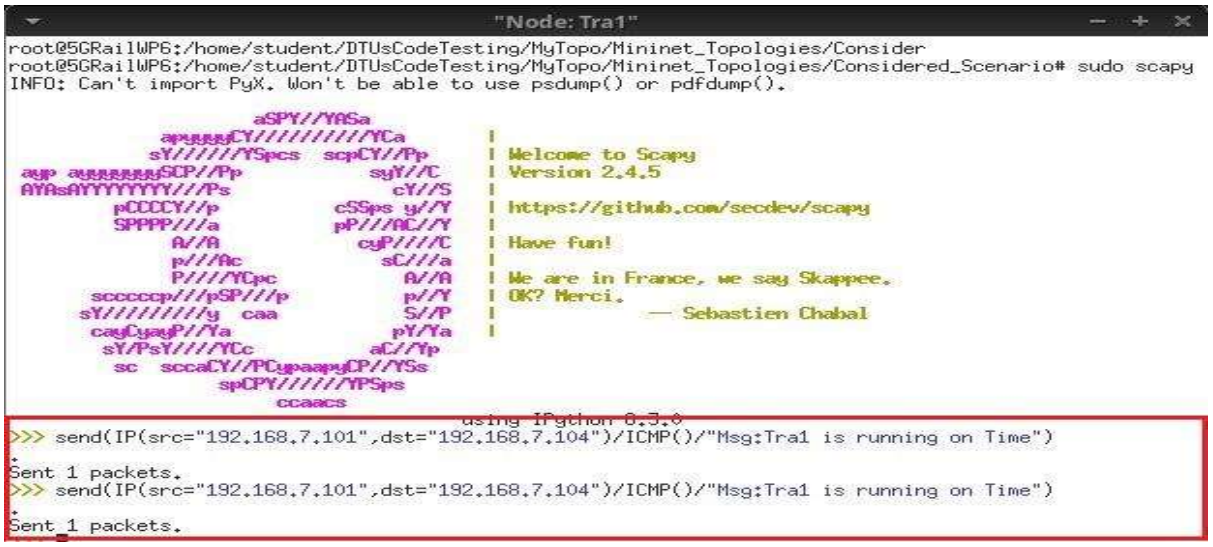

Figure A15: Scapy: Message Creation from Tra1 to RailServer

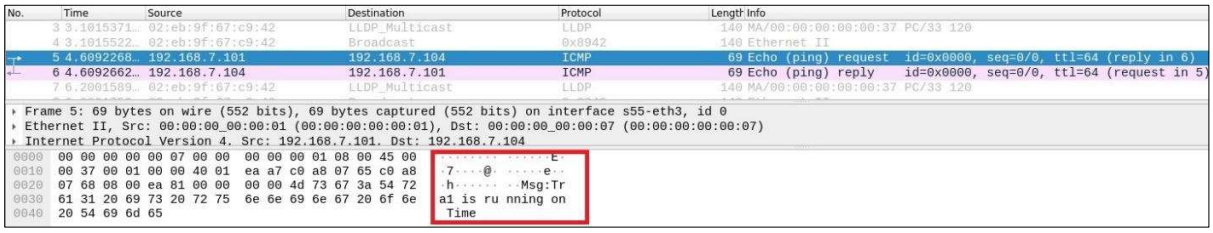

Figure A16: Wireshark: Scapy Packet with a Message

8.2 Implementation and tests for Coexistence Scenario 3

S2(5/6)1: Different Access Network and Shared Core, Single Serving Technology, Track Parallel to Road: In this scenario, railway and road domains have different radio access networks, and both domains share backhaul and core network infrastructure. In this considered scenario, railway tracks are perpendicular to roads.

8.2.1 Handover/Mobility

The mobility and handover capability of the nodes were tested by configuring Car1 and Tra1 nodes with mobility features. After a 60-second interval, Car1 started moving towards access point ap2, Tra1 moved towards access point ap4. During the nodes' movement, Car1 pinged Car2, while Tra1 pinged Tra2. The ping test results are shown in Figure 46, indicating that when Car1 entered the coverage range of access point ap2 after leaving the edge of ap1, it successfully switched its connection from ap1 to ap3. Similarly, when Tra1 entered the coverage range of access point ap4, it successfully switched its connection from ap3 to ap4.

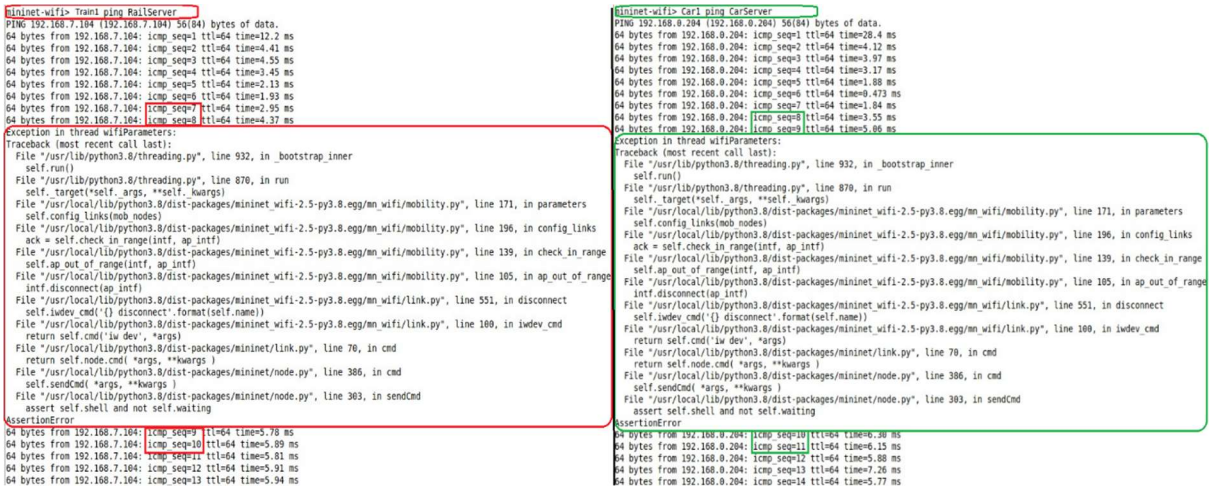

Figure A17: Checking Connectivity During Moving

In order to verify the handover and mobility functionality of the nodes, a ping test was performed between the selected nodes (Car1 and Tra1) and their respective service servers. Figure A17 demonstrates that during the nodes' movement, when they cross the coverage range of their previously connected access points, they seamlessly switch to the nearest access point (ap2 for Car1 and ap4 for Tra1). The results show that there is no packet loss during the handover process.

To further confirm the handover and mobility functionality of the nodes, the command "Car1 iw dev Car1-wlan0 link" was executed for Car1, and "Tra1 iw dev Tra1-wlan0 link" was executed for Train1, both before and after the nodes' movement. The results are shown in Figures A18 and A19, which indicate that Car1 was initially connected to ap1 but, after the handover, it successfully switched to ap2. Similarly, Tra1 was initially connected to ap3 but, after the movement, it successfully switched to ap4. Finally, Figure 49 displays the topology after the nodes' movement, demonstrating that Car1 is now connected to ap2, and Tra1 is connected to ap4.

```
mininet-wifi> Car1 iw dev Car1-wlan0 link
Connected to 02:00:00:00:08:00 (on Car1-wlan0)
        SSID: ssid-ap1
        freq: 5180
        RX: 19167 bytes (409 packets)
        TX: 2236 bytes (21 packets)
        signal: -27 dBm
        rx bitrate: 54.0 MBit/s
        tx bitrate: 54.0 MBit/s

        bss flags:      short-slot-time
        dtim period:    2
        beacon int:     100
```

a. Before Handover

```
mininet-wifi> Car1 iw dev Car1-wlan0 link
Connected to 02:00:00:00:09:00 (on Car1-wlan0)
        SSID: ssid-ap2
        freq: 5200
        RX: 388479 bytes (8306 packets)
        TX: 46700 bytes (413 packets)
        signal: -27 dBm
        rx bitrate: 54.0 MBit/s
        tx bitrate: 54.0 MBit/s

        bss flags:      short-slot-time
        dtim period:    2
        beacon int:     100
```

b. After Handover

Figure A18: Connected Access Point for Car1 Before and After Handover/Moving

```
mininet-wifi> Tra1 iw dev Tra1-wlan0 link
Connected to 02:00:00:00:0a:00 (on Train1-wlan0)
        SSID: ssid-ap3
        freq: 5180
        RX: 33665 bytes (707 packets)
        TX: 4444 bytes (41 packets)
        signal: -27 dBm
        rx bitrate: 54.0 MBit/s
        tx bitrate: 54.0 MBit/s

        bss flags:      short-slot-time
        dtim period:    2
        beacon int:     100
```

a. Before Handover

```
mininet-wifi> Tra1 iw dev Tra1-wlan0 link
Connected to 02:00:00:00:0b:00 (on Train1-wlan0)
        SSID: ssid-ap4
        freq: 5200
        RX: 6081 bytes (129 packets)
        TX: 960 bytes (10 packets)
        signal: -27 dBm
        rx bitrate: 48.0 MBit/s
        tx bitrate: 36.0 MBit/s

        bss flags:      short-slot-time
        dtim period:    2
        beacon int:     100
```

b. After Handover

Figure A19: Connected Access Point for Train1 Before and After Handover/Moving

In scenario S2(5/6)1, Figure 29 of Section 3 depicts the original configuration of nodes and access points before their movement, while Figure A20 shows their positions after the movement and handover. Comparing these two figures demonstrates that Mininet-WiFi is capable of simulating moving and handover scenarios in a coexisting railway and road environment effectively. The handover process incurs a delay due to the network joining process carried out by the nodes/stations, but no data loss is recorded.

