network

Tutorial

Coexistence of Railway and Road Services by Sharing Telecommunication Infrastructure Using SDN-Based Slicing: A Tutorial

Radheshyam Singh [1,†], José Soler [1,*,†], Tidiane Sylla [2], Leo Mendiboure [2] and Marion Berbineau [3]

1 Department of Electrical and Photonics Engineering, Technical University of Denmark, 2800 Kgs Lyngby, Denmark
2 COSYS-ERENA Lab, University Gustave Eiffel, IFSTTAR, 33067 Bordeaux, France
3 COSYS Department, University Gustave Eiffel, IFSTTAR, 59650 Villeneuve d'Ascq, France
* Correspondence: joss@fotonik.dtu.dk
† These authors contributed equally to this work.

Abstract: This paper provides a detailed tutorial to develop a sandbox to emulate coexistence scenarios for road and railway services in terms of sharing telecommunication infrastructure using software-defined network (SDN) capabilities. This paper provides detailed instructions for the creation of network topology using Mininet–WiFi that can mimic real-life coexistence scenarios between railways and roads. The network elements are programmed and controlled by the ONOS SDN controller. The developed SDN application can differentiate the data traffic from railways and roads. Data traffic differentiation is carried out using a VLAN tagging mechanism. Further, it also provides comprehensive information about the different tools that are used to generate the data traffic that can emulate messaging, video streaming, and critical data transmission of railway and road domains. It also provides the steps to use SUMO to represent the selected coexistence scenarios in a graphical way.

Keywords: railways; SDN; ONOS controller; handover; traffic differentiation; Mininet–WiFi; OpenFlow; VLAN

check for updates

Citation: 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

Academic Editor: Bin Han, Simon Pietro Romano, Patrick Seeling

Received: 17 October 2022
Accepted: 24 November 2022
Published: 1 December 2022

1. Introduction

The European Railway Traffic Management System (ERTMS) is considering available modern communication technologies to redefine the communication system for railways. The new communication system is based on multiple-access wireless technologies with 5G as main its target [1] to enhance service capabilities and safety. The Future Railway Mobile Communication System (FRMCS) [2] has defined the desired specifications and user requirements [3]. A number of use-cases, for instance: voice communication to/from the controller, public emergency communication, on-train safety, critical data sharing, remote control rail engine communication, etc., related to the FRMCS system are considered in [4]. To validate the FRMCS's considered requirements and use-cases, multiple tests and functional analyses are required while developing the prototypes [5].

The empirical work presented in this paper is part of a European Union project "EU H2020 ICT 5G for FRMCS (5GRAIL)" [6]. The goal of the 5GRAIL project is to create and test FRMCS ecosystem prototypes in order to validate the initial set of FRMCS requirements and standards (FRMCS V1). In this research, we emulate the telecommunication infrastructure for different scenarios for railway and road coexistence environments that are considered in Work Package 6 (WP6) [7] based on software-defined networks. Using SDN, networks can be configured dynamically and programmatically, and this mechanism improves the performance and monitoring of the network. The technology is more like cloud computing than traditional network management due to its ability to configure networks dynamically. The network elements are controlled by a centralized SDN controller [8].

The initial steps of this research are to find out the tools and technology that can be used to develop and emulate the different scenarios for railway and road coexistence. In addition, the primary objective of this paper is to provide a detailed tutorial to develop an SDN-based application using an ONOS SDN controller that should have the potential to manage the network handover and differentiate the data traffic from railways and roads. To differentiate the data traffic, SDN-based slicing is used. Over a common infrastructure, network slicing creates multiple end-to-end virtual networks. These virtual networks are logically isolated from each other and can be customized to serve different types of services with different requirements.

These are the considered objectives of Work Package 6 of the 5GRAIL project [7]. The second objective of this paper is to provide steps and logic to develop the network topology. The network topology is created in such a way that hosts should have the mobility function, i.e., hosts can move in the given direction at an assigned speed. The purpose of this functionality is to represent the hosts as cars and trains/trams. The third objective of this empirical work is to provide steps and procedures to represent the railway and road coexistence scenario in a pictorial way using SUMO tools. Along with these issues, the following questions are considered and analyzed to provide a fruitful outcome:

1. **Emulation:** How can emulation of trains and cars (mobility and handover scenarios) be performed in telecommunication infrastructure with the selected SDN-based tools for railway and road coexistence scenarios?
2. **Differentiation of Data Traffic:** How are the data traffic from railways and roads differentiated, or how is the SDN application designed and developed to perform SDN-based slicing to differentiate between data packets from railways and roads?
3. **Traffic Generation Tools:** Which tools are selected to generate different data traffic that can mimic the real-life data transmission scenario, and how can these tools be used to generate the intended data traffic?
4. **Assessment of Developed SDN Application and Tools:** Will the considered tools and technology have the potential to emulate the coexistence scenario of railways and roads?

This paper follows the following structure: Section 2 presents some previously carried out research. Section 3 describes the selected scenarios to develop the network topology. Information about the selected tools is given in Section 4. The setup description to carry out the test is presented in Section 5. Section 6 demonstrates network topology creation using Mininet–WiFi. The selected tools for data traffic generation are presented in Section 7. Section 8 elaborates on the developed ONOS SDN slicing application to differentiate the data traffic based on VLAN tagging. Validation of the developed SDN application and demonstration of selected tools are given in Section 9. SUMO integration for train and car mobility traffic generation and visualization purposes is presented in Section 10. The conclusion of this practical work is given in Section 11.

2. Related Work

In this section, some previously executed research and findings are given. In [9], the authors Mininet–WiFi to compare flow-based monitoring mechanisms for the Internet of Vehicles (IoV) environment. Performance comparison of open-source SDN controllers is presented in [10] to investigate the end-to-end delay for SDN-based vehicular networks. The authors in [11] developed a network prototype to define a car as a node with Mininet–WiFi to emulate the vehicular networks. The authors in [8] demonstrated the emulation of software-defined wireless networks using Mininet–WiFi. An emulation framework is presented using Mininet–WiFi in [12] for connected autonomous vehicles. A detailed survey for connected vehicles is presented in [13]. In the paper [14], the authors investigated the handover mechanism for communication at the intersection of the coverage range of the cells for vehicular networks. They introduced multiple-access edge computing (MEC) with a roadside unit (RSU) based on SDN to improve the handover process and curtail the handover time. In paper [15], the authors used a simulation tool known as

Mininet–IoT with a Ryu–SDN controller to create a vehicular ad hoc network (VANET). They also introduced a new interface based on the global positioning system (GPS) for cars to establish communication between cars when they are not in the coverage range of access points. This proposed system has better throughput and lower delay compared to WiFi–VANET. In [16], the authors emulated the 5G-NR for the vehicular network using Mininet–WiFi-Containernet and SUMO tools.

From the research mentioned above, it can be observed that narrow research or emulation work has been carried out in the field related to vehicular networks based on software-defined networks. Based on our analysis at the time of writing this paper, we did not find any related work that considered the emulation of the coexistence scenario of railways and roads based on SDN slicing to differentiate the data traffic to/from trains and cars by sharing a telecommunication infrastructure. We can say that this is one of the first studies where the authors have considered coexistence scenarios of railways and roads using SDN-based slicing. In addition, we present a detailed tutorial about the execution of selected tools.

3. Selecting and Defining the Scenarios

This section provides information about the considered scenarios for railway and road coexistence.

3.1. Essential Parameters to Define the Scenario

The selection of scenarios to demonstrate the coexistence of railways and roads is highly complex since it depends upon multiple parameters. If we consider telecommunication infrastructure sharing, radio access network sharing parameters, and elements that are coupled with network topology components that are presented in Table 1, then the following variables are taken into the consideration to define the scenarios:

(A) **Telecommunication Network Elements:** To define the railway and road coexistence scenario, telecommunication network parameters such as radio access network (RAN), backhaul, and core are considered as one vital parameter. Railway and road access networks can be shared or dedicated for both domains. Similarly, backhaul and core can be dedicated or shared [7].

Table 1. Considered combination cases for radio access and core network [7].

		Dedicated RAN	**Shared RAN**
Radio Access Network	**Single Technology**	R1	R2
	Multiple Technology	R3	R4
Core Network		C1	C2

Nomenclature used to represent the telecommunication infrastructure is given below:

- T1 R1C1: single-serving technology is used in the access network and each domain has its own dedicated RAN and its own dedicated core network.
- T2 R1C2: single-serving technology is used in the access network, each domain has its own dedicated RAN, and the core network is shared by both domains.
- T3 R2C1: single-serving technology is used in the access network and the RAN is shared by both domains, but both domains have their own dedicated core network.
- T4 R2C2: single-serving technology is used in the access network and each domain shares the access network and the core network.
- T5 R3C1: different-serving technology is used in the access network and each domain has its own dedicated RAN and its own dedicated core network.

- T6 R3C2: different-serving technology is used in the access network, each domain has its own dedicated RAN, and the core network is shared by both domains.
- T7 R4C1: different-serving technology is used in the access network and the RAN is shared by both domains, but both domains have their own dedicated core network.
- T8 R4C2: different-serving technology is used in the access network and each domain shares the access network and the core network.

(B) **Mobility Parameters:** The speed of the vehicles and the operating region of vehicles are also considered as shaping parameters to define the scenarios, e.g., train versus high-speed train versus urban train versus regional train versus highway versus road [7].
Nomenclature used to represent the telecommunication infrastructure is given below:

- M1: mobility of highway and tram;
- M2: mobility of highway and urban train;
- M3: mobility of highway and regional train;
- M4: mobility of highway and high-speed train;
- M5: mobility of road and tram;
- M6: mobility of road and urban train;
- M7: mobility of road and regional train;
- M8: mobility of road and high-speed train.

(C) **Topological Elements in Civil Engineering Infrastructure:** That deployed railway track components are parallel or perpendicular to roads is also an essential factor to define railway and road coexistence scenarios. This also includes transport infrastructure such as open places versus tunnels versus bridges [7].
Nomenclature used to represent the telecommunication infrastructure is given below:

- P1: railway tracks parallel to the road, open-air/bridge, the same plane;
- P2: railway tracks parallel to the road, open-air/bridge, different planes;
- P3: railway tracks parallel to the road, tunnel, same plane;
- P4: railway tracks perpendicular to the road, open-air/bridge, same plane (level crossing);
- P5: railway tracks perpendicular to the road, open-air/bridge, different planes;
- P6: railway tracks perpendicular to the road, tunnel, different planes.

(D) **Services:** Services and applications defined for railways and roads play an important role in defining the scenario. For instance, speed-limit-monitoring applications, traffic rule violation monitoring applications for vehicles, applications to monitor the railway tracks to locate cracks, etc.

3.2. Selected Scenario

Considering all the parameters and dependent variables mentioned above, approximately 400 scenarios can be defined related to railway and road coexistence scenarios [7]. For this practical work, five different feasible scenarios are considered that are designed, developed, and executed using the considered tools. The selected scenarios have nomenclature SX(Y)Z, where: S represents the term “Scenario”, X represents the “Telecommunication Network Parameters”, Y represents the “Mobility Parameters”, and Z represents the “Topological Elements in Civil Engineering Infrastructure”.

1. **S1(5/6)1: Different Access Network and Different Core, Single Serving Technology, Track Parallel to Road:** This scenario is considered 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 [7].

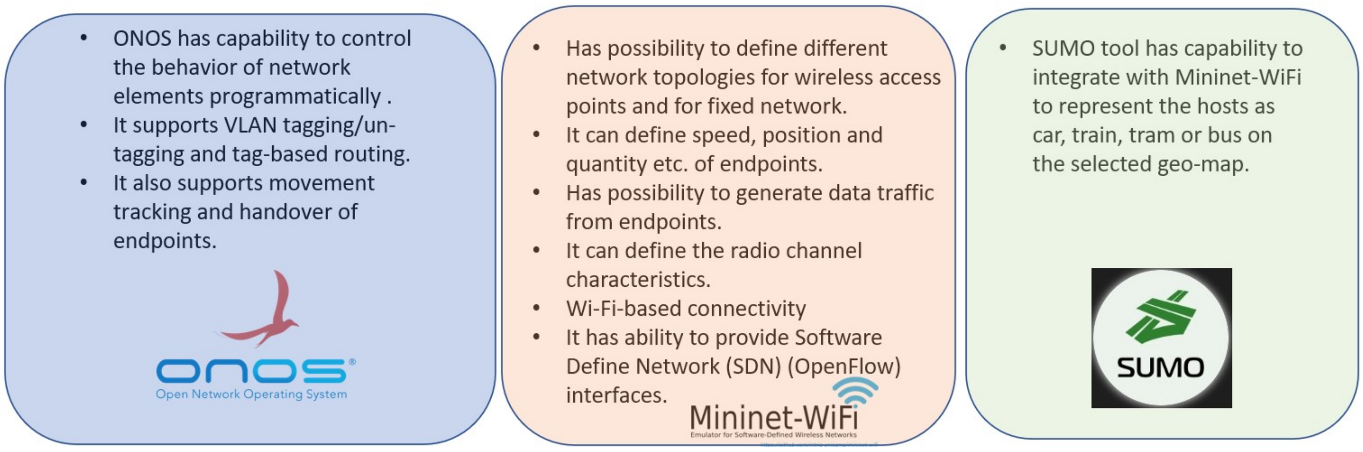
2. **S1(5/6)4: Different Access Network and Different Core, Single Serving Technology, Track Perpendicular to Road:** In this considered scenario, the network parameters are similar to scenario S1(5/6)1, i.e., railways and roads have dedicated radio access networks with dedicated cores, but in this scenario, railway tracks are perpendicular to roads [7].
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 [7].
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 [7].
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 [7].