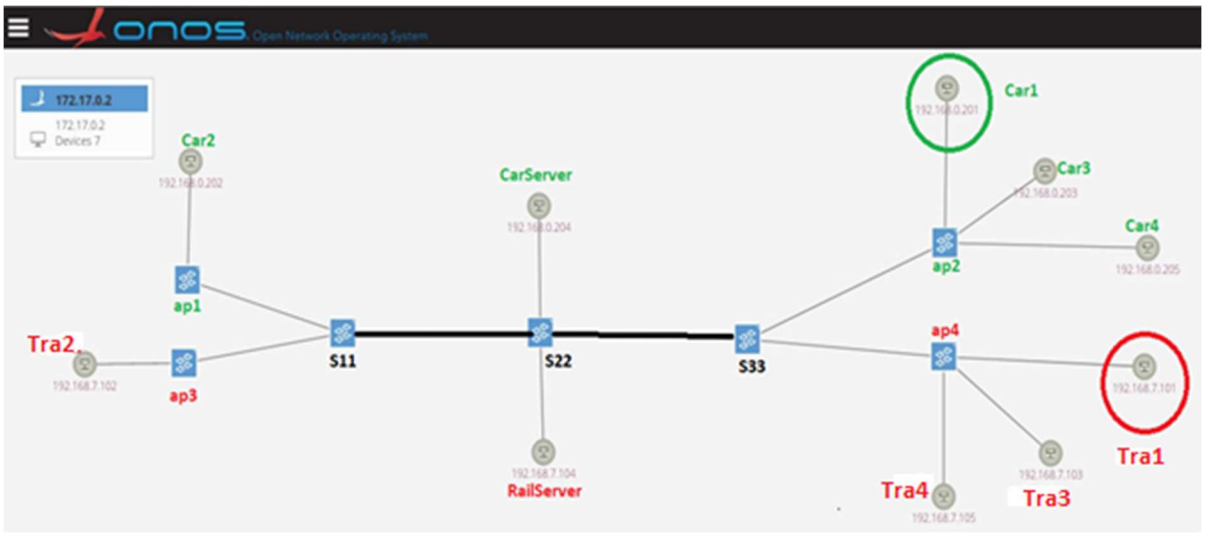

Figure A20: S2(5/6)1 Handover Scenario: ONOS Screenshot

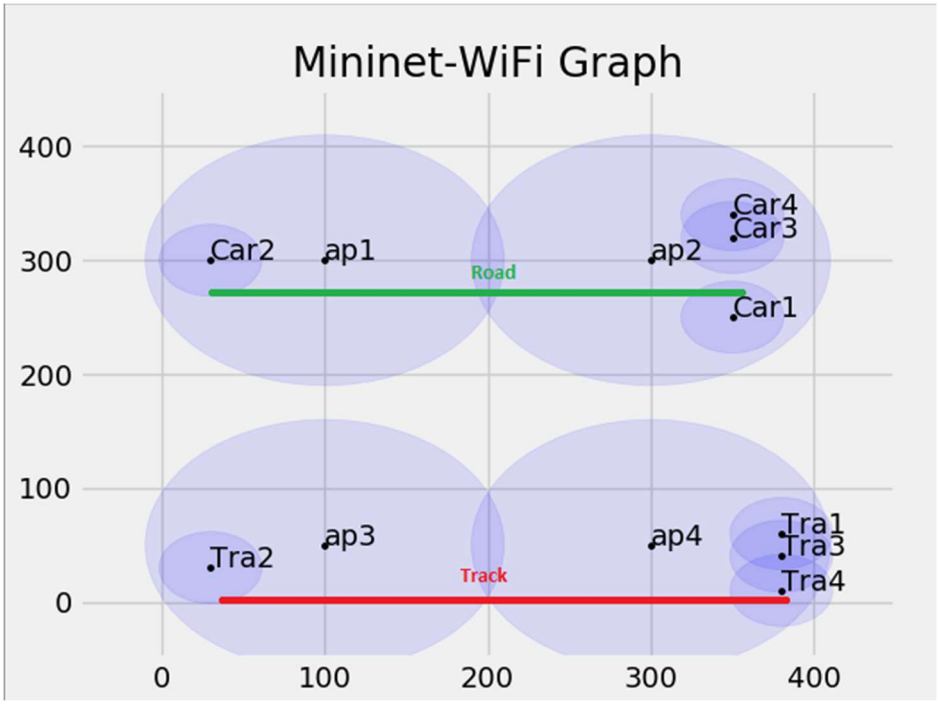

Figure A21: S2(5/6)1 Hosts and Access Points After Handover and Moving: Mininet-WiFi Graph

8.2.2 Reachability Test and Data Traffic Differentiation

For all nodes and hosts connected to network topology S2(5/6)1, this test is conducted, and Table A2 presents the results. According to the table, Cars can communicate solely with other Cars and assigned road service servers, i.e., CarServer. Similarly, Trains can communicate only with other Trains and assigned railway service servers, i.e., RailServer.

Table A2: Reachability Test

Src/Dst	Car1	Car2	Car3	Car4	CarServer	Tra1	Tra2	Tra3	Tra4	RailServer
Car1	ü	ü	ü	ü	ü	X	X	X	X	X
Car2	ü	ü	ü	ü	ü	X	X	X	X	X
Car3	ü	ü	ü	ü	ü	X	X	X	X	X
Car4	ü	ü	ü	ü	ü	X	X	X	X	X
CarServer	ü	ü	ü	ü	ü	X	X	X	X	X
Tra1	X	X	X	X	X	ü	ü	ü	ü	ü
Tra2	X	X	X	X	X	ü	ü	ü	ü	ü
Tra3	X	X	X	X	X	ü	ü	ü	ü	ü
Tr4	X	X	X	X	X	ü	ü	ü	ü	ü
RailServer	X	X	X	X	X	ü	ü	ü	ü	ü

8.2.3 TCP and UDP Data Transmission

The objective of this test is to demonstrate the standard data communication between cars to CarServer and trains to RailServer. Figure A22 shows UDP data packet transmission and Figure A23

shows TCP data packet transmission from Tr1 to RailServer. In this case, Tra1 is acting as the client, and RailServer is configured as a listening server.

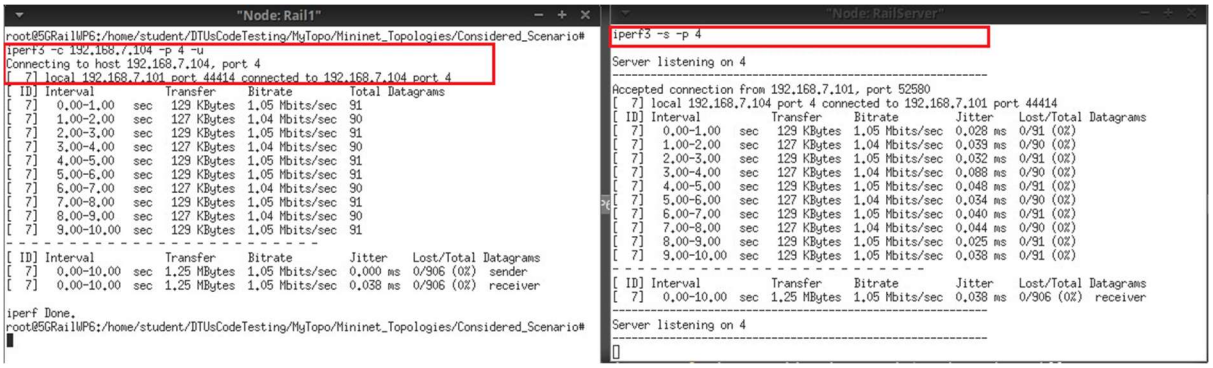

Figure A22: UDP Data Packet Transmission from Train1 to RailServer

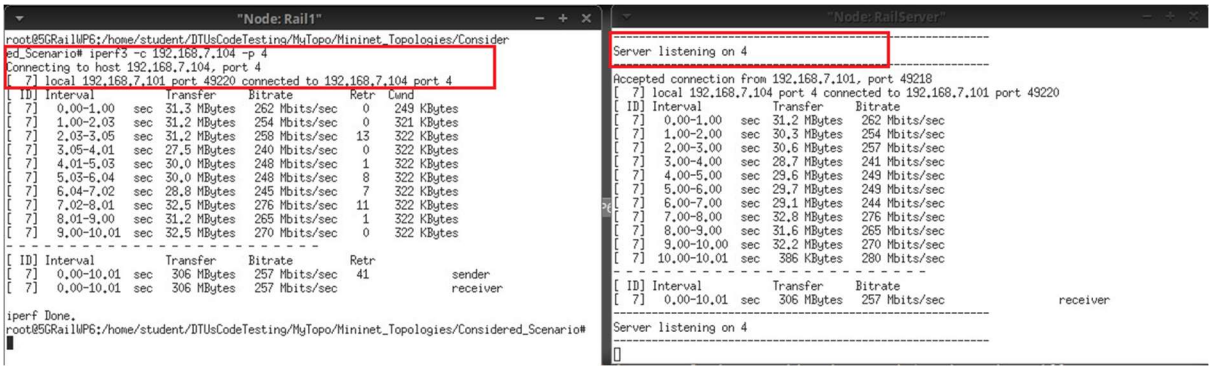

Figure A23: TCP Data Packet Transmission from Train1 to RailServer

8.2.4 Link Capacity Test

This test is carried out using the iperf tool to measure the bandwidth between two network links. To measure the bandwidth, the “iperf <Host1> <Host2>” command is used. Figure A24 shows the link capacity measurement between Car1 and CarServer and Train1 and RailServer. The achieved bandwidth measurement shows that it is adequate to send and receive messages, voice, and video data for coexistence scenarios for roads and railways.

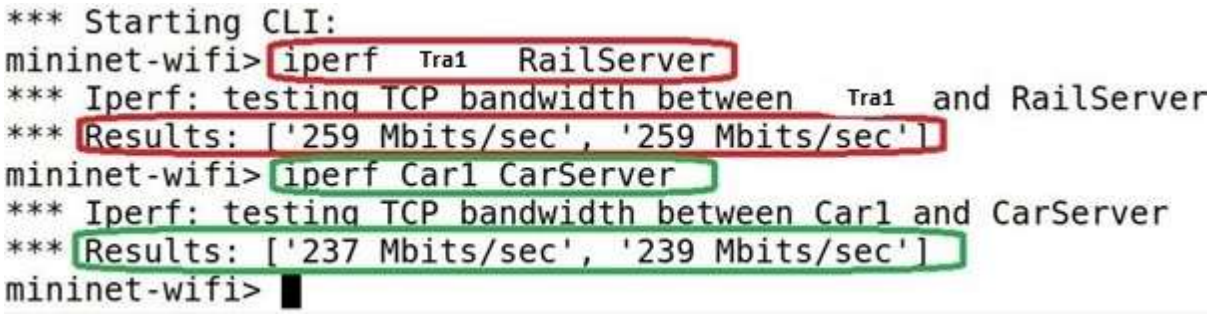

Figure A24: Link Capacity Test

8.2.5 Latency Test and Network Jitter Test

The MTR tool is used to measure losses, latency, and network jitter. To conduct the latency test, 100 UDP and TCP data packets are sent from Car1 to CarServer and Tra1 to RailServer. The latency for this network topology is in the range of 4.7 to 5.7 milliseconds. Figures A25 and A26 show the latency test for the selected network topology.

Figure A27 shows that the jitter of the network is in the range of 5.2 to 5.5 milliseconds.