4. Selected Tools

Based on the objectives presented in Section 1 for this research, the following are the key requirements that should be fulfilled by the selected tools:

1. The selected tools should have the capability to define the end nodes in such a way that the user can configure the mobility of the host in a selected direction with an assigned moving speed. The selected tool should also be able to define the quantity and frequency of the end points.
2. Using the selected tools, users should have the possibility to define multiple wireless network interfaces for end nodes.
3. Using the selected tools, there should be the possibility to generate different kinds of data traffic from the end nodes, compliant with different types of services: for example, messaging, critical data communication using messaging services or video streaming, and voice communication for operational purposes.
4. Radio channel characteristics can be defined to mimic real characteristics of radio links such as packet loss, network jitter, and delays.
5. The defined end nodes should have the capability to connect to an available WiFi-based network.
6. The selected tools should have the ability to define different network topologies for wireless access points as well as for fixed network entities and provide SDN-based OpenFlow interfaces. In addition, from a software-defined network perspective, a defined network component's behavior can be controlled and managed by a network controller.
7. To differentiate the data traffic from different network end nodes, selected tools should have the ability to support virtual local area network (VLAN)-based tagging/untagging, and they should support tag-based forwarding.
8. The selected tools have the ability to emulate cellular-based network connectivity, principally 5G.
9. It would be nice to have a tool that can graphically represent the network emulation for the different selected scenarios.

Based on the requirements considered above, the Open Network Operating System (ONOS) SDN controller [17] is selected to programmatically control the network topology, and Mininet–WiFi [18] is selected to define the network topology. The tool SUMO [19] is considered for the graphical representation of the selected scenario. Figure 1 shows the selected tools with their key properties that are considered to emulate the railway and road coexistence scenarios.



- ONOS has capability to control the behavior of network elements programmatically .
- It supports VLAN tagging/un-tagging and tag-based routing.
- It also supports movement tracking and handover of endpoints.

- Has possibility to define different network topologies for wireless access points and for fixed network.
- It can define speed, position and quantity etc. of endpoints.
- Has possibility to generate data traffic from endpoints.
- It can define the radio channel characteristics.
- Wi-Fi-based connectivity
- It has ability to provide Software Define Network (SDN) (OpenFlow) interfaces.

- SUMO tool has capability to integrate with Mininet-WiFi to represent the hosts as car, train, tram or bus on the selected geo-map.

ONOS
Open Network Operating System
Mininet-WiFi
Emulator for Software-Defined Wireless Networks
SUMO

Figure 1. Selected tools.

4.1. ONOS SDN Controller

The ONOS software-defined network controller is an open-sourced SDN and network function virtualization (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 [17].

4.2. Mininet–WiFi

Mininet–WiFi [18] is a software-defined network emulator. It is a branch of Mininet [20] embedded with additional functionalities such as the ability to define and configure WiFi access points and nodes with moving capability based on Linux wireless driver and simulation driver 80211_hwsim [21]. 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). 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 fulfills the requirements (1-7) 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 11). Mininet–WiFi installation files and processes are available at [18,22].

4.3. SUMO

Simulation of urban mobility, commonly known as SUMO [19,23], is an open-sourced traffic simulator used to design and visualize 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 [24,25] used SUMO for visualization, modeling, and defining nodes in traffic routes. Therefore, SUMO is considered to fulfill Requirement 9 mentioned above.

5. Setup Description

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 represents the movement of network nodes on an open street map.

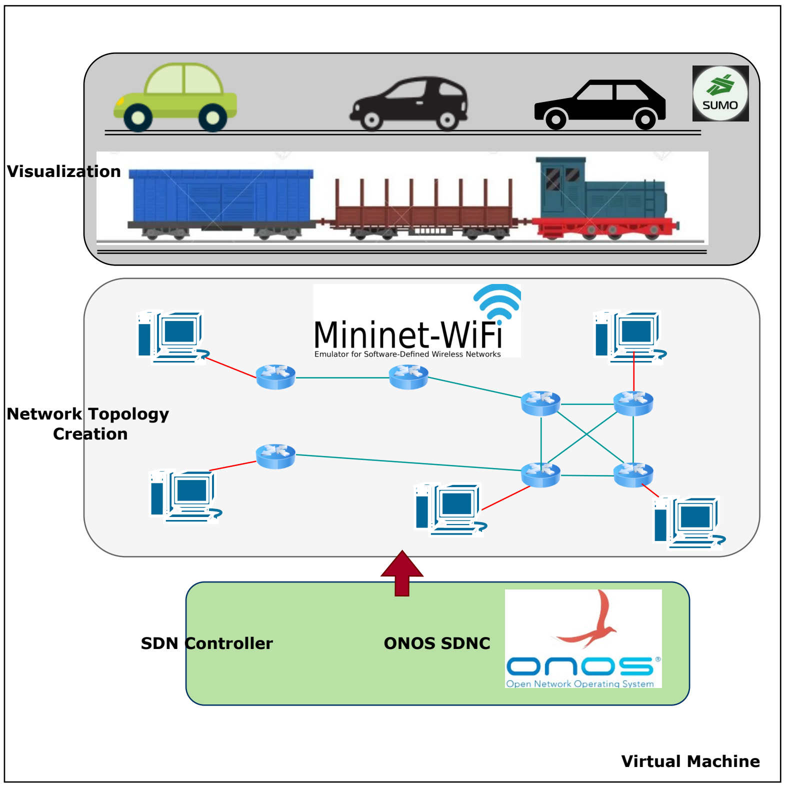

SUMO
Visualization
Mininet-WiFi
Emulator for Software-Defined Wireless Networks
Network Topology Creation
SDN Controller
ONOS SDNC
onos
Open Network Operating System
Virtual Machine

Figure 2. Test setup overview.

6. Network Topology Creation Using Mininet–WiFi

Before starting the validation of the selected tools and ONOS applications, the network topologies are created based on the scenarios explained in Section 3.2 for the coexistence of railway and road environments.

In this research, we implement and investigate all five scenarios presented in Section 3.2. For demonstration purposes to explain the tutorial to emulate the coexistence scenario of railways and roads, S2(5/6)1 and S4(5/6)4 scenarios are selected. The objective of this section is to provide all the necessary instructions to create the selected network topologies.

6.1. Network Topology S2(5/6)1—Different Access Network and Shared Core, Railway Track Parallel to Road

A Python script is written to create a network topology with Mininet–WiFi where trains and cars have different access networks, both domains shared the core network, and railways have parallel tracks to roads. Figure 3 shows the S2(5/6)1 scenario, where an ONOS SDN controller programmatically controls the forwarding elements of the topology. The network switches and access points are SDN-based devices and they are operated and controlled via OpenFlow protocol. Host Car1 represents a car, and host Train1 represents a rail. Access points ap1 and ap2 are defined for roads, and ap3 and ap4 are defined for railways. Access points ap1 and ap3 are connected to network switch S11, and ap2 and ap4 are connected to switch S33. The switch S22 is connected to S11 and S33. The host "RailServer" is defined as a railways service server, and "CarServer" is defined as a road service server; both servers are connected to switch S22. In this simple network topology, switch S22 is the core network switch, and S11 and S33 are the edge switches.

The network topology can be complex with more core and edge network elements. For the demonstration, a simple topology is selected.

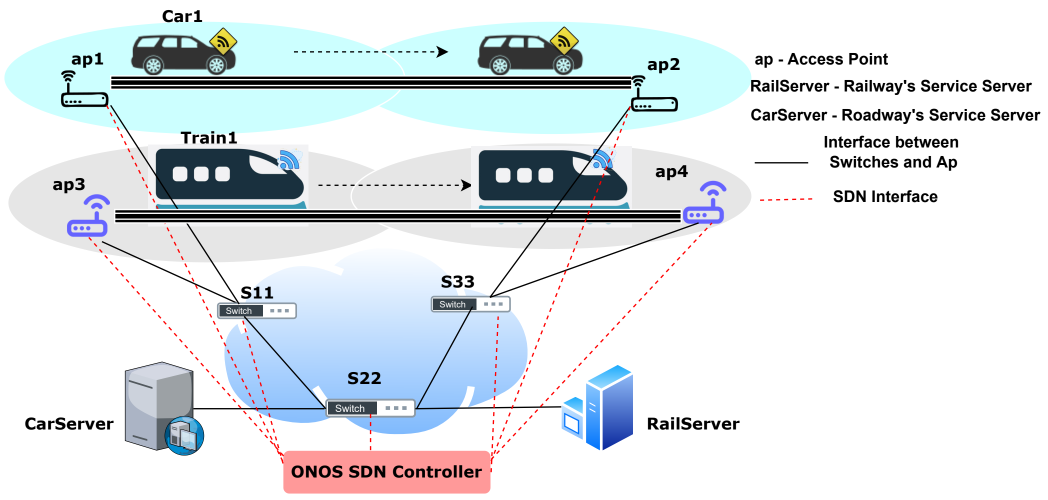

Car1
ap1
ap2
Train1
ap3
ap4
S11
S33
S22
Switch
CarServer
RailServer
ONOS SDN Controller
ap - Access Point
RailServer - Railway's Service Server
CarServer - Roadway's Service Server
Interface between Switches and Ap
SDN Interface

Figure 3. S2(5/6)1: Different Access Network and Shared Core, Track Parallel to Road.

In Figure 3, we can see that railways and roads have different access networks but a shared core network. To design this scenario, a topology development code is written in Python. The source code of this complete project is available at [26].

As we mentioned, for this practical work, the ONOS SDN controller is used. To define the ONOS as a remote controller as the element of the topology, the code given in Listing 1 is used as the Python topology code. To use the remote controller in Mininet–WiFi, port 6653 is assigned to the ONOS controller defined as "c1". Since Mininet–WiFi and the SDN controller are executed from the same virtual machine/laptop, IP '127.0.0.1' is assigned to the controller.

Listing 1. Adding Remote Controller ONOS.

```
''Create_a_network.''
net = Mininet_wifi( topo=None,
                   build=False,
                   ipBase='192.168.0.0/24')
info( '*** Adding_Remote_controller\n' )
c1=net.addController(name='c1',
    controller=RemoteController,
                   ip='127.0.0.1',protocol='tcp',port=6653)
```

After adding the ONOS controller, network switches and access points are defined. To define the switches, *net.addSwitch()* with "OpenFlow13" protocol is used, as shown in Listing 2. Using *net.addSwitch()*, three Open Virtual Kernel switches (OVS), S11, S22, and S33, are defined that work on "OpenFlow13" protocol. The command *net.addStation()* is used to define the cars and trains because it has the capability of defining the moving functionality and speed of the host. The hosts "CarServer" and "RailServer" are defined using the *net.addHost()*. The *args()* function is used to define the initial location of the moving nodes.

Listing 2. Defining the Moving Host and Server Host.

```
info( '***Add_switches\n')
s11 = net.addSwitch('s11',cls=OVSKernelSwitch,protocols="OpenFlow13")

info(''***Creating nodes\n'')
Train1_args, Car1_args = dict(), dict()
if '-s' in args:
    Train1_args['position'] , Car1_args['position']
    = '30,10,0', '30,320,0'

Train1 = net.addStation('Train1', mac='00:00:00:00:00:01',
        ip='192.168.7.101/24', position='30,10,0', **Train1_args)

Car1 = net.addStation('Car1' , mac='00:00:00:00:00:02',
        ip='192.168.0.201/24', position='30,320,0',**Car1_args)

info( '***Add_hosts_for_Service_Server\n')
CarServer = net.addHost('CarServer', cls=Host,
            ip='192.168.0.204/24', mac='00:00:00:00:00:08')
RailServer = net.addHost('RailServer', cls=Host,
            ip='192.168.7.104/24', mac='00:00:00:00:00:07')
```

To define the access points in Mininet–WiFi, *net.addAccessPoint()* is used as shown in Listing 3. It provides the ability to define and assign parameters such as the position of the access point, mode, operating channel, transmitting power in dBm, and coverage range in meters. Using a similar mechanism, four access points, ap1, ap2, ap3, and ap4, are defined for the selected scenario. The access points are assigned with different operating frequency ranges. The parameters *mode* and *channel* define the operating frequency range to broadcast and receive signals. The available modes for WiFi are b/g/n on a 2.4 GHz network and a/ac/n on a 5 GHz network [27]. The *net.setPropagationModel()* function is used with path loss exponent 5.

Listing 3. Defining the Access Point.

```
ap1 = net.addAccessPoint('ap1', ssid='ssid-ap1', mode='a',
        channel='36', position='100,300,0',
        'protocols':'OpenFlow13',
        'txpower':'49dBm','range': 110 )

net.setPropagationModel(model="logDistance", exp=5)

info(''***Configuring_wifi_nodes\n'')
net.configureWifiNodes()
```

After creating the network switches, hosts, and service server, the connectivity links are created using *net.addLink()* as shown in Listing 4. The parameter *bw* is used to define the bandwidth in Mbps of the connecting link. The line of codes present in Listing 4 create links between the hosts, switches, access points, and nodes. Access points ap1 and ap3 are connected to switch S11, and ap2 and ap4 are connected to switch S33.

Listing 4. Defining the connectivity.

```
info(''***Creating_links\n'')

s11s22 = {'bw':1000}
net.addLink (s11, s22, cls=TCLink , **s11s22)
s22s33 = {'bw':1000}
net.addLink (s22, s33, cls=TCLink , **s22s33)

CarServers22 ={'bw':500}
net.addLink (CarServer, s22, cls=TCLink , **CarServers22)
RailServers22 = {'bw':500}
net.addLink (RailServer, s22, cls=TCLink , **RailServers22)

s11ap1 = {'bw':500}
net.addLink (s11, ap1, cls=TCLink , **s11ap1)
s11ap3 = {'bw':500}
net.addLink (s11, ap3, cls=TCLink , **s11ap3)
s33ap2 = {'bw':500}
net.addLink (s33, ap2, cls=TCLink , **s33ap2)
s33ap4 = {'bw':500}
net.addLink (s33, ap4, cls=TCLink , **s33ap4)
```

Mininet–WiFi has a method *net.plotGraph()* to show the position of defined hosts and access points on a graph. Using this method, the user can define the range of x- and y-coordinates for the plot limits, as shown in Listing 5. Figure 4 shows the graphical representation of the selected topology; we can clearly see access points ap1, ap2, ap3, and ap4 and defined nodes. The developed network topology with assigned switches, access points, connected hosts, and nodes/stations can be seen on the ONOS graphical representation platform as shown in Figure 5. The network topology is built using *net.build()*, and access points are started using *start()*. Mininet–WiFi also provides a command line interface (CLI) using the function *CLI(net)*, which enables interaction with any element of the developed topology. To stop the network, the *net.stop()* function is used. In the code below, "c1" is the name assigned to the ONOS SDN controller.

Listing 5. Defining the Plot Limits of Graph and Start the Access Points and Switches.

```
if '-p' not in args:{
   net.plotGraph (max_x=450, min_x=-50, min_y=-50, max_y=450)}

 info(''***Starting_the_network\n'')
 net.build()
 info( '***Starting_the_controllers\n')
 for controller in net.controllers:
      controller.start()
 ap1.start([c1])
 ap2.start([c1])
 ap3.start([c1])
 ap4.start([c1])
 net.get('s11').start([c1])
 net.get('s22').start([c1])
 net.get('s33').start([c1])

 info(''***Starting_CLI\n'')
 CLI(net)
```

```
        info('***Stopping_network\n'')
        net.stop()
if __name__ == '__main__':
        setLogLevel('info')
        topology(sys.argv)
```

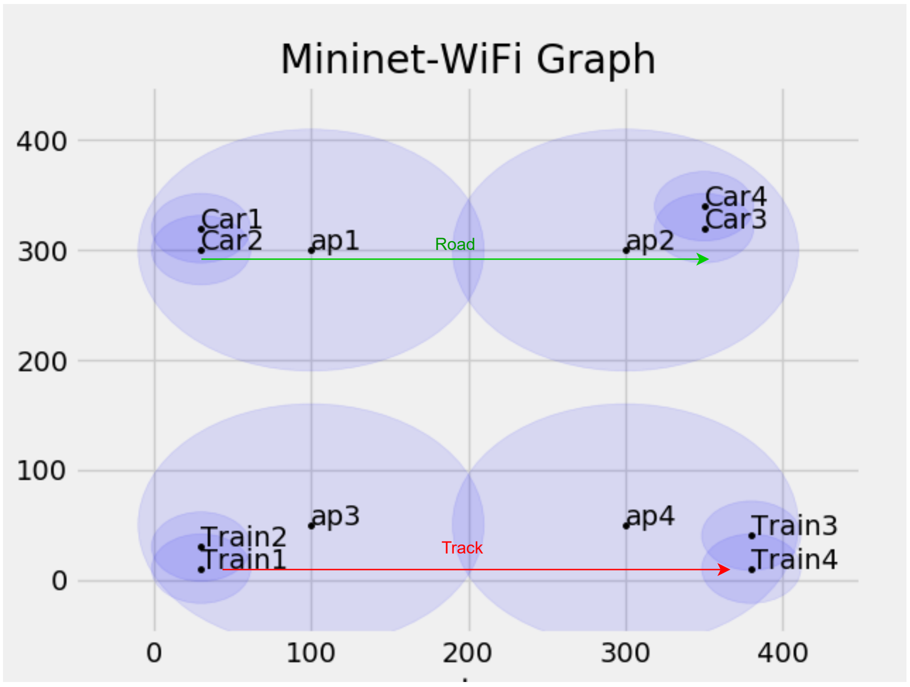


Figure 4. Hosts and access points: Mininet–WiFi graph.

Using the above-mentioned methods, functions, and procedures, users can emulate any salable and complex topology. Figure 5 represents the topology created using the Python script for the S2(5/6)1 scenario represented by the ONOS SDN controller's graphical user interface (GUI), where hosts Car1 and Train1 are defined with moving capability.

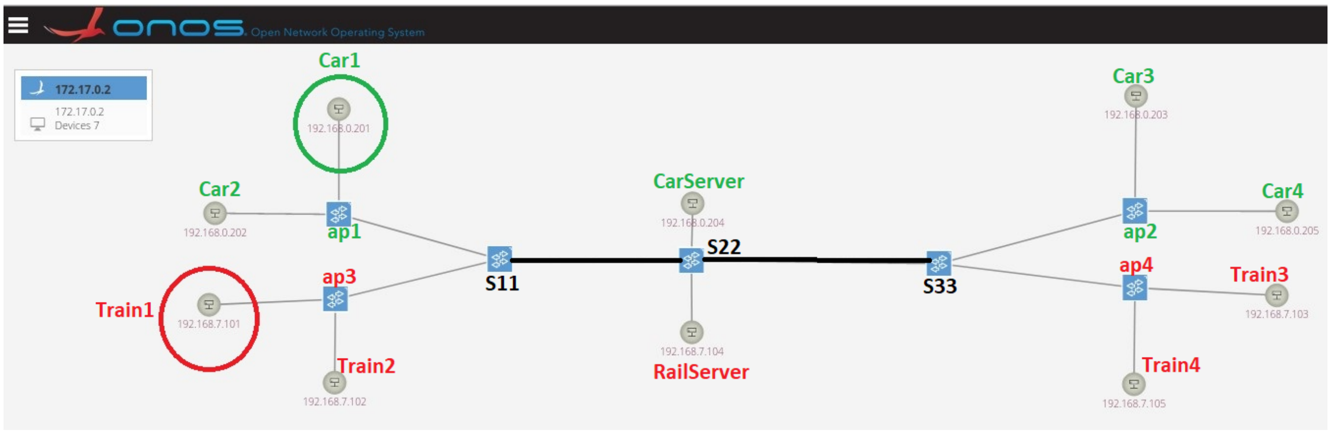


Figure 5. S2(5/6)1: Different Access Network and Shared Core, Track Parallel to Road Topology: ONOS screenshot.

6.2. *S4(5/6)4: Shared Access Network and Shared Core, Track Perpendicular to Road*

To design the S4(5/6)4 scenario, a similar Python script is written using the methods and functions mentioned in Section 6.1, and a network topology is developed in such a way that access points and the core network are shared by both railway and road domains.

In this selected scenario, railway tracks are perpendicular to roads. To emulate the railway tracks being perpendicular to the road, the y-coordinate of the moving node Car1 is kept constant (y = 70) while moving in the defined virtual space, and the x-coordinate of node Train1 is kept constant (x = 130) while moving in the defined virtual space, as shown in Figure 6.

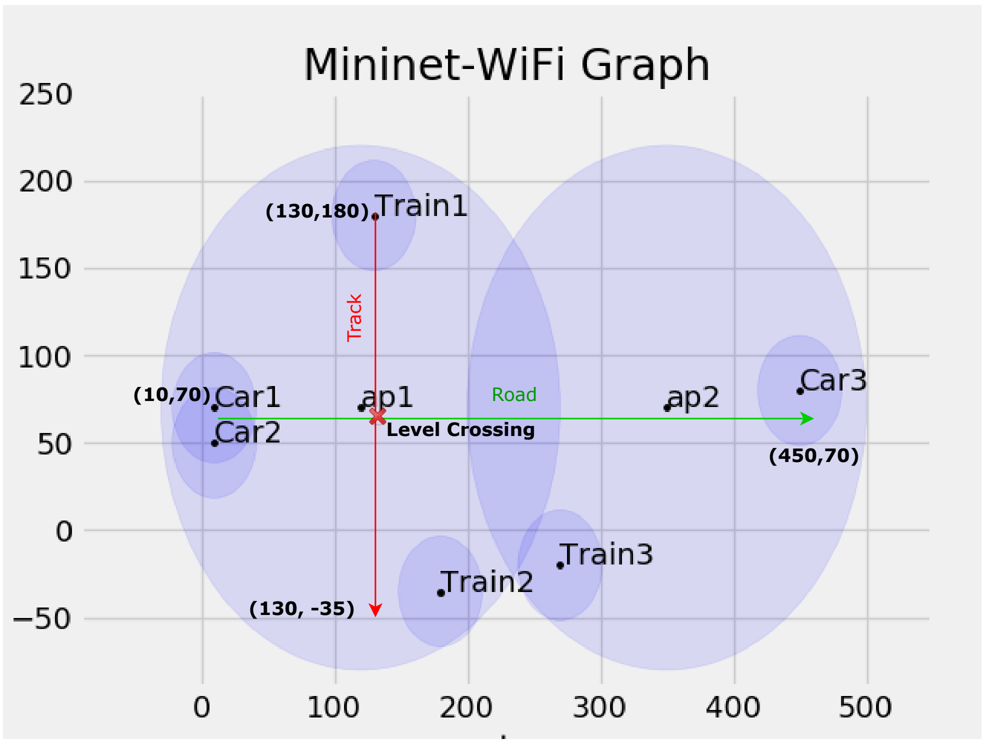


Figure 6. S4(5/6)4 hosts and access points: Mininet–WiFi graph.

A pictorial representation of this scenario is shown in Figure 7, where we can see that access points ap1 and ap2 are shared by both the domains, and the railway track is perpendicular to the road. This particular scenario for railway and road coexistence represents the "Level Crossing" at cross points.

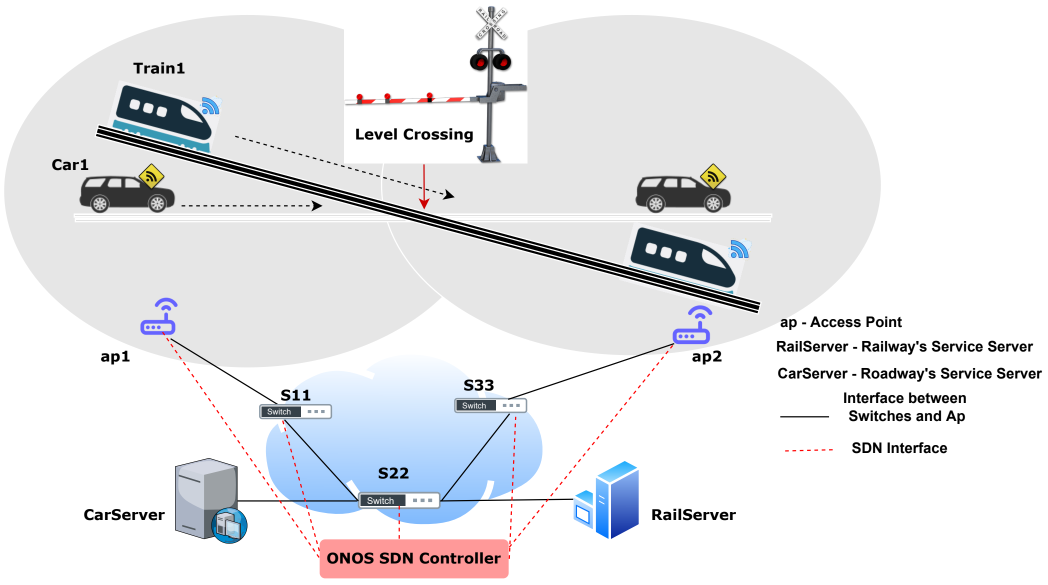


Figure 7. S4(5/6)4: Shared Access Network and Shared Core, Track Perpendicular to Road.

Figure 8 shows the network topology for scenario S4(5/6)4 created with Mininet–WiFi. Car1, Car2, Train1, and Train2 are connected to access point ap1, whereas Car3 and Train3 are connected to access point ap2. Hosts CarServer and RailServer are connected to switch S22. Similar to topology S2(5/6)1, in this topology, Car1 and Train1 are defined with moving functionality. Comparing Figures 5 and 8, we can see that in Figure 5 both domains have dedicated access networks, but in Figure 8, both domains, i.e., railways and roads, share the same access network.

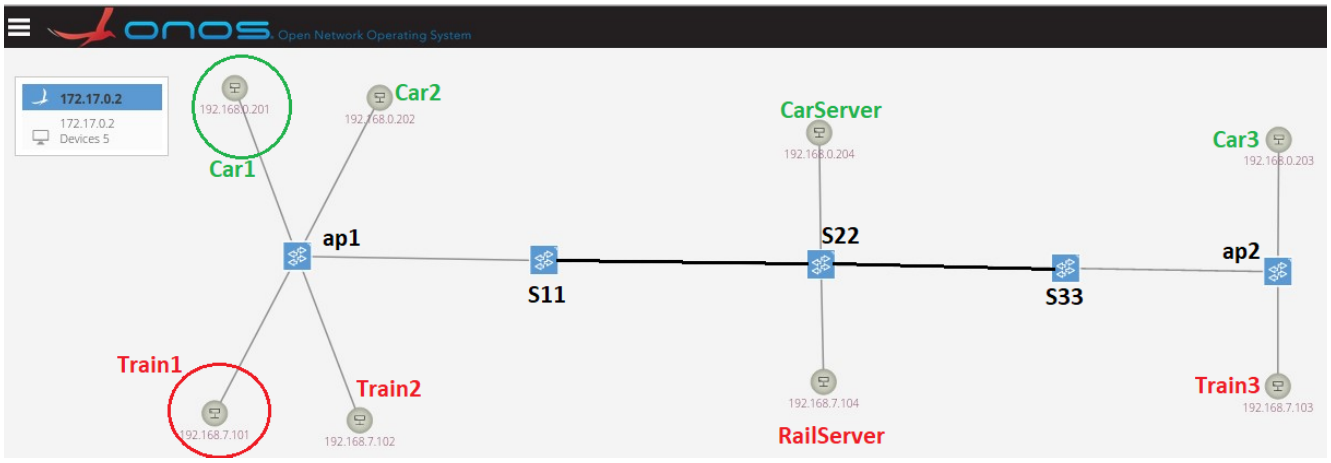


Figure 8. S4(5/6)4: Shared Access Network and Shared Core, Track Perpendicular to Road Topology: ONOS screenshot.

7. Selected Tools to Generate Data Traffic to Validate the Scenarios

To map the data traffic generation that can define the scenarios mentioned in Table 2, different tools are analyzed and selected. These mentioned scenarios are taken from the documentation of 5GRail project Deliverable D6.1 [7].