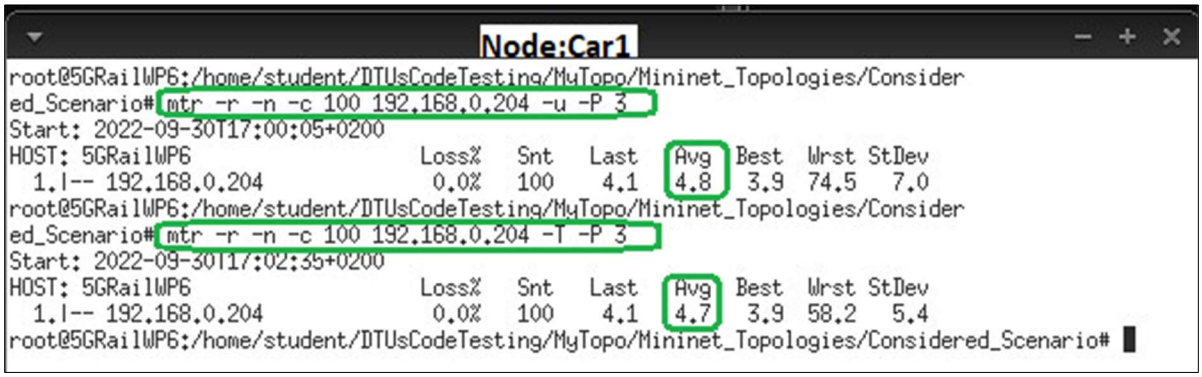

Figure A25: Latency Test from Car1

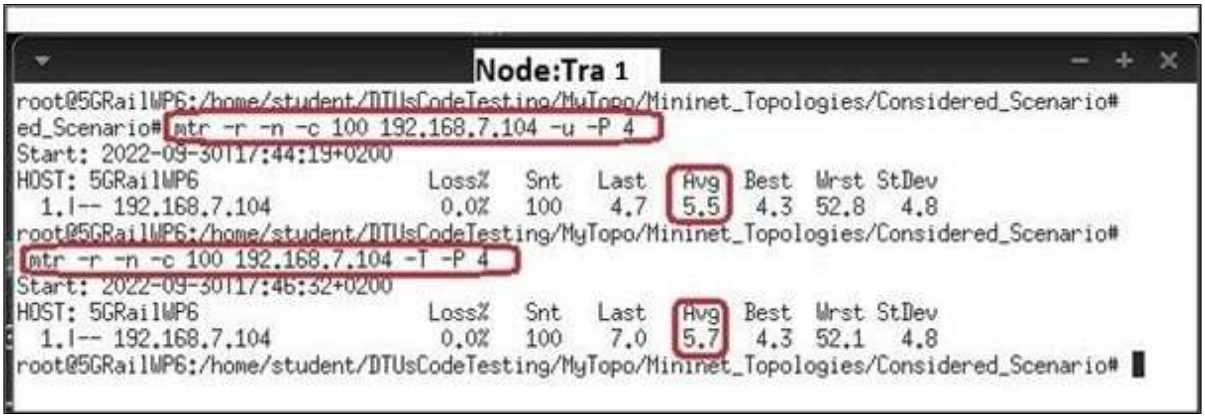

Figure A26: Latency Test from Tra1

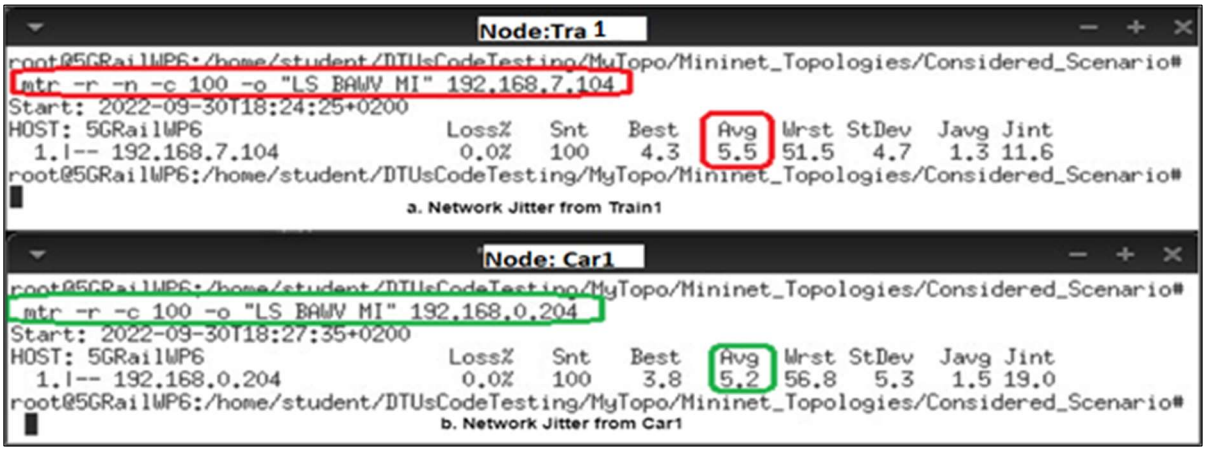

Figure A27: Network Jitter Test

8.2.6 Sending a Critical Message to the Assigned Server

To send a critical message or information, the Scapy tool is used for the considered scenarios. Using this tool, a user-defined message is sent from any node/station to the assigned service server.

Figure A28 shows that an ICMP data packet is sent with a message ``Emergency Msg: Engine Failure'' from node Tra1 to RailServer for demonstration purposes in the selected scenario. This data packet can be sent using the Scapy Python API and by writing a Python script. Figure A29 shows the data packet captured using the Wireshark tool.

Figure A28: Scapy: Message Creation

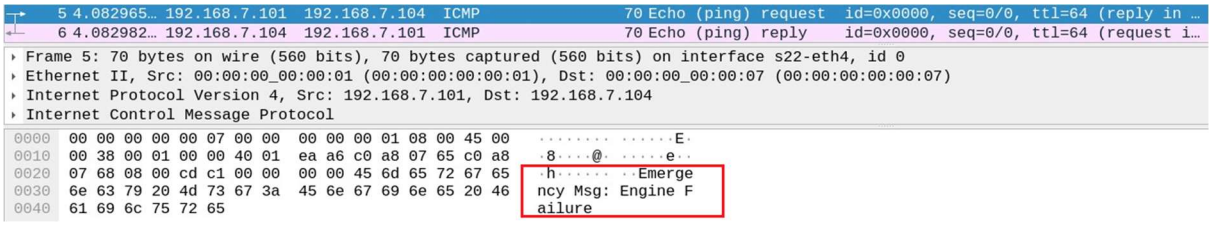

Figure A29: Wireshark: Scapy Packet with a Message

8.3 Implementation and tests for Coexistence Scenario 4

S4(5/6)1: Shared Access Network and Shared Core, Single Serving Technology, Track Parallel to Road: In this scenario, railway and road domains share the radio access network along with backhaul and core network infrastructure. Railway tracks are parallel to roads.

8.3.1 Handover/Moving

The network's handover and mobility capabilities were evaluated by configuring Car1 and Tra1 with mobility features. After 60 seconds, Car1 began moving towards access point ap2, while Tra1 started moving towards access point ap4. While moving, Car1 pinged Car2, and Tra1 pinged Tra2. The results of the ping test demonstrate that when Car1 reached the edge of access point ap1 and entered the coverage range of access point ap2, it automatically switched its connection from ap1 to ap3. Similarly, when Tra1 entered the coverage range of access point ap4, it switched its connection from ap3 to ap4.

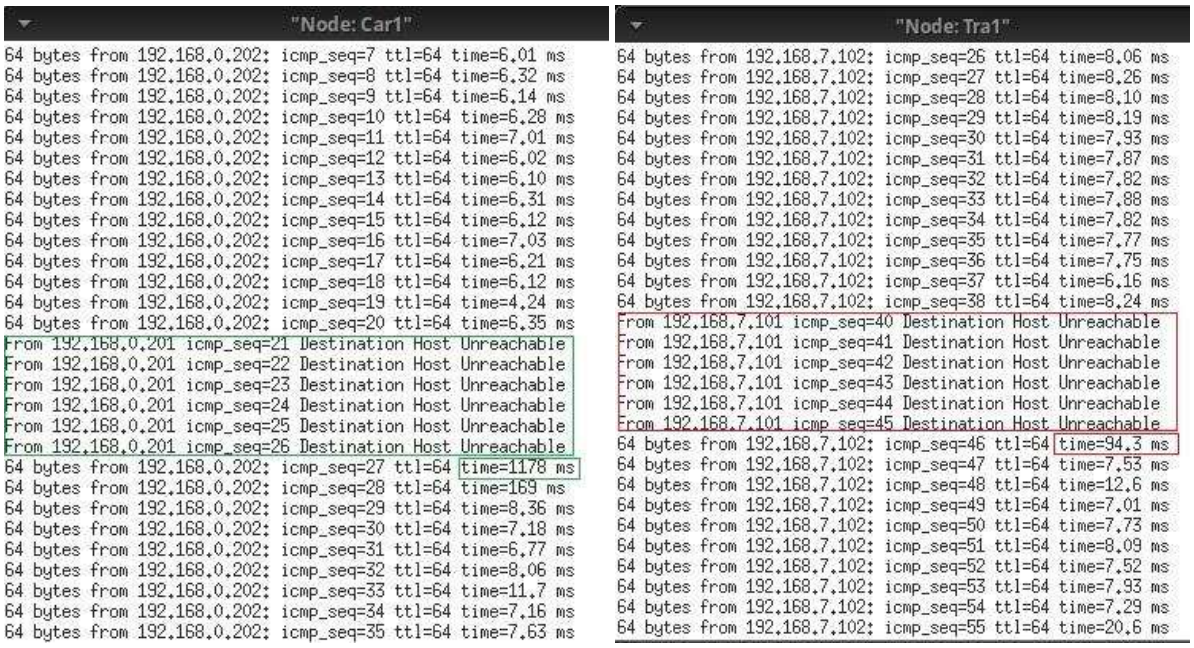

Figure A30: Checking Connectivity During Moving

The network connectivity and handover between assigned access points are tested by pinging the respective service servers of selected nodes (Car1 and Tra1). Figure A30 shows that when Car1 and Train1 cross the coverage range of their previously connected access points, they connect to the nearest access point (ap2 for Car1 and ap4 for Tra1) automatically. The figure also shows that there is no packet loss during the handover process.