This section provides information about the selected tools that are used to generate the data traffic. Along with this, it also explains the procedure to install these tools and which parameters should be selected to generate the desired data streams.

Table 2. Selected tools to generate different kinds of data traffic for compliance with real-case scenario [7].

Scenario	Considered Tool to Demonstrate the Scenario	Tool Information
Voice Communication for Operational Purposes 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, 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.

The following tools are used and tested to generate different kinds of data traffic compliant with real-case scenarios:

1. **iperf3:** Iperf3 is a network performance measurement tool. It is able to execute on Linux, Unix, Android, macOS, and Windows platforms. It works in client and server mode functionality. It has the ability to generate user datagram protocol (UDP) and transmission control protocol (TCP) packets from one host/client and send them to another host/server. It generates a packet data stream to measure the throughput, bandwidth, and packet loss between two hosts [28,29].
   The installation file and documentation to use ipef3 are given in the link [29]. The iperf3 tool is used to send and receive UDP and TCP data packet streams to demonstrate voice communication and standard data communication for operational purposes between car-to-car and assigned car service servers or train-to-train and assigned rail service servers. Table 3 shows the commands used to generate the UDP/TCP data traffic from the client host to the server host.
2. **Scapy:** Scapy is an interactive packet manipulation tool with a CLI, and its API is implemented using Python. Therefore, this tool can be used through CLI or by importing it to Python scripts. Using this tool, users can create, forge, or decode packets from a wide range of protocols and send them to the network. It can also capture data packets, match criteria, and reply to requests [30]. Along with these functions, most classical tasks can be handled easily, including scanning, probing, and detecting networks. Scapy has the ability to run on Linux, macOS, and Windows systems. The principal advantage of Scapy is that it offers a technique to modify and create network packages at a low level by leveraging available network protocols and configuring them based on the user's needs [31]. Therefore, Scapy is used in this practical work to create a data packet to indicate critical data communication. In addition, it is also used to create special messages associated with emergencies or to convey any kind of information to the rail service server or car service server.
   Scapy is installed in a Linux environment using the specific instances shown in the Table 4. A demonstration of using Scapy to generate data packets associated with critical data communication or messages is presented in Section 9.7 of this paper.

Table 3. Iperf3 commands to generate UDP/TCP data packets [29].

Command	Explanation
1. iperf3 [-s | -c] [options] 2. iperf3 –s –p <port number> 3. iperf3 –c <server IP/host IP> -u –p <port number> -b <bitrate>	Where **-s** represents host as server, **-c** represents host as client, **–u** represents UDP packets, **-b** is used to set the bitrate, **-i** option allows setting the reporting interval time in seconds; e.g., iperf3 -c 10.0.0.7 -i 2

Table 4. Scapy installation commands [32].

Options	Linux Command
1. Install default Scapy	pip install scapy
2. Install Scapy with Python	pip install - -pre scapy[basic]
3. Install Scapy with dependencies	pip install - -pre scapy[complete]

3. **VLC Player:** VideoLAN Client, generally known as VLC, is open-source, compact media player software developed by the VideoLAN project. VLC is capable of streaming and receiving media files (video/audio). In addition to DVD video and video CD formats, VLC supports a wide variety of audio and video compression methods and media streaming protocols. It can be installed and used on any desktop system as well as on smartphones [33,34].
   To demonstrate video or critical video communication in railway and road coexistence scenarios, VLC Player is used to transmit video from a train to the RailServer and

from a car to the CarServer. A detailed process for streaming a video from one host to another or to a local system is presented in Section 9.8 of this paper.

4. **MTR:** Matt's Traceroute (MTR) is a network performance monitoring tool. Ping is one of the most widely used network diagnostics tools to figure out network reachability from one host/network entity to another. Using ping, a user sends ICMP packets from one system/host to another, and after getting the ICMP packets, the destination host/system sends an echo reply. This echo reply informs about the availability of the destination host, end-to-end delay, and packet loss, whereas the traceroute provides information about the path between the sender and receiver host. MTR utilizes both ping and traceroute functionality [35,36].
   The significant reason to use the MTR tool is that it allows measurement of the data packet loss and jitter of the network. Therefore, we can say that MTR measures the quality of the network path. For installing the MTR in the Ubuntu environment, use the following command: *sudo apt-get -y install mtr*. Table 5 shows the commands and options to measure the loss and jitter of the network. There are more options available to measure packet data losses and jitter with the mentioned commands, which can be found in [35].

Table 5. MTR data traffic generation to measure the loss and jitter of networks [35].

Command	Available Options	Use
1. mtr –r –n –c <number of data packets> -T –P <port number> <server IP/Host IP>	Where **r** reports print; **c** defines the number of Packets; **n** disables the DNS or no DNS option; **T** shows TCP data; **u** shows UDP data;	Calculate the loss in percentage
2. mtr –r –n –o "L BAWV MI" <server IP/Host IP>	Where **L** shows loss ratio; **B** shows min/best RTT (ms); **A** shows avg RTT (ms); **W** shows Mmx/worst RTT (ms); **V** shows standard deviation; **M** shows jitter mean/avg; **I** shows interarrival jitter	Calculate the jitter in ms

8. 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 objective:

- 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.

Keeping the above-mentioned objectives in consideration, an SDN data-forwarding application is developed using ONOS JAVA APIs [37] and deployed in the ONOS controller. The developed ONOS forwarding application has the potential to enable network slicing and to differentiate the data traffic to/from railways and roads. This implies that trains can only communicate with trains and assigned rail service servers; similarly, cars can only communicate with other cars and assigned car service servers. Along with this, additional functionality is added to manage the moving and handover of nodes between the assigned access points/cells. The application decides whether a data packet should be forwarded or dropped between the nodes, switches, and access points. Figure 9 represents the different steps of the developed ONOS application. After installing and activating the application, it initializes the "packet processor". The packet processor is an ONOS API that allows defining of the header context of packets and activates the developed applications. Two arrays are created in the application that contain the IP addresses of hosts and nodes. The first array contains the IP addresses of all cars, and the second array contains the IP addresses of all rails.

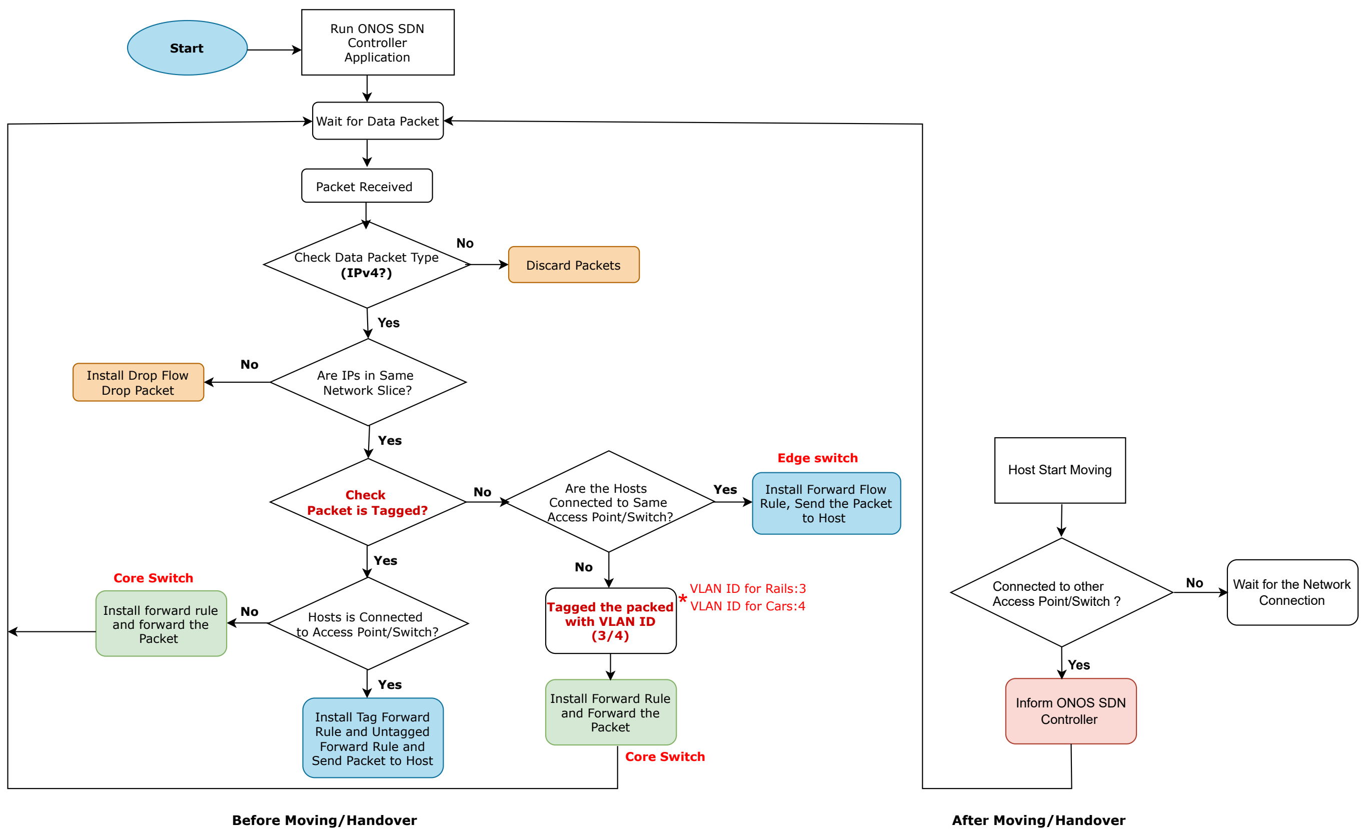

Start
Run ONOS SDN Controller Application
Wait for Data Packet
Packet Received
Check Data Packet Type (IPv4?)
No
Discard Packets
Yes
Are IPs in Same Network Slice?
No
Install Drop Flow Drop Packet
Yes
Check Packet is Tagged?
No
Are the Hosts Connected to Same Access Point/Switch?
Yes
Edge switch
Install Forward Flow Rule, Send the Packet to Host
No
Tagged the packed with VLAN ID (3/4)
*
VLAN ID for Rails:3
VLAN ID for Cars:4
Install Forward Rule and Forward the Packet
Core Switch
Yes
Hosts is Connected to Access Point/Switch?
No
Core Switch
Install forward rule and forward the Packet
Yes
Install Tag Forward Rule and Untagged Forward Rule and Send Packet to Host
Host Start Moving
Connected to other Access Point/Switch ?
No
Wait for the Network Connection
Yes
Inform ONOS SDN Controller
Before Moving/Handover
After Moving/Handover

Figure 9. Flow diagram of SDN application.

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

If both source and destination IP pairs are in the same network slice, the application checks whether the data packet is tagged with a VLAN ID or not. If the data packet is not tagged with VLAN, and source and destination hosts are connected to the same access point/switch, the controller installs the forwarding rule using OpenFlow13 protocol at the current access point/switch, and the data packet is sent to the destination host/node. If the data packet is not tagged with a VLAN ID, and the source and destination host/node are connected to the same access point/switch, the data packet is tagged with a VLAN ID based on the network slice. For this forwarding application, the number 3 is used as the VLAN ID to tag the data packets from/to railways/rails, and the number 4 is used as the VLAN ID to tag the data packets from/to roads/cars. After tagging the data packet, the forwarding tag rule and forwarding untag rule are installed using the OpenFlow13 protocol at the current access point/switch for the IP pairs, and the data packet is forwarded 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 forward tag rule and untag rule are installed 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 forward rule is installed at the current access point/switch and the data packet is forwarded in the network.

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

VLAN Tagging

VLAN tagging is a mechanism that allows the creation of multiple networks at Layer 2 of the core network. VLAN tagging is carried out by assigning a VLAN ID to the data header as an additional element to the Ethernet header of a packet [38]. The assigned tag can be used as a filtering decision factor for the forwarding operation at switches/access points. The VLAN tag defines which side of the network part a data packet belongs to by matching the tag of the data packet header.

Figure 10 shows data packet tagging with the VLAN mechanism used in the developed ONOS application. Switches S1, S2, S7, and S8 are the edge switches, and S3, S4, S5, and S6 are the core switches. Access points ap1, ap2 and ap3 are the access network elements for cars, and ap4, ap5, and ap6 are the access network elements for trains. When a data packet is sent from the car to CarServer, access point ap1 tags the data packet with VLAN ID: 4 (cars’ slice) and forwards the packet to the next switch S3. Since the data packet is already tagged with a VLAN ID, S3 matches the VLAN ID and forwards the data packet to S5; it also matches the VLAN ID and sends the data packet into the network. When this data packet arrives at the edge switch S7 where the destination host CarServer is connected, it removes the VLAN ID and sends the data packet to the CarServer. A similar working principle is followed for the train’s data. The 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 the packet into the network. When this data packet arrives at switch S8, it removes the VLAN ID and sends the data packet to the host rail server.