To further validate the handover and mobility functionality of the nodes, the command "Car1 iw dev Car1-wlan0 link" is executed for Car1, and "Tra1 iw dev Tra1-wlan0 link" is executed for Tra1, both before and after the nodes' movement. Figure A31 illustrates that Car1 was initially connected to ap1 but, after the handover, it successfully switched to ap2. Similarly, Figure A32 demonstrates that Tra1 was initially connected to ap3, but after the movement, it successfully switched to ap4. Figure A33 presents the topology after the nodes' movement, indicating that Car1 is now connected to ap2, and Tra1 is connected to ap4.

```
mininet-wifi> Car1 iw dev Car1-wlan0 link
Connected to 02:00:00:00:04:00 (on Car1-wlan0)
        SSID: ssid-ap1
        freq: 5180
        RX: 10058 bytes (234 packets)
        TX: 88 bytes (2 packets)
        signal: -27 dBm
        tx bitrate: 6.0 MBit/s

        bss flags:      short-slot-time
        dtim period:    2
        beacon int:     100
```

a) Before Handover

```
mininet-wifi> Car1 iw dev Car1-wlan0 link
Connected to 02:00:00:00:05:00 (on Car1-wlan0)
        SSID: ssid-ap2
        freq: 5200
        RX: 433396 bytes (8435 packets)
        TX: 4854792 bytes (3190 packets)
        signal: -27 dBm
        rx bitrate: 54.0 MBit/s
        tx bitrate: 6.0 MBit/s

        bss flags:      short-slot-time
        dtim period:    2
        beacon int:     100
```

b)After Handover

Figure A31: Connected Access Point for Car1 Before and After Handover/Moving

```
mininet-wifi> Tra1 iw dev Tra1-wlan0 link
Connected to 02:00:00:00:05:00 (on Tra1-wlan0)
        SSID: ssid-ap2
        freq: 5200
        RX: 21284 bytes (494 packets)
        TX: 88 bytes (2 packets)
        signal: -27 dBm
        tx bitrate: 6.0 MBit/s

        bss flags:      short-slot-time
        dtim period:    2
        beacon int:     100
```

a) Before Handover

```
mininet-wifi> Tra1 iw dev Tra1-wlan0 link
Connected to 02:00:00:00:04:00 (on Tra1-wlan0)
        SSID: ssid-ap1
        freq: 5180
        RX: 696727 bytes (16079 packets)
        TX: 688 bytes (27 packets)
        signal: -27 dBm
        tx bitrate: 6.0 MBit/s

        bss flags:      short-slot-time
        dtim period:    2
        beacon int:     100
```

b) After Handover

Figure A32: Connected Access Point for Tra1 Before and After Handover/Moving

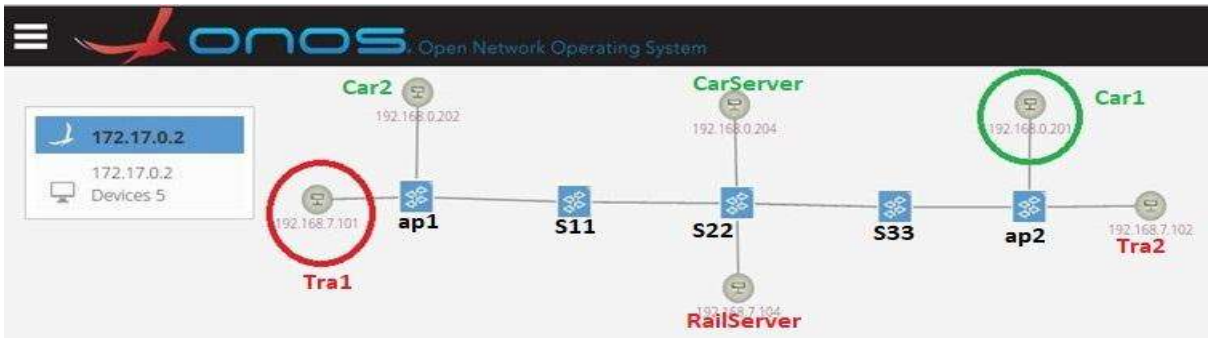

Figure A33: S4(5/6)1 Shared Access Network and Shared Core, Single, Track Parallel to Road: ONOS Screenshot (After Handover)

The comparison between these figures indicates that Mininet-WiFi can effectively simulate moving and handover scenarios for both railway and road environments. Although there is a delay during the handover process, no data loss is recorded. This delay happens due to the network joining process carried out by the nodes or stations while switching access points.

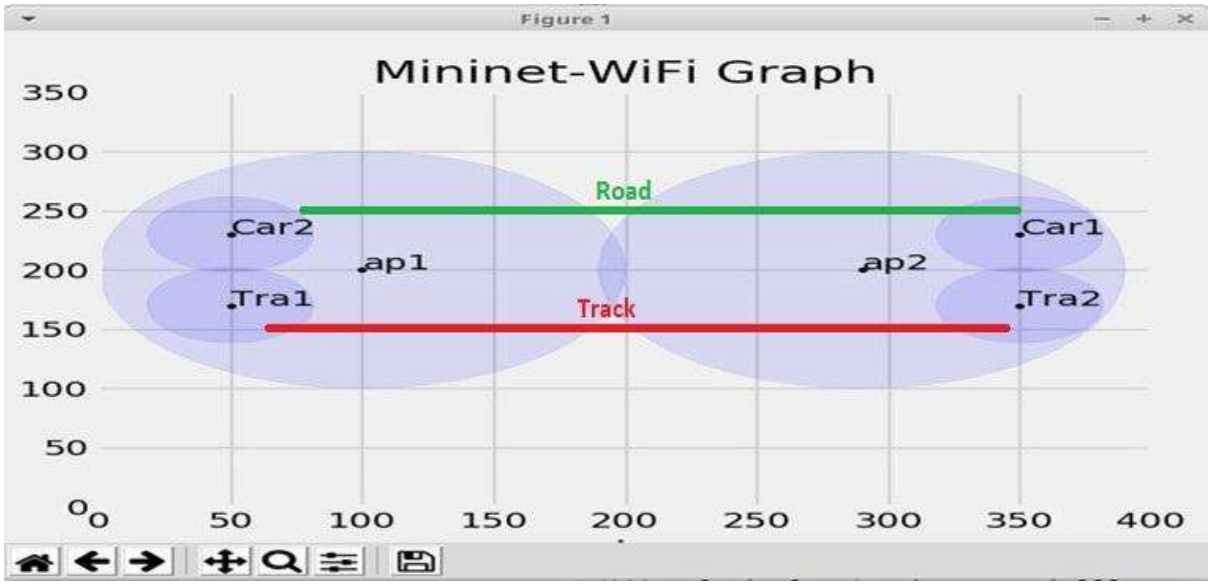

Figure A34: S4(5/6)1 Hosts and Access Points: Mininet-WiFi Graph (After Handover)

8.3.2 Reachability Test and Data Traffic Differentiation

This test is performed on all nodes and hosts connected to the network topology S4(5/6)1, and the results are presented in Table A3. As per the table, Cars can communicate only with other Cars and the designated road service server i.e., CarServer. Likewise, Trains can communicate solely with other Trains and the designated railway service server i.e., RailServer.

Table A3: Reachability Test

Src/Dst	Car1	Car2	CarServer	Tra1	Tra2	RailServer
Car1	✓	✓	✓	X	X	X
Car2	✓	✓	✓	X	X	X
CarServer	✓	✓	✓	X	X	X
Tra1	X	X	X	✓	✓	✓
Tra2	X	X	X	✓	✓	✓
RailServer	X	X	X	✓	✓	✓

8.3.3 UDP and TCP Transmission

This test is designed to showcase the standard data communication between cars and CarServer, as well as between trains and RailServer. The transmission of UDP data packets is depicted in Figure A35, while the transmission of TCP data packets is illustrated in Figure A36. In this scenario, Train1 functions as a client, and RailServer is set up as a listening server.

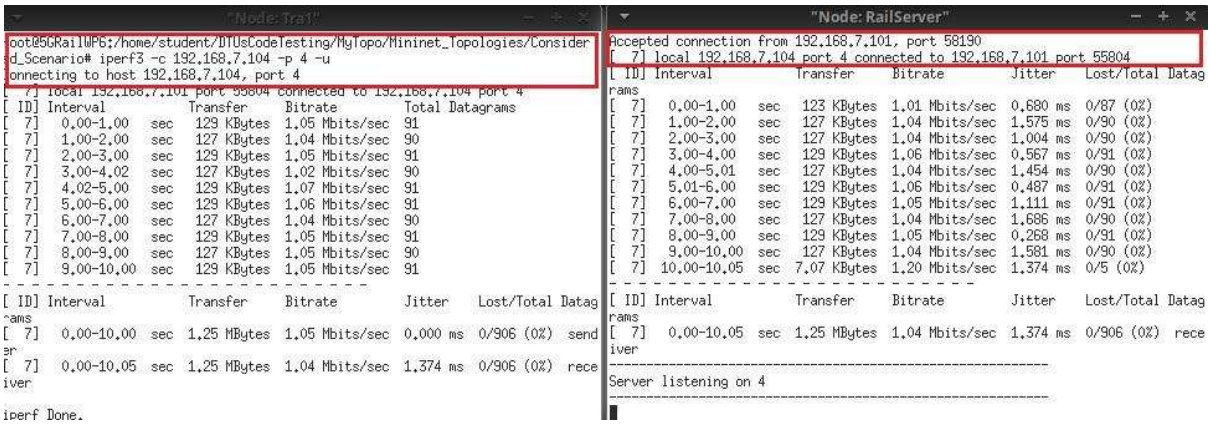

Figure A35: UDP Data Packet Transmission from Train1 to RailServer

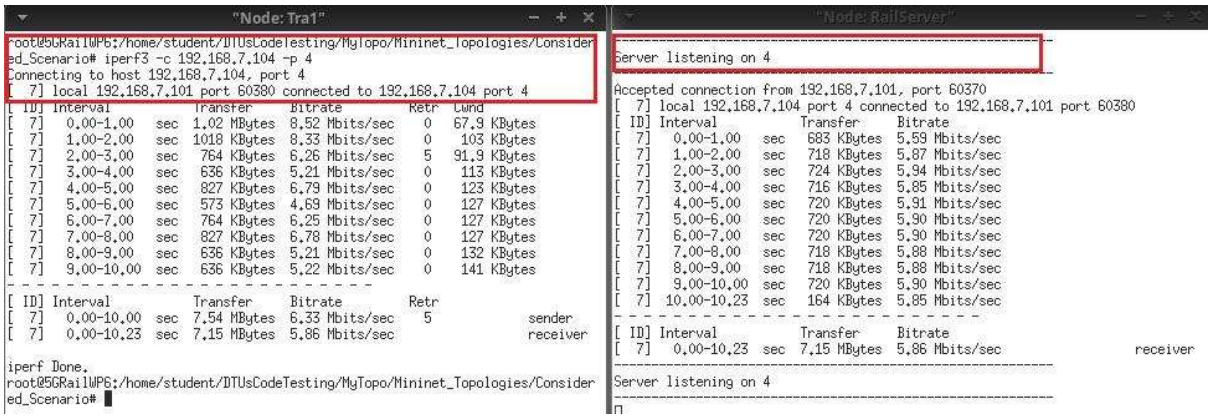

Figure A36: TCP Data Packet Transmission from Train1 to RailServer

8.3.4 Link Capacity Test

This test is carried out using the iperf tool to measure the bandwidth between two network links. To measure the bandwidth, the iperf <Host1> <Host2> command is used. Figure A37 shows the link capacity measurement between Car1 and CarServer and Tra1 and RailServer. The achieved bandwidth measurement shows that it is adequate to send and receive messages, voice, and video data for coexistence scenarios for roads and railways.

```
mininet-wifi> iperf Car1 CarServer
*** Iperf: testing TCP bandwidth between Car1 and CarServer
*** Results: ['5.75 Mbits/sec', '7.32 Mbits/sec']
mininet-wifi> iperf Tra1 RailServer
*** Iperf: testing TCP bandwidth between Tra1 and RailServer
*** Results: ['6.34 Mbits/sec', '7.99 Mbits/sec']
```

Figure A37: Link Capacity Test

8.3.5 Latency Test and Network Jitter Test

To measure losses, latency, and network jitter, the MTR tool is used. 100 UDP and TCP data packets are transmitted from Car1 to CarServer and Tra1 to RailServer to carry out the latency test. For this network topology, the latency ranges from 4.9 to 5.3 milliseconds, as shown in Figures A38 and A39. Moreover, Figures A40 and A41 exhibit that the network jitter ranges from 4.3 to 6.1 milliseconds.