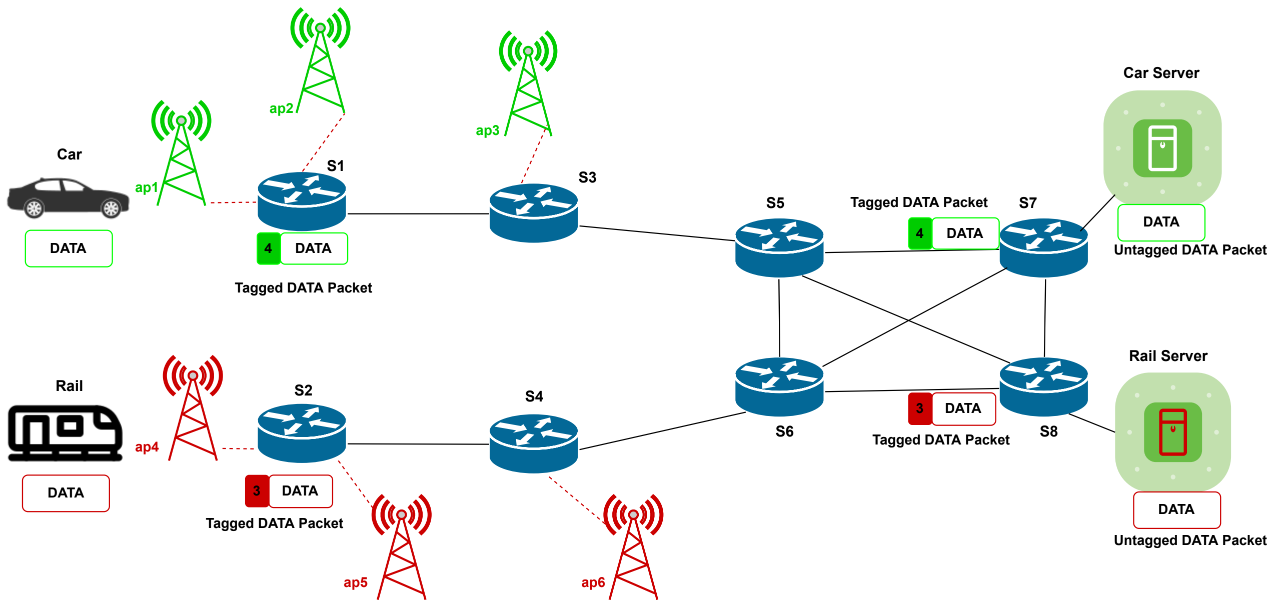


Figure 10. Data packet tagging.

9. Validation of Developed SDN Application and Selected Tools

This section provides the procedure and logic to validate the selected tools that are used to generate the data traffic. Along with this, it also investigates the developed ONOS application and analyses its functionalities such as handover and traffic differentiation capabilities based on VLAN tagging.

9.1. Handover/Mobility (Speed/Direction/Providing Coordinates)

To test the handover scenario, the first criteria is to design a network topology where hosts should have the capability to connect to wireless networks (WiFi) and users can define the mobility and speed of the moving host in a certain direction. These criteria can be fulfilled by Mininet–WiFi since it supports the mobility model and Mininet–WiFi Python API integrates a new method *net.addStation()* that authorizes users to define nodes that can move around in the defined virtual space, as mentioned in Section 6.1. By developing a predefined mobility model, Mininet–WiFi can also move the nodes automatically in the defined virtual space as soon as the emulation scenario starts running. Mininet–WiFi supports the following mobility models: RandomDirection, RandomWalk, RandomWayPoint, GaussMarkov, RefrencePoint, TruncatedLevyWalk, and TimeVariantCommunity [18,22].

To demonstrate the handover and moving for railway and road coexistence environments, scenario S2(5/6)1 is selected, which is described in Section 6.1 of this paper.

In the above-given Listing 6, two nodes, Car1 and Train1, are defined with moving capability. The *speed* parameter can be any feasible number greater than zero. For demonstration purposes, we keep the speed at 3 m/s. In this scenario, to define the moving and handover of nodes, the lines of codes given in List 8 are used. Node Car1 is connected to ap1 and starts moving towards ap2 after 60 s. Similarly, Train1 is initially connected to ap3 and starts moving towards ap4. The mobility of nodes starts using the *net.startMobility()* function and stops by using the *net.stopMobility()* function as shown in Listing 7. The *start* parameter initializes the starting time of the node's mobility, and*stop* initializes the stop time of the node's mobility. While moving, the ping command is executed using CLI from Car1 to CarServer and Train1 to RailServer, as shown in Figure 11.

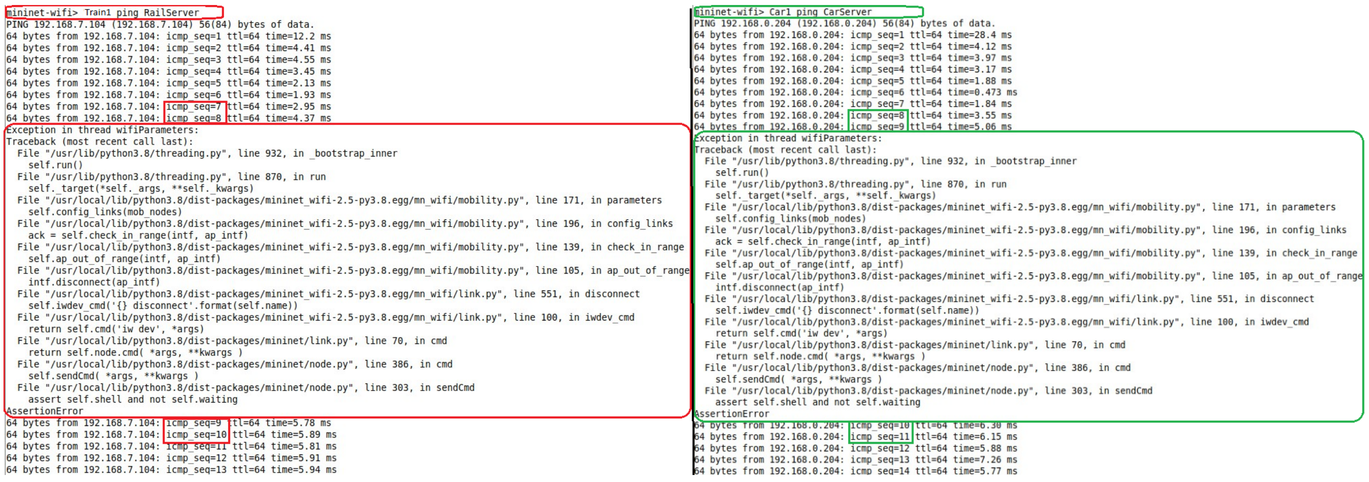
Listing 6. Defining the Host With Their Moving Speeds.

```
 Train1  =net.addStation('Train1',
          mac='00:00:00:00:00:01', ip='192.168.7.101/24', speed=3)
 Car1   =net.addStation('Car1',
          mac='00:00:00:00:00:02', ip='192.168.0.201/24', speed=3)
```

Listing 7. Defining Moving Functionality of Hosts.

```
if '-s' not in args:
        net.startMobility(time=0)
        net.mobility(Train1, 'start', time=60, position='30,10,0')
        net.mobility(Car1, 'start', time=60, position='30,320,0')
        net.mobility(Train1, 'stop', time=70, position='380,60,0')
        net.mobility(Car1, 'stop', time=70, position='350,250,0')
        net.stopMobility(time=75)
```

Both of the selected nodes are pinging their assigned service server so that we can verify the network connectivity and handover between the assigned access points. Figure 11 shows that when Car1 crosses the ap1 coverage range, it automatically connects to ap2. Similarly, Train1 connects to access point ap4 automatically after crossing the coverage range of access point ap3. In Figure 11, we can see that after crossing the previously connected access points, Car1 and Train1 are disconnected, but there is no packet loss, and they are automatically connected to the nearest access point again.

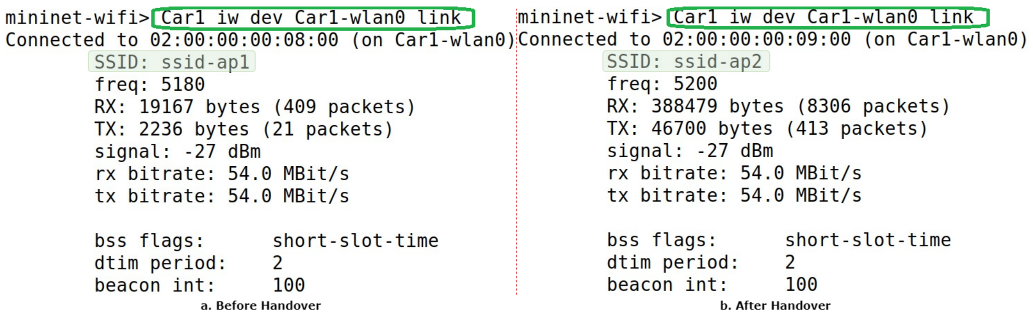


```
mininet-wifi> Train1 ping RailServer
PING 192.168.7.104 (192.168.7.104) 56(84) bytes of data.
64 bytes from 192.168.7.104: icmp_seq=1 ttl=64 time=12.2 ms
64 bytes from 192.168.7.104: icmp_seq=2 ttl=64 time=4.41 ms
64 bytes from 192.168.7.104: icmp_seq=3 ttl=64 time=4.55 ms
64 bytes from 192.168.7.104: icmp_seq=4 ttl=64 time=3.45 ms
64 bytes from 192.168.7.104: icmp_seq=5 ttl=64 time=2.13 ms
64 bytes from 192.168.7.104: icmp_seq=6 ttl=64 time=1.93 ms
64 bytes from 192.168.7.104: icmp_seq=7 ttl=64 time=2.95 ms
64 bytes from 192.168.7.104: icmp_seq=8 ttl=64 time=4.37 ms
Exception in thread wifiParameters:
Traceback (most recent call last):
  File "/usr/lib/python3.8/threading.py", line 932, in _bootstrap_inner
    self.run()
  File "/usr/lib/python3.8/threading.py", line 870, in run
    self._target(*self._args, **self._kwargs)
  File "/usr/local/lib/python3.8/dist-packages/mininet_wifi-2.5-py3.8.egg/mn_wifi/mobility.py", line 171, in parameters
    self.config_links(mob_nodes)
  File "/usr/local/lib/python3.8/dist-packages/mininet_wifi-2.5-py3.8.egg/mn_wifi/mobility.py", line 196, in config_links
    ack = self.check_in_range(intf, ap_intf)
  File "/usr/local/lib/python3.8/dist-packages/mininet_wifi-2.5-py3.8.egg/mn_wifi/mobility.py", line 139, in check_in_range
    self.ap_out_of_range(intf, ap_intf)
  File "/usr/local/lib/python3.8/dist-packages/mininet_wifi-2.5-py3.8.egg/mn_wifi/mobility.py", line 105, in ap_out_of_range
    intf.disconnect(ap_intf)
  File "/usr/local/lib/python3.8/dist-packages/mininet_wifi-2.5-py3.8.egg/mn_wifi/link.py", line 551, in disconnect
    self.iwdev_cmd('{} disconnect'.format(self.name))
  File "/usr/local/lib/python3.8/dist-packages/mininet_wifi-2.5-py3.8.egg/mn_wifi/link.py", line 100, in iwdev_cmd
    return self.cmd('iw dev', *args)
  File "/usr/local/lib/python3.8/dist-packages/mininet/link.py", line 70, in cmd
    return self.node.cmd( *args, **kwargs )
  File "/usr/local/lib/python3.8/dist-packages/mininet/node.py", line 386, in cmd
    self.sendCmd( *args, **kwargs )
  File "/usr/local/lib/python3.8/dist-packages/mininet/node.py", line 303, in sendCmd
    assert self.shell and not self.waiting
AssertionError
64 bytes from 192.168.7.104: icmp_seq=9 ttl=64 time=5.78 ms
64 bytes from 192.168.7.104: icmp_seq=10 ttl=64 time=5.89 ms
64 bytes from 192.168.7.104: icmp_seq=11 ttl=64 time=5.81 ms
64 bytes from 192.168.7.104: icmp_seq=12 ttl=64 time=5.91 ms
64 bytes from 192.168.7.104: icmp_seq=13 ttl=64 time=5.94 ms
```

```
mininet-wifi> Car1 ping CarServer
PING 192.168.0.204 (192.168.0.204) 56(84) bytes of data.
64 bytes from 192.168.0.204: icmp_seq=1 ttl=64 time=28.4 ms
64 bytes from 192.168.0.204: icmp_seq=2 ttl=64 time=4.12 ms
64 bytes from 192.168.0.204: icmp_seq=3 ttl=64 time=3.97 ms
64 bytes from 192.168.0.204: icmp_seq=4 ttl=64 time=3.17 ms
64 bytes from 192.168.0.204: icmp_seq=5 ttl=64 time=1.88 ms
64 bytes from 192.168.0.204: icmp_seq=6 ttl=64 time=0.473 ms
64 bytes from 192.168.0.204: icmp_seq=7 ttl=64 time=1.84 ms
64 bytes from 192.168.0.204: icmp_seq=8 ttl=64 time=3.55 ms
64 bytes from 192.168.0.204: icmp_seq=9 ttl=64 time=5.06 ms
Exception in thread wifiParameters:
Traceback (most recent call last):
  File "/usr/lib/python3.8/threading.py", line 932, in _bootstrap_inner
    self.run()
  File "/usr/lib/python3.8/threading.py", line 870, in run
    self._target(*self._args, **self._kwargs)
  File "/usr/local/lib/python3.8/dist-packages/mininet_wifi-2.5-py3.8.egg/mn_wifi/mobility.py", line 171, in parameters
    self.config_links(mob_nodes)
  File "/usr/local/lib/python3.8/dist-packages/mininet_wifi-2.5-py3.8.egg/mn_wifi/mobility.py", line 196, in config_links
    ack = self.check_in_range(intf, ap_intf)
  File "/usr/local/lib/python3.8/dist-packages/mininet_wifi-2.5-py3.8.egg/mn_wifi/mobility.py", line 139, in check_in_range
    self.ap_out_of_range(intf, ap_intf)
  File "/usr/local/lib/python3.8/dist-packages/mininet_wifi-2.5-py3.8.egg/mn_wifi/mobility.py", line 105, in ap_out_of_range
    intf.disconnect(ap_intf)
  File "/usr/local/lib/python3.8/dist-packages/mininet_wifi-2.5-py3.8.egg/mn_wifi/link.py", line 551, in disconnect
    self.iwdev_cmd('{} disconnect'.format(self.name))
  File "/usr/local/lib/python3.8/dist-packages/mininet_wifi-2.5-py3.8.egg/mn_wifi/link.py", line 100, in iwdev_cmd
    return self.cmd('iw dev', *args)
  File "/usr/local/lib/python3.8/dist-packages/mininet/link.py", line 70, in cmd
    return self.node.cmd( *args, **kwargs )
  File "/usr/local/lib/python3.8/dist-packages/mininet/node.py", line 386, in cmd
    self.sendCmd( *args, **kwargs )
  File "/usr/local/lib/python3.8/dist-packages/mininet/node.py", line 303, in sendCmd
    assert self.shell and not self.waiting
AssertionError
64 bytes from 192.168.0.204: icmp_seq=10 ttl=64 time=6.30 ms
64 bytes from 192.168.0.204: icmp_seq=11 ttl=64 time=6.15 ms
64 bytes from 192.168.0.204: icmp_seq=12 ttl=64 time=5.88 ms
64 bytes from 192.168.0.204: icmp_seq=13 ttl=64 time=7.26 ms
64 bytes from 192.168.0.204: icmp_seq=14 ttl=64 time=5.77 ms
```

Figure 11. Checking connectivity during moving.