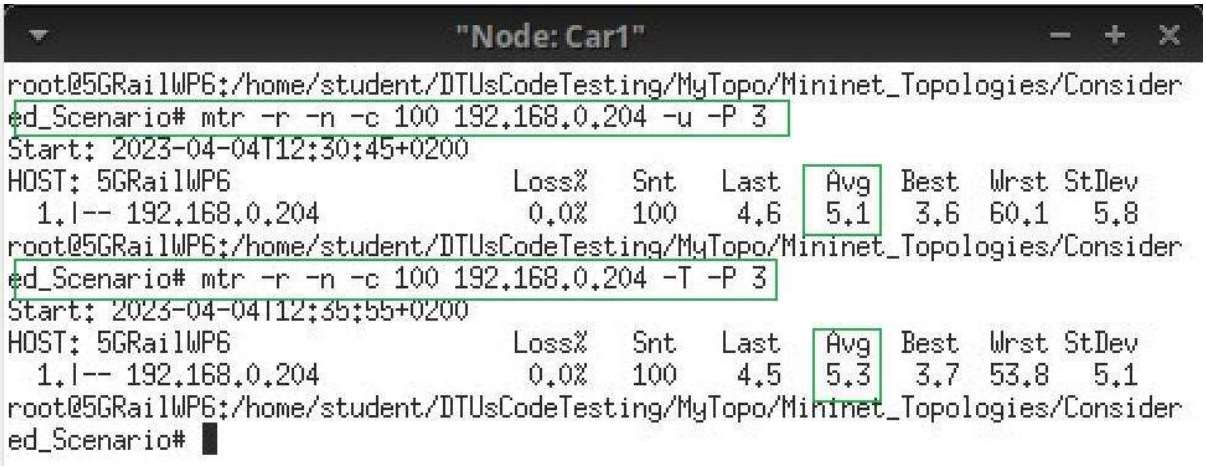

Figure A38: Latency Test from Car1

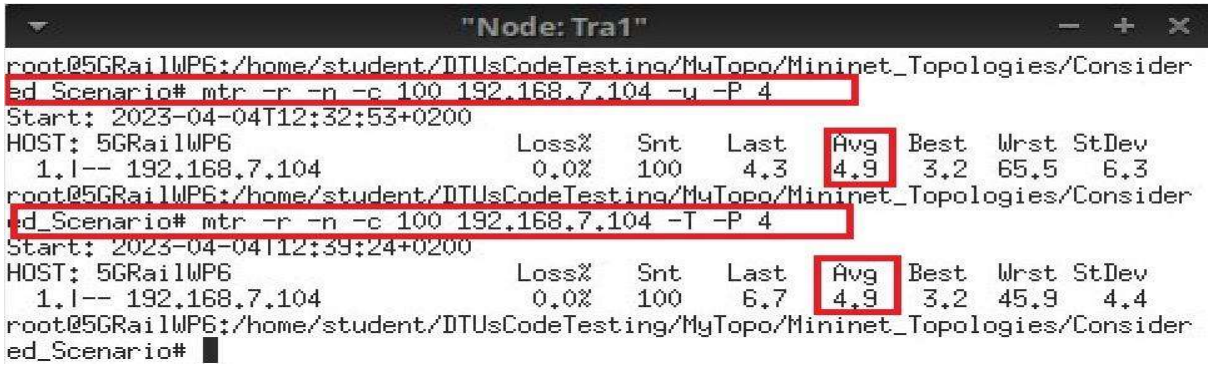

Figure A39: Latency Test from Tra1

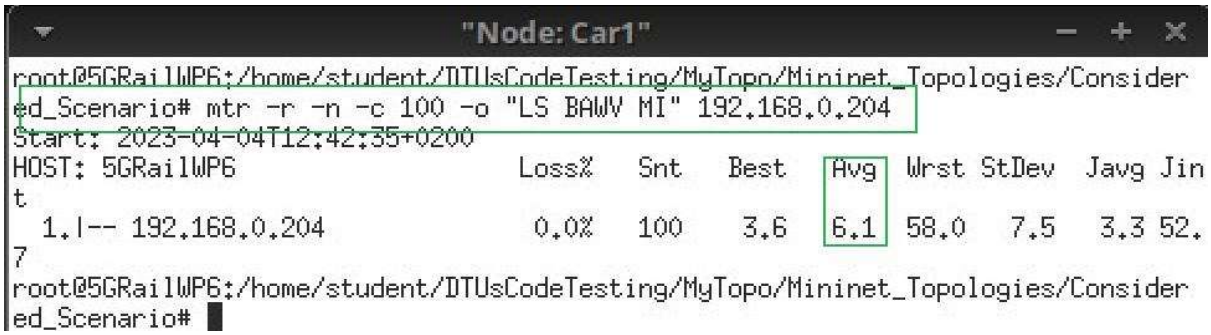

Figure A40: Network Jitter from Car1

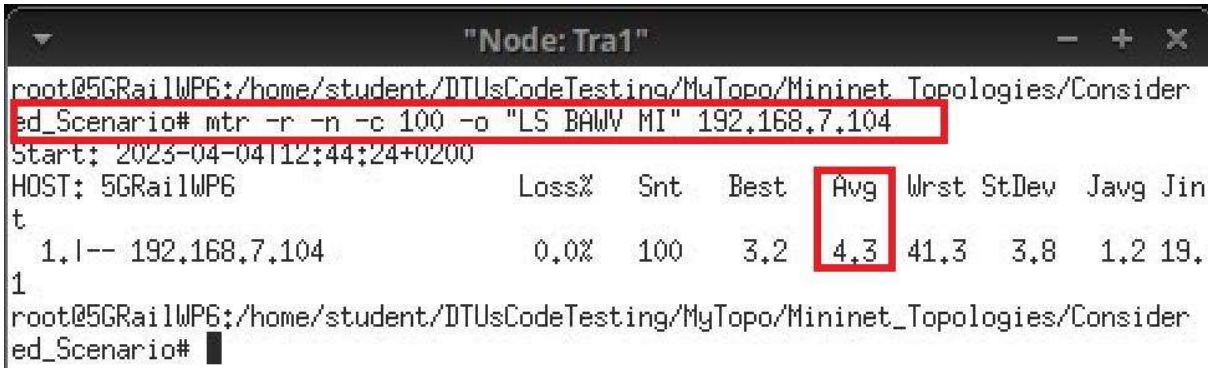

Figure A41: Network Jitter Test from Tra1

8.3.6 Sending a Message to the Assigned Server

The Scapy tool is a useful tool for transmitting messages and information in various scenarios. It allows users to send customized messages from any node or station to a designated service server.

Let's consider an example scenario where a message needs to be sent from Car1 to CarServer. To accomplish this, we can create an ICMP data packet with the message "Msg: Car1 is running with Speed 60 Km/hr" using the Scapy Python API within a Python script. Figure A42 illustrates the message creation process using Scapy. Finally, we can use the Wireshark tool to capture the data packet, as depicted in Figure A43.

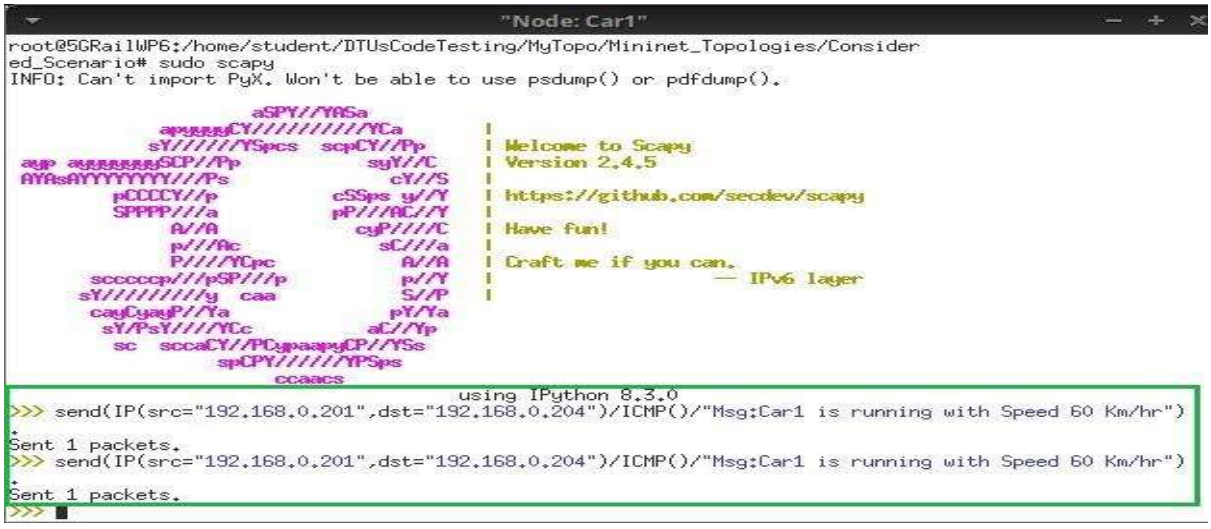

Figure A42: Scapy: Message Creation from Car1 to CarServer

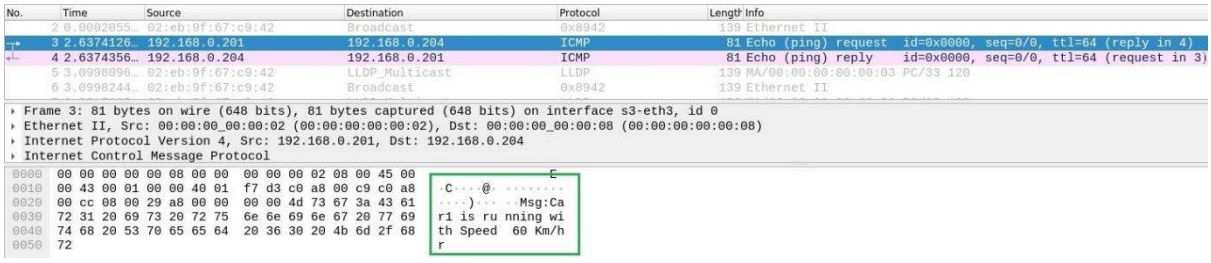

Figure A43: Wireshark: Scapy Packet with a Message

Similarly, using the Scapy a message is sent from Tra1 to RailServer with the message “Tra1 is running on Time” as shown in Figure A44. This data packet is captured at RailServer using the Wireshark tool shown in Figure A45.

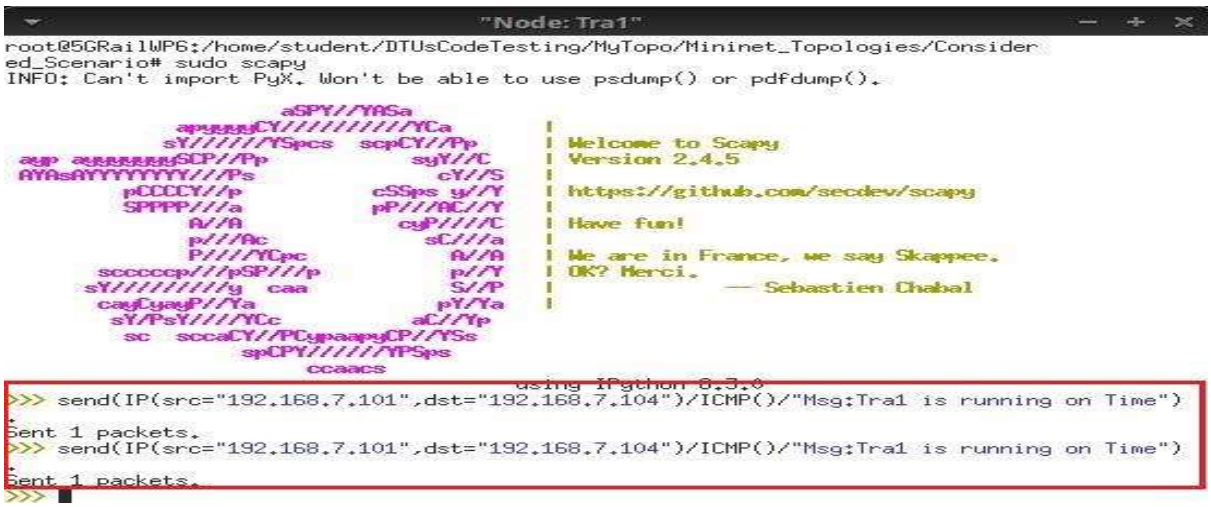

Figure A44: Scapy: Message Creation from Tra1 to RailServer

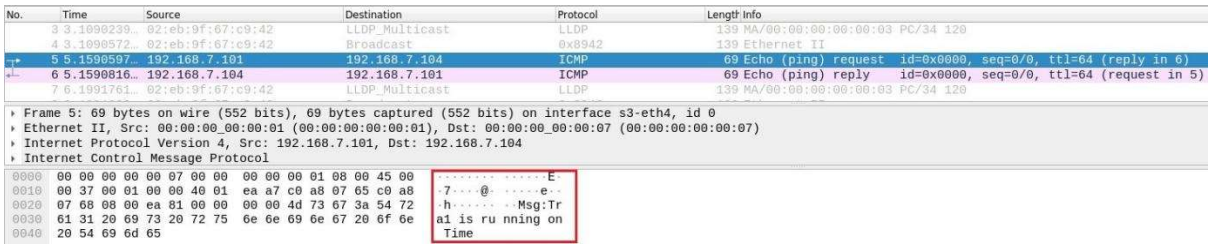

Figure A45: Wireshark: Scapy Packet with a Message

8.4 Implementation and tests for Coexistence Scenario 5

S4(5/6)4: Shared Access Network and Shared Core, Single Serving Technology, Track Perpendicular to Road: In this considered scenario, the network deployment infrastructures are similar to those of scenario S4(5/6)1, but in this case, railway tracks are kept perpendicular to roads.

8.4.1 Handover/Moving

Car1 and Tra1 have been configured with mobility capabilities to analyse handover and mobility. After 60 seconds, Car1 starts moving towards access point ap2, while Tra1 moves towards access point ap4. During their movement, Car1 pings Car2 and Tra1 pings Tra2. Figure A46 depicts the results of the ping test, which show that as Car1 reaches the edge of access point ap1 and enters the coverage range of access point ap2, it switches its connection from ap1 to ap3. Similarly, when Tra1 enters the coverage range of access point ap4, it switches its connection from ap3 to ap4.

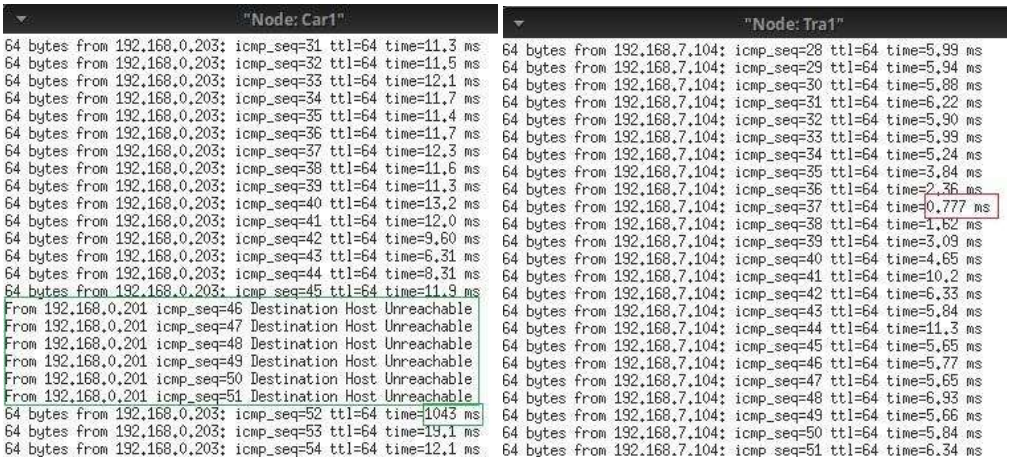

Figure A46: Checking Connectivity During Moving

To verify the handover and mobility functionality of the nodes, both Car1 and Tra1 are pinging their respective service servers. Figure A46 demonstrates that when Car1 and Tra1 move beyond the coverage range of their original access points, they automatically connect to the nearest access point (ap2 for Car1 and ap4 for Tra1). The figure also shows that the handover process is smooth with no packet loss.

To further confirm the handover and mobility capability of the nodes, the command "Car1 iw dev Car1-wlan0 link" is executed for Car1, and "Tra1 iw dev Tra1-wlan0 link" is executed for Tra1, both before and after the nodes' movement. Figure A47 illustrates that Car1 was initially connected to ap1, but after the handover, it successfully switched to ap2. Similarly, Figure A48 demonstrates that Tra1 was initially connected to ap3, but after the movement, it successfully switched to ap4.

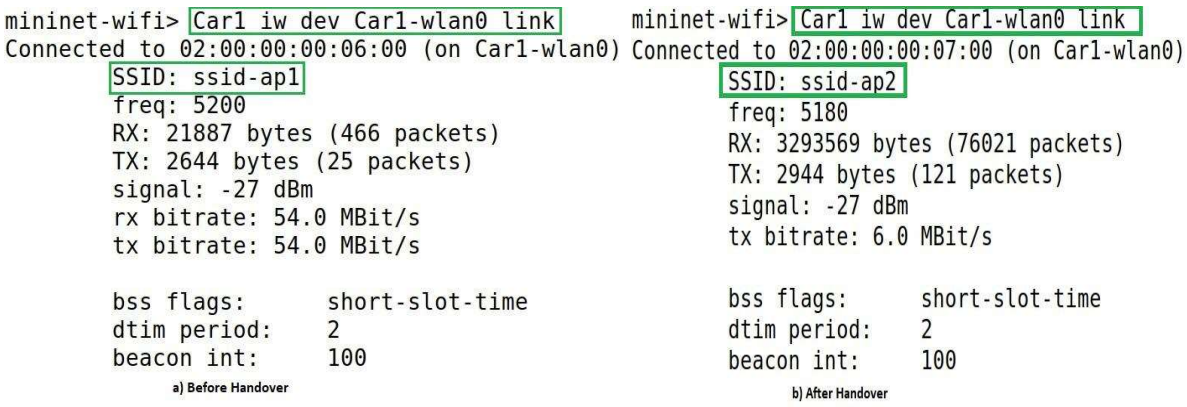

Figure A47: Connected Access Point for Car1 Before and After Handover/Moving

```
mininet-wifi> Tra1 iw dev Tra1-wlan0 link
Connected to 02:00:00:00:06:00 (on Tra1-wlan0)
        SSID: ssid-ap1
        freq: 5200
        RX: 9776 bytes (209 packets)
        TX: 1192 bytes (12 packets)
        signal: -27 dBm
        rx bitrate: 54.0 MBit/s
        tx bitrate: 54.0 MBit/s

        bss flags:      short-slot-time
        dtim period:    2
        beacon int:     100
```

```
mininet-wifi> Tra1 iw dev Tra1-wlan0 link
Connected to 02:00:00:00:06:00 (on Tra1-wlan0)
        SSID: ssid-ap1
        freq: 5200
        RX: 3388285 bytes (78190 packets)
        TX: 4012 bytes (135 packets)
        signal: -27 dBm
        rx bitrate: 48.0 MBit/s
        tx bitrate: 6.0 MBit/s

        bss flags:      short-slot-time
        dtim period:    2
        beacon int:     100
```

Figure A48: Connected Access Point for Tra1 Before and After Handover/Moving

Figure A49 and A50 show the updated positions after the nodes moved and handover took place. The comparison of these figures demonstrates that Mininet-WiFi is capable of simulating mobility and handover scenarios in railway and road environments effectively. Although there may be a delay during the handover process due to the network joining procedure carried out by the nodes or stations, there is no recorded data loss.