To verify the handover and moving functionality of nodes, command *Car1 iw dev Car1-wlan0 link* is executed for Car1, and for Train1 command *Train1 iw dev Train1-wlan0 link* is executed before and after the moving of nodes. Figure 12 shows that before the handover, Car1 is connected to access point ap1, and after handover, it is connected to access point ap2. Figure 13 shows that before moving, Train1 is connected to access point ap3, and after moving, it is connected to access point ap4. Figure 14 shows the topology after the movement of nodes Car1 and Train1, and it also shows that Car1 is now connected to access point ap2 and that Train1 is connected to ap4.



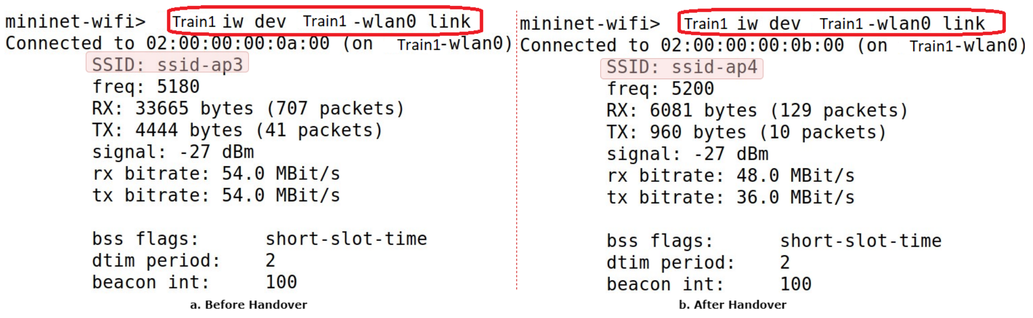
```
mininet-wifi> Car1 iw dev Car1-wlan0 link
Connected to 02:00:00:00:08:00 (on Car1-wlan0)
        SSID: ssid-ap1
        freq: 5180
        RX: 19167 bytes (409 packets)
        TX: 2236 bytes (21 packets)
        signal: -27 dBm
        rx bitrate: 54.0 MBit/s
        tx bitrate: 54.0 MBit/s

        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 12. Connected access point for Car1 before and after handover/moving.



```
mininet-wifi> Train1 iw dev Train1-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> Train1 iw dev Train1-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 13. Connected access point for Train1 before and after handover/moving.

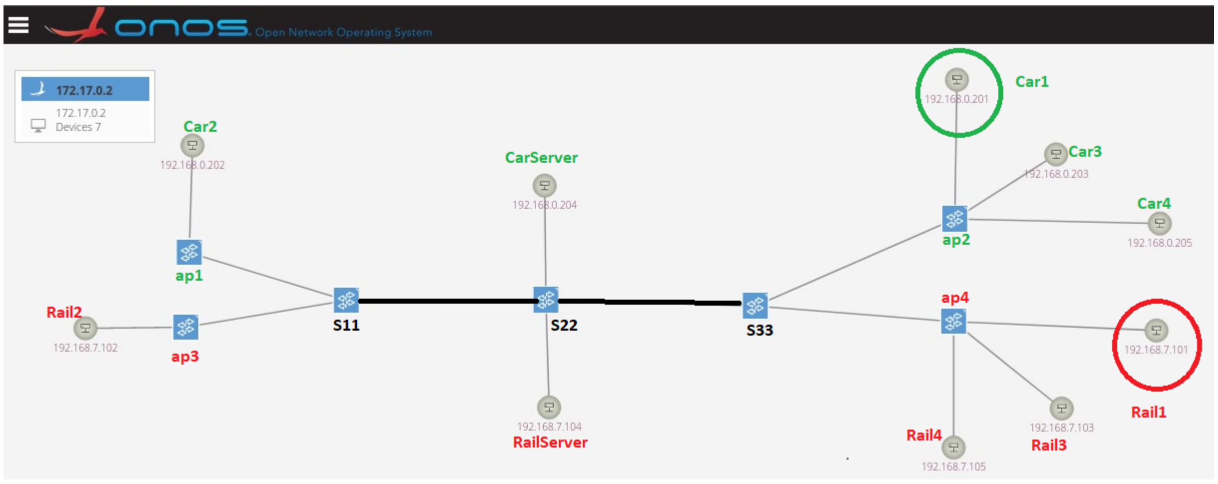


Figure 14. Handover scenario for S2(5/6)1 topology: ONOS screenshot.

Figure 4 shows the position of nodes and access points defined for scenario S2(5/6)1 before the moving scenario, and Figure 15 shows the positions of nodes and access points after moving and handover. After comparing these two graphs, we can conclude that Mininet–WiFi has the potential to emulate moving and handover scenarios for railway and road coexistence environments. There is some delay between the handover. This delay is because of the network joining process carried out by the nodes/stations and during this process; no data loss is recorded.

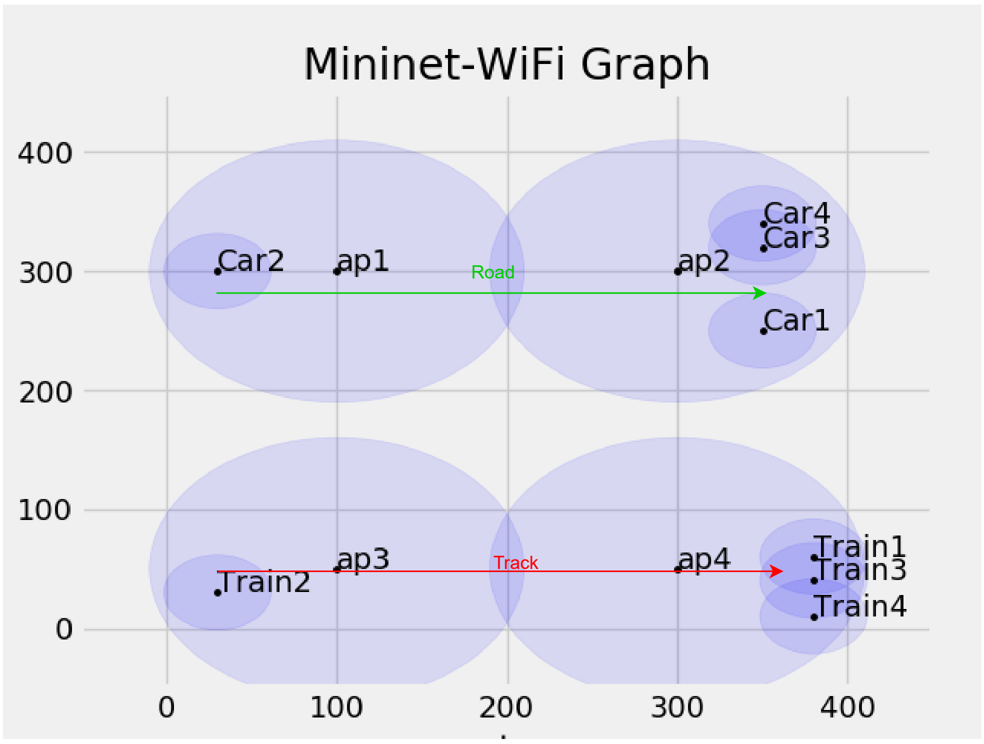


Figure 15. Hosts and access points after handover and moving: Mininet–WiFi graph.

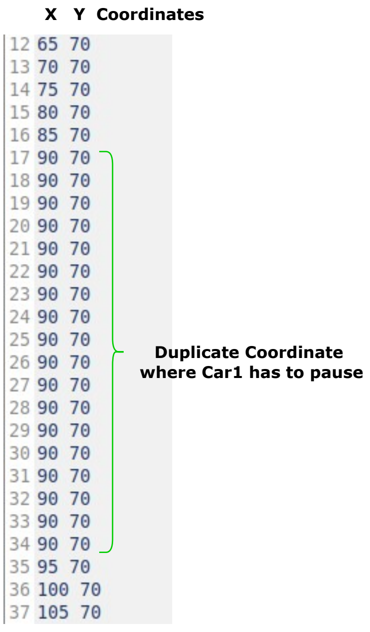
9.2. Level Crossing

To manifest the level crossing scenario for railway and road coexistence environments, the S4(5/6)4 scenario is selected, which is presented in Section 6.2 of this paper. In this scenario, railway tracks are perpendicular to roads, as shown in Figure 7. We know that at a level crossing, when a Train/Tram is coming, Cars or other vehicles should have to stop for some time and move when the train/tram has passed. It is one of the challenges for us to develop this scenario using the Mininet–WiFi and ONOS SDN controller. As we described in Section 9.1, the Mininet–WiFi Python API has a mobility method. Using this method, the user can start and stop the mobility of the nodes, but it does not have a pause and restart moving node functionality. To replicate the level crossing scenario, Car1 should stop its mobility when Train1 is moving at the level crossing point. To achieve this scenario, the *replayingMobility()* function is used, and the location (x- and y-coordinates) of nodes are saved in .dat files; these locations are provided using the function *get_trace()*, as shown in Listing 8.

Listing 8. Defining Moving Functionality of Hosts.

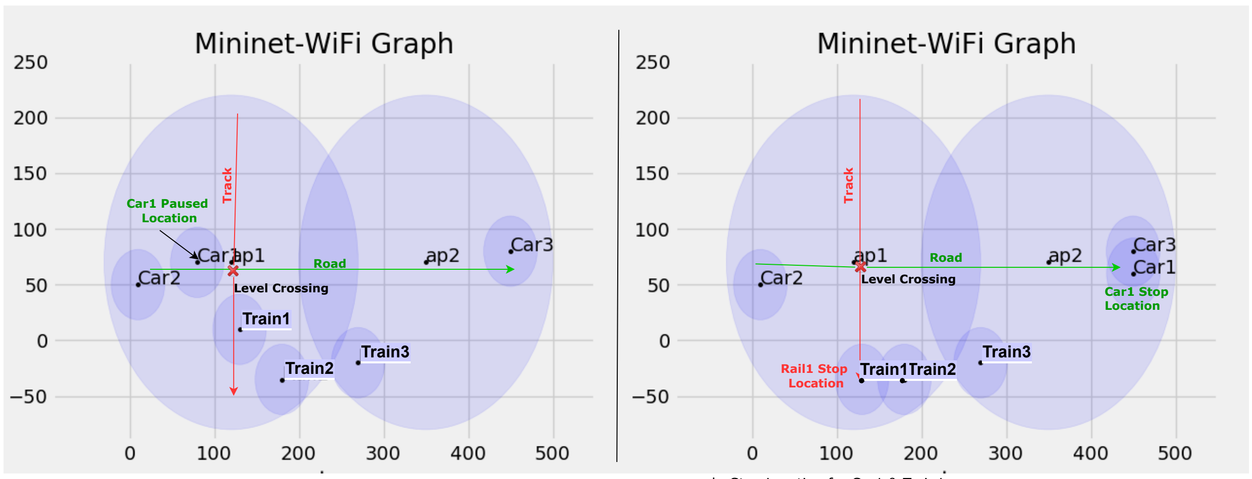
```
net.isReplaying = True
    path = os.path.dirname(os.path.abspath(__file__))
             + '/replayingMobility/'
    get_trace(Car1, '{}node1.dat'.format(path))
    get_trace(Car2, '{}node2.dat'.format(path))
    get_trace(Car3, '{}node3.dat'.format(path))
    get_trace(Train1, '{}node4.dat'.format(path))
    get_trace(Train2, '{}node5.dat'.format(path))
    get_trace(Train3, '{}node6.dat'.format(path))
```

The file node1.dat contains the location coordinates for the route of Car1 in virtual space. Figure 6 shows the initial location of Car1 and Train1. Car1's initial location is x = 10, y = 70, and after moving, the stop location x = 450, y = 70 is assigned. Train1's initial location is x = 130, y = 180, and the stopping location x = 130, y = −35 is assigned. To achieve the pause functionality for Car1, the coordinate x = 90, y = 70, where Car1 has to pause, is duplicated multiple times in the node1.dat file to reserve some time so that Train1 can pass the level crossing, as shown in Figure 16. Therefore, the *get_trace()* function makes Car1 pause at location x = 90, y = 70, and in that pause time, Train1 passes the level crossing; after that, Car1 starts moving again along the assigned route. Figure 17 shows that Car1 is paused at location x = 90, y = 60 and starts moving again when Train1 passed the level crossing.



```
65 70
70 70
75 70
80 70
85 70
90 70
90 70
90 70
90 70
90 70
90 70
90 70
90 70
90 70
90 70
90 70
90 70
90 70
90 70
90 70
90 70
90 70
90 70
95 70
100 70
105 70
```

Figure 16. Car1: node1.dat file screenshot.



a. Car1 Paused Location

b. Stop Location for Car1 & Train1

Figure 17. Level crossing scenario.

9.3. Reachability Test and Data Traffic Differentiation

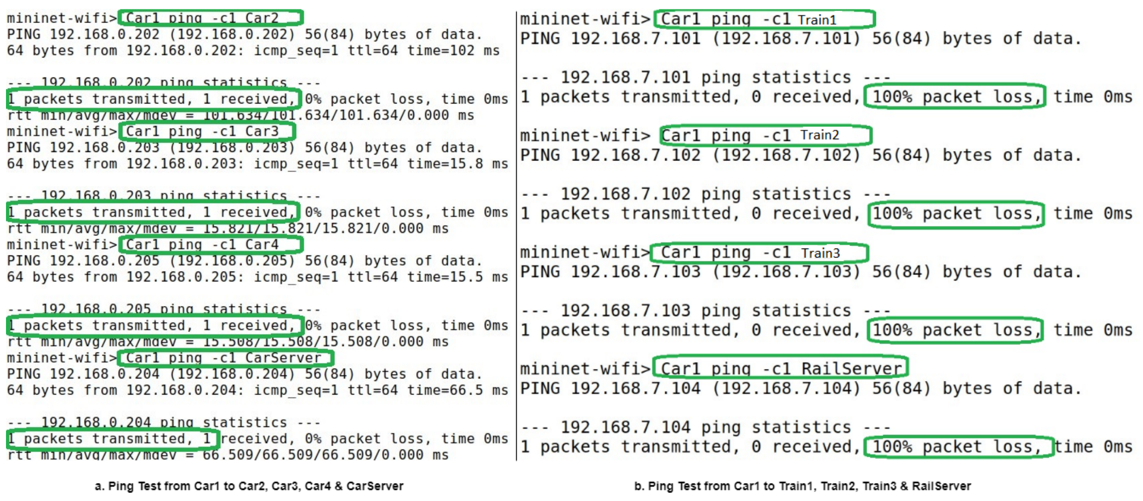
To demonstrate the data traffic differentiation and network reachability test, scenario S2(5/6)1 is selected. In this scenario, the network topology has eight nodes, Car1, Car2, Car3, Car4, Train1, Train2, Train3, and Train4, representing cars and trains, respectively. It also has two hosts, CarServer and RailServer, assigned as service servers for roads and railways, respectively, as shown in the network topology in Figure 5.

The developed SDN application has the ability to differentiate the data traffic based on the assigned VLAN ID, and this test is carried out to validate the data traffic differentiation of the developed application. To carry out this test, the ping command is used to check the connectivity. Nodes Car1, Car2, Car3, Car4, and CarServer should be able to ping, send, and receive a data packet to/from each other but should not be able to communicate with trains and RailServer. In the same manner, Train1, Train2, Train3, Train4, and RailServer should be able to ping, send, and receive a data packet to/from each other but should not be able to communicate with cars and CarServer.

Figure 18 shows the connectivity test for node Car1. We can see that Car1 is able to ping other cars and CarServer, but it is not able to communicate with rails and RailServer. Similarly, Figure 19 shows the connectivity test for node Train1: it is able to communicate with other rails and RailServer but it is not able to communicate with cars and CarServer. This test is carried out for each and every node and host associated with network topology S2(5/6)1, and the outcome of this test is presented in Table 6, which shows that cars are able to communicate only with other cars and with the assigned road service server, i.e., CarServer, and trains are able to communicate only with other trains and with the assigned railway service server, i.e., RailServer.

Table 6. Reachability test.

Src/Dst	Car1	Car2	Car3	Car4	CarServer	Train1	Train2	Train3	Train4	RailServer
Car1	✓	✓	✓	✓	✓	X	X	X	X	X
Car2	✓	✓	✓	✓	✓	X	X	X	X	X
Car3	✓	✓	✓	✓	✓	X	X	X	X	X
Car4	✓	✓	✓	✓	✓	X	X	X	X	X
CarServer	✓	✓	✓	✓	✓	X	X	X	X	X
Train1	X	X	X	X	X	✓	✓	✓	✓	✓
Train2	X	X	X	X	X	✓	✓	✓	✓	✓
Train3	X	X	X	X	X	✓	✓	✓	✓	✓
Train4	X	X	X	X	X	✓	✓	✓	✓	✓
RailServer	X	X	X	X	X	✓	✓	✓	✓	✓



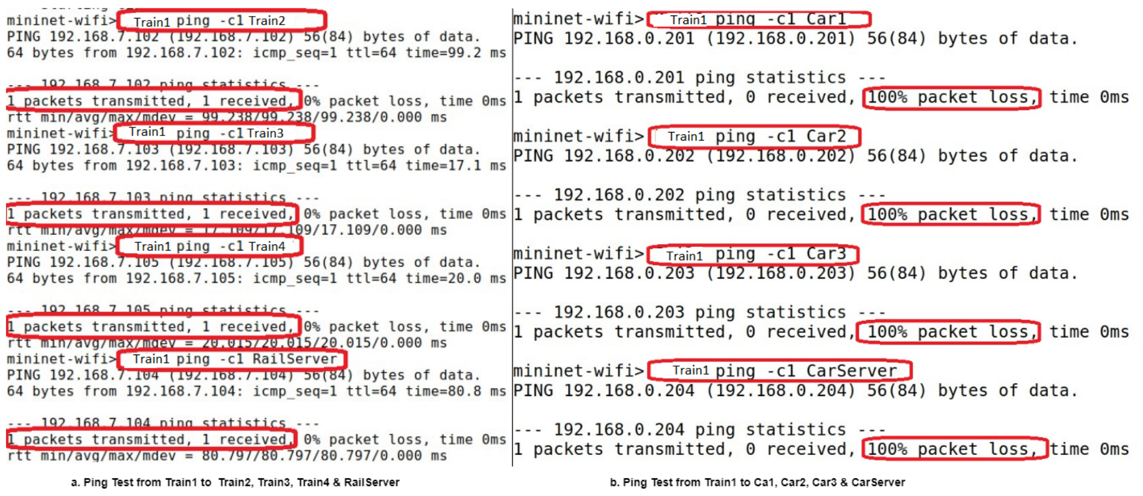
```
mininet-wifi>Car1 ping -c1 Car2
PING 192.168.0.202 (192.168.0.202) 56(84) bytes of data.
64 bytes from 192.168.0.202: icmp_seq=1 ttl=64 time=102 ms

--- 192.168.0.202 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 101.634/101.634/101.634/0.000 ms
mininet-wifi>Car1 ping -c1 Car3
PING 192.168.0.203 (192.168.0.203) 56(84) bytes of data.
64 bytes from 192.168.0.203: icmp_seq=1 ttl=64 time=15.8 ms

--- 192.168.0.203 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 15.821/15.821/15.821/0.000 ms
mininet-wifi>Car1 ping -c1 Car4
PING 192.168.0.205 (192.168.0.205) 56(84) bytes of data.
64 bytes from 192.168.0.205: icmp_seq=1 ttl=64 time=15.5 ms

--- 192.168.0.205 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 15.508/15.508/15.508/0.000 ms
mininet-wifi>Car1 ping -c1 CarServer
PING 192.168.0.204 (192.168.0.204) 56(84) bytes of data.
64 bytes from 192.168.0.204: icmp_seq=1 ttl=64 time=66.5 ms

--- 192.168.0.204 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 66.509/66.509/66.509/0.000 ms
```

a. Ping Test from Car1 to Car2, Car3, Car4 & CarServer

```
mininet-wifi>Car1 ping -c1 Train1
PING 192.168.7.101 (192.168.7.101) 56(84) bytes of data.

--- 192.168.7.101 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms

mininet-wifi> Car1 ping -c1 Train2
PING 192.168.7.102 (192.168.7.102) 56(84) bytes of data.

--- 192.168.7.102 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms

mininet-wifi>Car1 ping -c1 Train3
PING 192.168.7.103 (192.168.7.103) 56(84) bytes of data.

--- 192.168.7.103 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms

mininet-wifi>Car1 ping -c1 RailServer
PING 192.168.7.104 (192.168.7.104) 56(84) bytes of data.

--- 192.168.7.104 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms
```

b. Ping Test from Car1 to Train1, Train2, Train3 & RailServer

Figure 18. Reachability test for cars.



```
mininet-wifi> Train1 ping -c1 Train2
PING 192.168.7.102 (192.168.7.102) 56(84) bytes of data.
64 bytes from 192.168.7.102: icmp_seq=1 ttl=64 time=99.2 ms

--- 192.168.7.102 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 99.238/99.238/99.238/0.000 ms
mininet-wifi>Train1 ping -c1 Train3
PING 192.168.7.103 (192.168.7.103) 56(84) bytes of data.
64 bytes from 192.168.7.103: icmp_seq=1 ttl=64 time=17.1 ms

--- 192.168.7.103 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 17.109/17.109/17.109/0.000 ms
mininet-wifi> Train1 ping -c1 Train4
PING 192.168.7.105 (192.168.7.105) 56(84) bytes of data.
64 bytes from 192.168.7.105: icmp_seq=1 ttl=64 time=20.0 ms

--- 192.168.7.105 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 20.015/20.015/20.015/0.000 ms
mininet-wifi> Train1 ping -c1 RailServer
PING 192.168.7.104 (192.168.7.104) 56(84) bytes of data.
64 bytes from 192.168.7.104: icmp_seq=1 ttl=64 time=80.8 ms

--- 192.168.7.104 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 80.797/80.797/80.797/0.000 ms
```

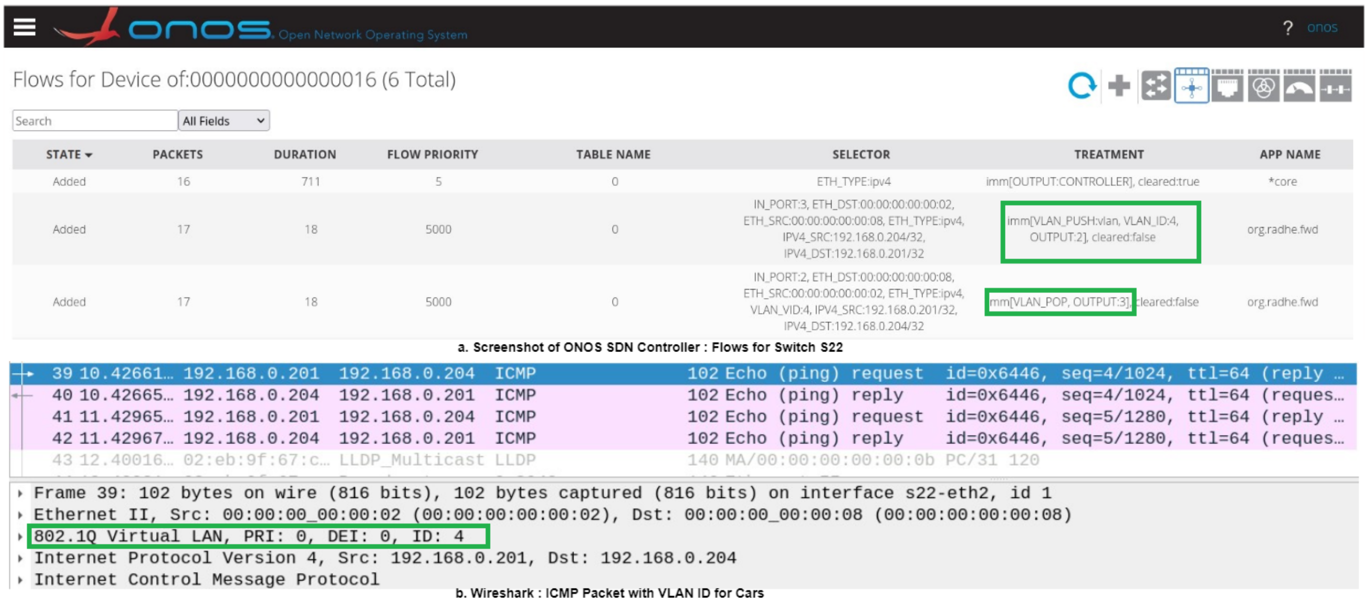
a. Ping Test from Train1 to Train2, Train3, Train4 & RailServer

```
mininet-wifi> Train1 ping -c1 Car1
PING 192.168.0.201 (192.168.0.201) 56(84) bytes of data.

--- 192.168.0.201 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms

mininet-wifi> Train1 ping -c1 Car2
PING 192.168.0.202 (192.168.0.202) 56(84) bytes of data.

--- 192.168.0.202 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms

mininet-wifi> Train1 ping -c1 Car3
PING 192.168.0.203 (192.168.0.203) 56(84) bytes of data.

--- 192.168.0.203 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms

mininet-wifi> Train1 ping -c1 CarServer
PING 192.168.0.204 (192.168.0.204) 56(84) bytes of data.

--- 192.168.0.204 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms
```

b. Ping Test from Train1 to Ca1, Car2, Car3 & CarServer

Figure 19. Reachability test for trains.

Figures 20 and 21 show the entries introduced by the developed SDN application on the switch S22. It shows that the developed SDN application is capable of differentiating the data packets from cars and trains by assigning a VLAN ID 4 to data packets to/from cars and CarServer and VLAN ID 3 to data packets to/from trains and RailServer. Figures 22 and 23 are the screenshots taken from the SDN application's log and show that the application restricts the communication between road entities and railway entities by maintaining connectivity and data traffic differentiation.