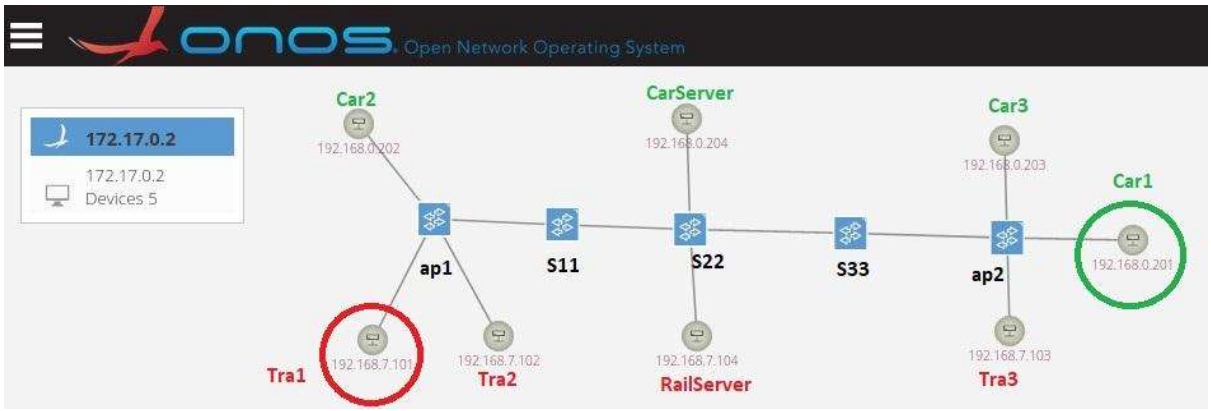

Figure A49: S4(5/6)4 Shared Access Network and Shared Core, Track Perpendicular to Road: ONOS Screenshot (After Handover)

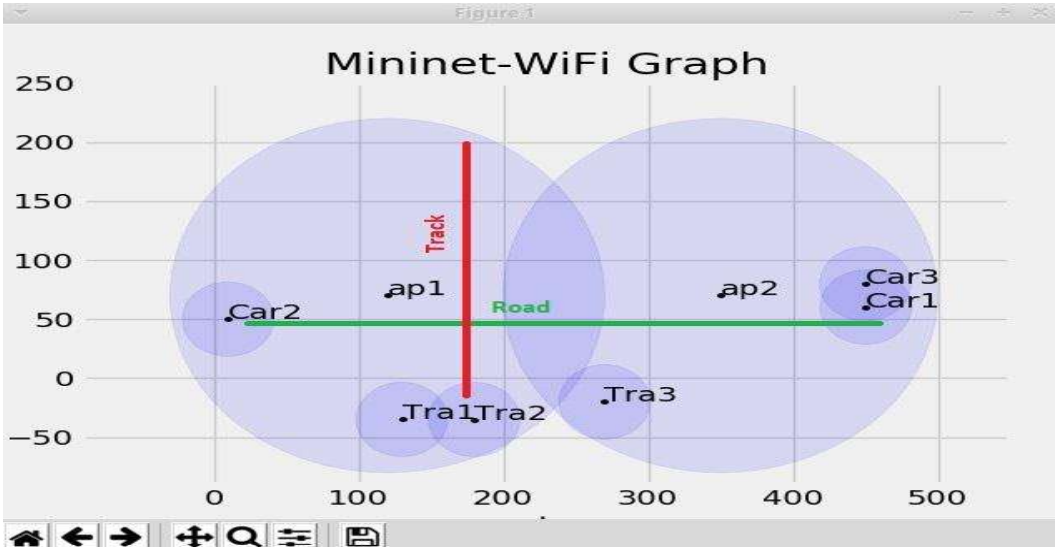

Figure A50: S4(5/6)4 Hosts and Access Points: Mininet-WiFi Graph (After Handover)

8.4.2 Reachability Test and Data Traffic Differentiation

For all nodes and hosts linked to the network topology S4(5/6)4, this test is conducted, and the findings are displayed in Table A4. According to the table, Cars can communicate solely with other Cars and the assigned road service server i.e., CarServer. Similarly, Trains can communicate only with other Trains and the designated railway service server i.e., RailServer.

Table A4: Reachability Test

Src/Dst	Car1	Car2	Car3	CarServer	Tra1	Tra2	Tra3	RailServer
Car1	ü	ü	ü	ü	X	X	X	X
Car2	ü	ü	ü	ü	X	X	X	X
Car3	ü	ü	ü	ü	X	X	X	X
CarServer	ü	ü	ü	ü	X	X	X	X
Tra1	X	X	X	X	ü	ü	ü	ü
Tra2	X	X	X	X	ü	ü	ü	ü
Tra3	X	X	X	X	ü	ü	ü	ü
RailServer	X	X	X	X	ü	ü	ü	ü

8.4.3 UDP and TCP Transmission

The purpose of this test is to demonstrate the typical data communication between cars and CarServer, as well as between trains and RailServer. Figure A51 illustrates the transmission of UDP data packets, while Figure A52 shows the transmission of TCP data packets from Tra1 to RailServer. In this scenario, Tra1 acts as a client, while RailServer is configured as a listening server.

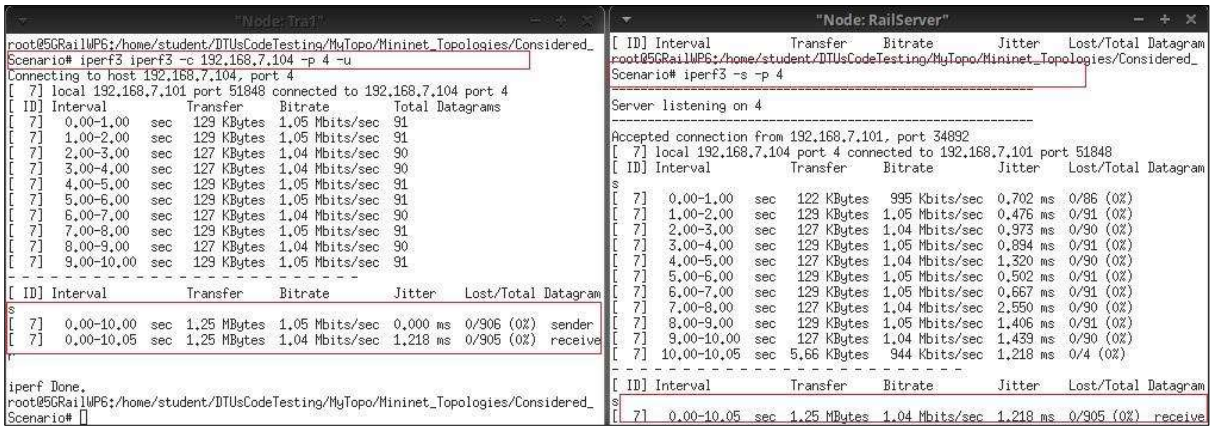

Figure A51: UDP Data Packet Transmission from Tra1 to RailServer

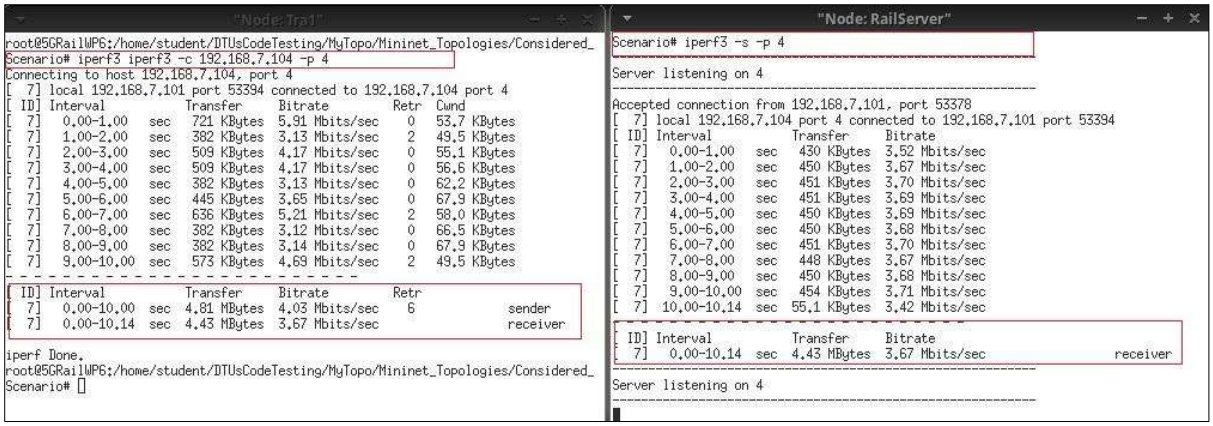

Figure A52: TCP Data Packet Transmission from Train1 to RailServer

8.4.4 Link Capacity Test

This test is carried out using the iperf tool to measure the bandwidth between two network links. To measure the bandwidth, the iperf <Host1> <Host2> command is used. Figure A53 shows the link capacity measurement between Car1 and CarServer and Tra1 and RailServer. The achieved bandwidth measurement shows that it is adequate to send and receive messages, voice, and video data for coexistence scenarios for roads and railways.

```
mininet-wifi> iperf Car1 CarServer
*** Iperf: testing TCP bandwidth between Car1 and CarServer
*** Results: ['3.53 Mbits/sec', '4.37 Mbits/sec']
mininet-wifi> iperf Tra1 RailServer
*** Iperf: testing TCP bandwidth between Tra1 and RailServer
*** Results: ['3.51 Mbits/sec', '4.32 Mbits/sec']
```

Figure A53: Link Capacity Test

8.4.5 Latency Test and Network Jitter Test

The MTR tool is utilized to measure losses, latency, and network jitter. To perform the latency test, 100 UDP and TCP data packets are transmitted from Car1 to CarServer and Tra1 to RailServer. For this network topology, the latency ranges between 7.2 to 17.7 milliseconds, as depicted in Figures A54 and A55. Additionally, Figures A56 and A57 demonstrate that the network jitter ranges from 7.1 to 7.2 milliseconds.

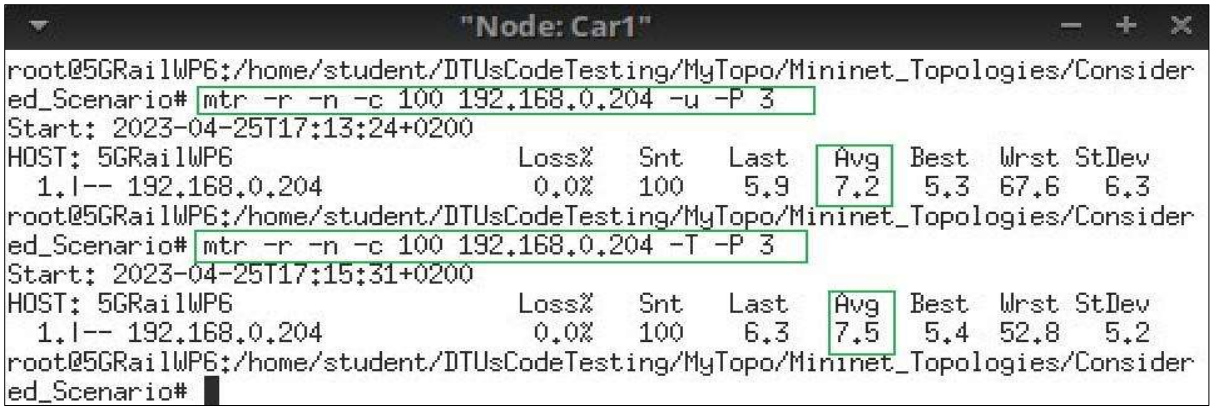

Figure A54: Latency Test from Car1

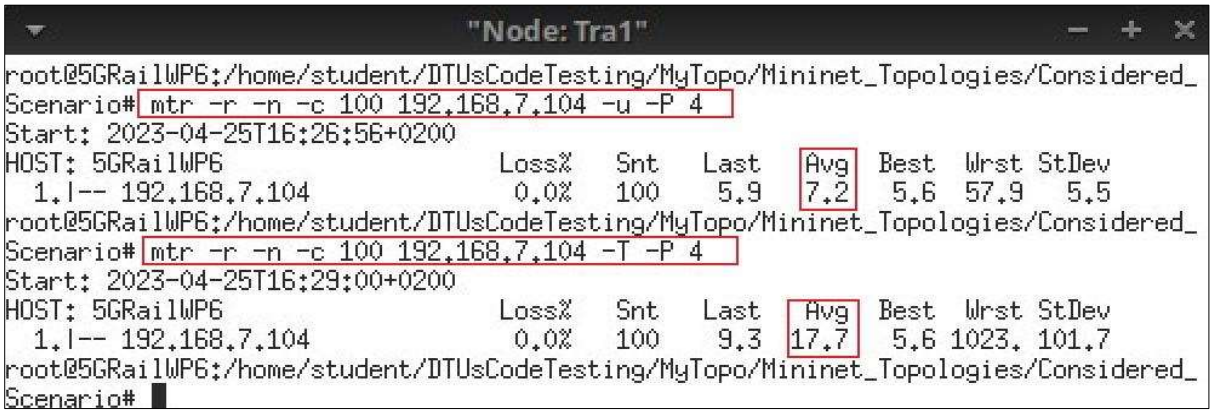

Figure A55: Latency Test from Tra1

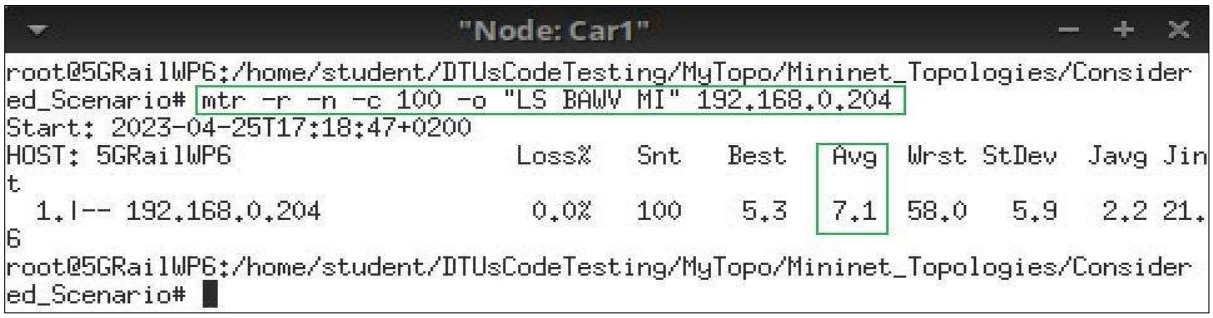

Figure A56: Network Jitter Test from Car1

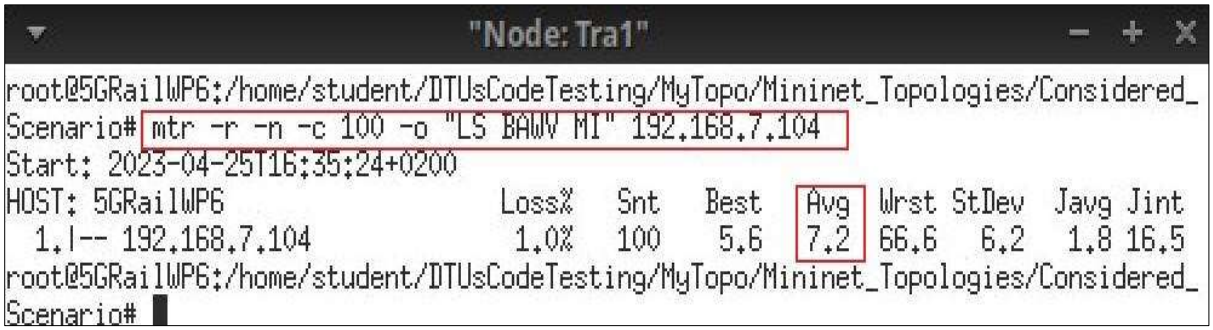

Figure A57: Network Jitter Test from Tra1

8.4.6 Sending a Message to the Assigned Server

The Scapy tool is a powerful tool for sending customized messages and information from any node or station to a designated service server. In various scenarios, users can utilize Scapy to create and transmit data packets efficiently.

For instance, consider an example scenario where a message needs to be sent from Car1 to CarServer. In this case, a user can create an ICMP data packet with the message "Msg: Car1 is running with Speed 60 Km/hr" using the Scapy Python API within a Python script, as shown in Figure A58. After that, the data packet can be captured using the Wireshark tool, as illustrated in Figure A59.

Similarly, Scapy can be used to send a message from Tra1 to RailServer with the message "Tra1 is running on Time", as demonstrated in Figure A60. The data packet can then be captured using Wireshark at RailServer, as depicted in Figure A61.

In summary, Scapy is a powerful tool for sending critical messages and information, while Wireshark offers a convenient way to capture and analyse these data packets in various scenarios.

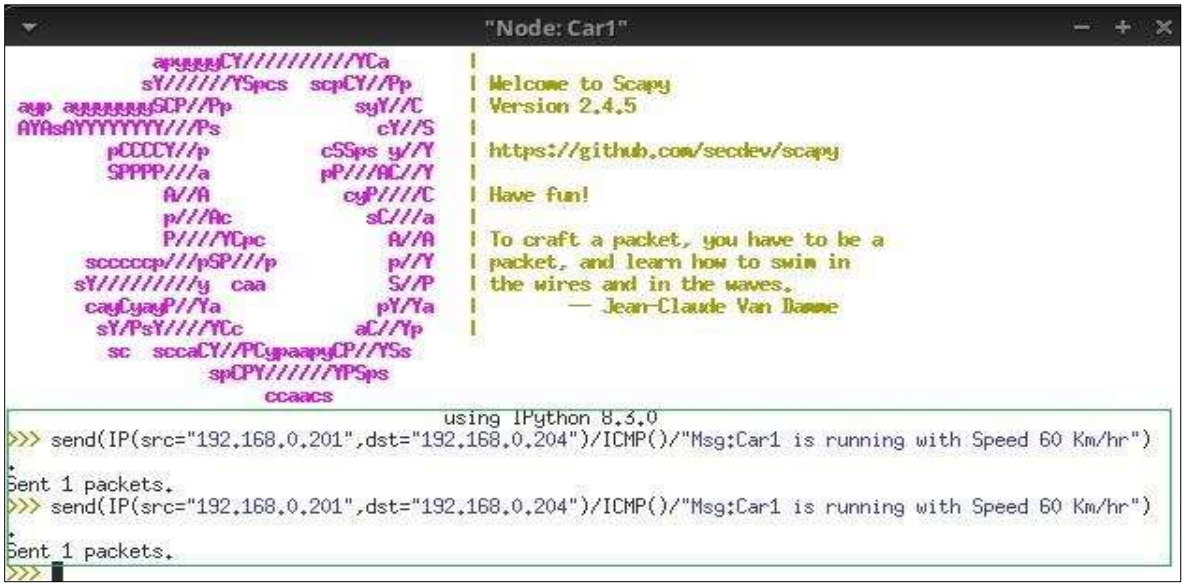

Figure A58: Scapy: Message Creation from Car1 to CarServer

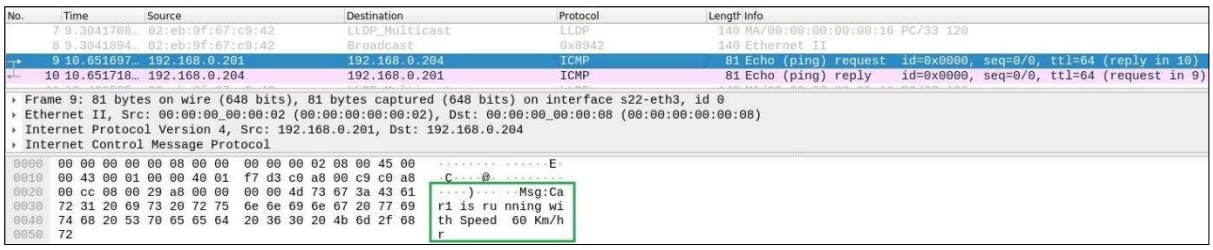

Figure A59: Wireshark: Scapy Packet with a Message

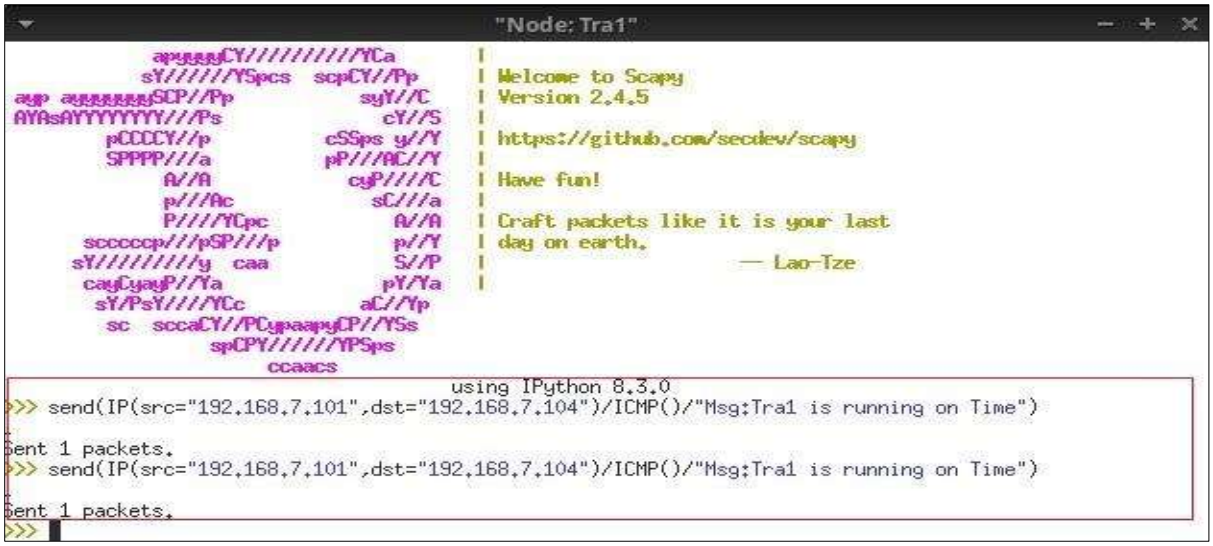

Figure A60: Scapy: Message Creation from Tra1 to RailServer