Flows for Device of:0000000000000016 (6 Total)

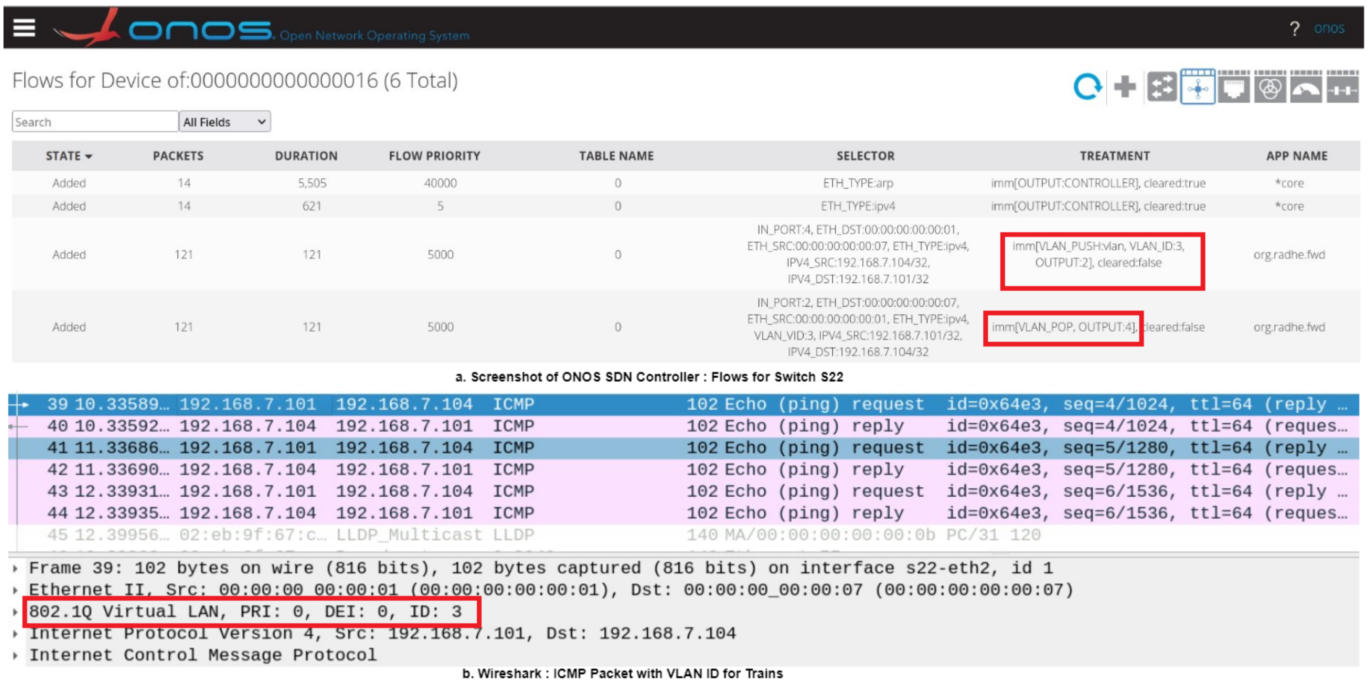
STATE	PACKETS	DURATION	FLOW PRIORITY	TABLE NAME	SELECTOR	TREATMENT	APP NAME
Added	16	711	5	0	ETH_TYPE:ipv4	imm[OUTPUT:CONTROLLER], cleared:true	*core
Added	17	18	5000	0	IN_PORT:3, ETH_DST:00:00:00:00:00:02, ETH_SRC:00:00:00:00:00:08, ETH_TYPE:ipv4, IPV4_SRC:192.168.0.204/32, IPV4_DST:192.168.0.201/32	imm[VLAN_PUSH:vlan, VLAN_ID:4, OUTPUT:2], cleared:false	org.radhe.fwd
Added	17	18	5000	0	IN_PORT:2, ETH_DST:00:00:00:00:00:08, ETH_SRC:00:00:00:00:00:02, ETH_TYPE:ipv4, VLAN_VID:4, IPV4_SRC:192.168.0.201/32, IPV4_DST:192.168.0.204/32	imm[VLAN_POP, OUTPUT:3], cleared:false	org.radhe.fwd

a. Screenshot of ONOS SDN Controller : Flows for Switch S22

```
39 10.42661… 192.168.0.201  192.168.0.204  ICMP        102 Echo (ping) request  id=0x6446, seq=4/1024, ttl=64 (reply …
40 10.42665… 192.168.0.204  192.168.0.201  ICMP        102 Echo (ping) reply    id=0x6446, seq=4/1024, ttl=64 (reques…
41 11.42965… 192.168.0.201  192.168.0.204  ICMP        102 Echo (ping) request  id=0x6446, seq=5/1280, ttl=64 (reply …
42 11.42967… 192.168.0.204  192.168.0.201  ICMP        102 Echo (ping) reply    id=0x6446, seq=5/1280, ttl=64 (reques…
43 12.40016… 02:eb:9f:67:c… LLDP_Multicast LLDP        140 MA/00:00:00:00:00:0b PC/31 120

Frame 39: 102 bytes on wire (816 bits), 102 bytes captured (816 bits) on interface s22-eth2, id 1
Ethernet II, Src: 00:00:00_00:00:02 (00:00:00:00:00:02), Dst: 00:00:00_00:00:08 (00:00:00:00:00:08)
802.1Q Virtual LAN, PRI: 0, DEI: 0, ID: 4
Internet Protocol Version 4, Src: 192.168.0.201, Dst: 192.168.0.204
Internet Control Message Protocol
```

b. Wireshark : ICMP Packet with VLAN ID for Cars

Figure 20. Flow entries for switch S22: VLAN ID assignment to car's data packet.



Flows for Device of:0000000000000016 (6 Total)

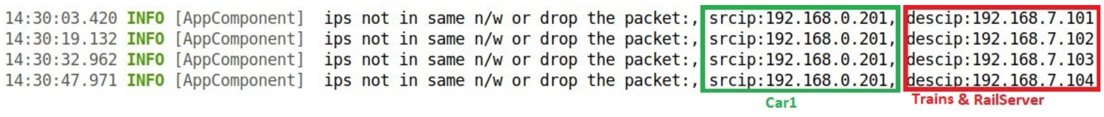
STATE	PACKETS	DURATION	FLOW PRIORITY	TABLE NAME	SELECTOR	TREATMENT	APP NAME
Added	14	5,505	40000	0	ETH_TYPE:arp	imm[OUTPUT:CONTROLLER], cleared:true	*core
Added	14	621	5	0	ETH_TYPE:ipv4	imm[OUTPUT:CONTROLLER], cleared:true	*core
Added	121	121	5000	0	IN_PORT:4, ETH_DST:00:00:00:00:00:01, ETH_SRC:00:00:00:00:00:07, ETH_TYPE:ipv4, IPV4_SRC:192.168.7.104/32, IPV4_DST:192.168.7.101/32	imm[VLAN_PUSH:vlan, VLAN_ID:3, OUTPUT:2], cleared:false	org.radhe.fwd
Added	121	121	5000	0	IN_PORT:2, ETH_DST:00:00:00:00:00:07, ETH_SRC:00:00:00:00:00:01, ETH_TYPE:ipv4, VLAN_VID:3, IPV4_SRC:192.168.7.101/32, IPV4_DST:192.168.7.104/32	imm[VLAN_POP, OUTPUT:4], cleared:false	org.radhe.fwd

a. Screenshot of ONOS SDN Controller : Flows for Switch S22

```
39 10.33589… 192.168.7.101  192.168.7.104  ICMP        102 Echo (ping) request  id=0x64e3, seq=4/1024, ttl=64 (reply …
40 10.33592… 192.168.7.104  192.168.7.101  ICMP        102 Echo (ping) reply    id=0x64e3, seq=4/1024, ttl=64 (reques…
41 11.33686… 192.168.7.101  192.168.7.104  ICMP        102 Echo (ping) request  id=0x64e3, seq=5/1280, ttl=64 (reply …
42 11.33690… 192.168.7.104  192.168.7.101  ICMP        102 Echo (ping) reply    id=0x64e3, seq=5/1280, ttl=64 (reques…
43 12.33931… 192.168.7.101  192.168.7.104  ICMP        102 Echo (ping) request  id=0x64e3, seq=6/1536, ttl=64 (reply …
44 12.33935… 192.168.7.104  192.168.7.101  ICMP        102 Echo (ping) reply    id=0x64e3, seq=6/1536, ttl=64 (reques…
45 12.39956… 02:eb:9f:67:c… LLDP_Multicast LLDP        140 MA/00:00:00:00:00:0b PC/31 120

Frame 39: 102 bytes on wire (816 bits), 102 bytes captured (816 bits) on interface s22-eth2, id 1
Ethernet II, Src: 00:00:00_00:00:01 (00:00:00:00:00:01), Dst: 00:00:00_00:00:07 (00:00:00:00:00:07)
802.1Q Virtual LAN, PRI: 0, DEI: 0, ID: 3
Internet Protocol Version 4, Src: 192.168.7.101, Dst: 192.168.7.104
Internet Control Message Protocol
```

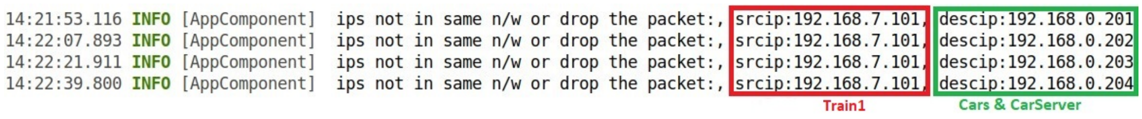
b. Wireshark : ICMP Packet with VLAN ID for Trains

Figure 21. Flow entries for switch S22: VLAN ID assignment to train's data packet.



```
14:30:03.420 INFO [AppComponent]  ips not in same n/w or drop the packet:, srcip:192.168.0.201, descip:192.168.7.101
14:30:19.132 INFO [AppComponent]  ips not in same n/w or drop the packet:, srcip:192.168.0.201, descip:192.168.7.102
14:30:32.962 INFO [AppComponent]  ips not in same n/w or drop the packet:, srcip:192.168.0.201, descip:192.168.7.103
14:30:47.971 INFO [AppComponent]  ips not in same n/w or drop the packet:, srcip:192.168.0.201, descip:192.168.7.104
```

Figure 22. Screenshot of SDN application log: ping from Car1 to Train1, Train2, Train3, and RailServer.

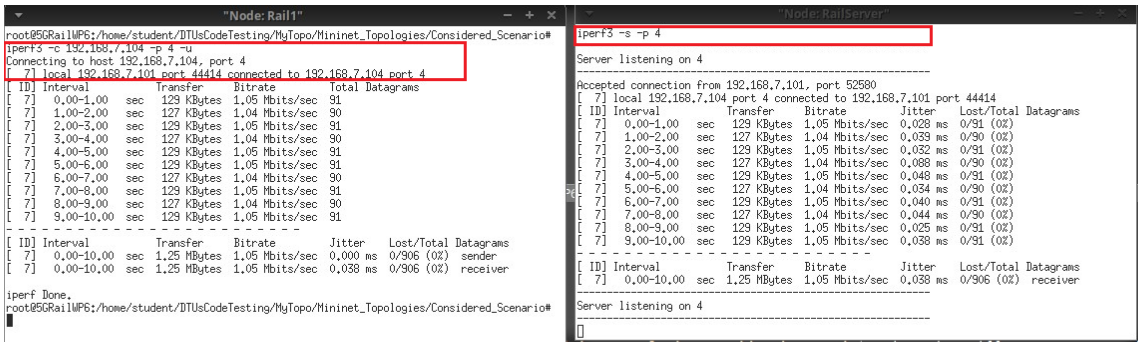


```
14:21:53.116 INFO [AppComponent]  ips not in same n/w or drop the packet:, srcip:192.168.7.101, descip:192.168.0.201
14:22:07.893 INFO [AppComponent]  ips not in same n/w or drop the packet:, srcip:192.168.7.101, descip:192.168.0.202
14:22:21.911 INFO [AppComponent]  ips not in same n/w or drop the packet:, srcip:192.168.7.101, descip:192.168.0.203
14:22:39.800 INFO [AppComponent]  ips not in same n/w or drop the packet:, srcip:192.168.7.101, descip:192.168.0.204
```

Figure 23. Screenshot of SDN application log: ping from Train1 to Car1,Car2, Car3, and CarServer.

9.4. TCP and UDP Data Transmission

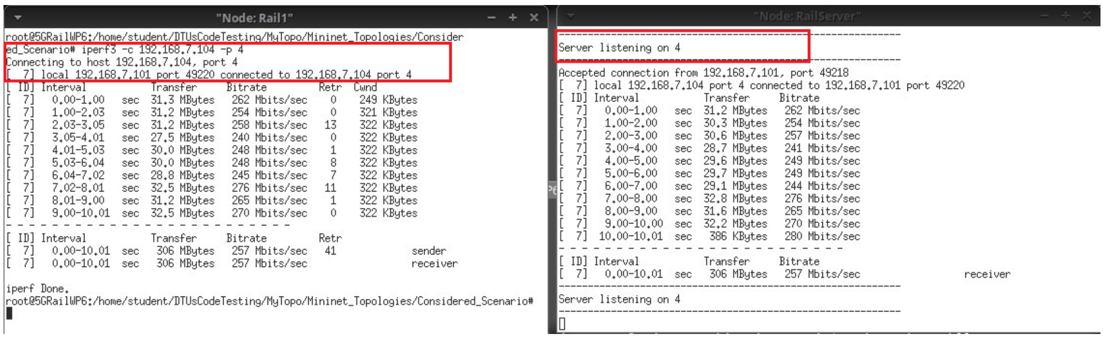
The objective of this test is to demonstrate the standard data communication between cars to CarServer and trains to RailServer. Figure 24 shows UDP data packet transmission and Figure 25 shows TCP data packet transmission from Train1 to RailServer. In this case, Train1 is acting as the client, and RailServer is configured as a listening server.



```
root@5GRailWP6:/home/student/DTUsCodeTesting/MyTopo/Mininet_Topologies/Considered_Scenario#
iperf3 -c 192.168.7.104 -p 4 -u
Connecting to host 192.168.7.104, port 4
[  7] local 192.168.7.101 port 44414 connected to 192.168.7.104 port 4
[ ID] Interval           Transfer     Bitrate         Total Datagrams
[  7]   0.00-1.00   sec   129 KBytes  1.05 Mbits/sec  91
[  7]   1.00-2.00   sec   127 KBytes  1.04 Mbits/sec  90
[  7]   2.00-3.00   sec   129 KBytes  1.05 Mbits/sec  91
[  7]   3.00-4.00   sec   127 KBytes  1.04 Mbits/sec  90
[  7]   4.00-5.00   sec   129 KBytes  1.05 Mbits/sec  91
[  7]   5.00-6.00   sec   129 KBytes  1.05 Mbits/sec  91
[  7]   6.00-7.00   sec   127 KBytes  1.04 Mbits/sec  90
[  7]   7.00-8.00   sec   129 KBytes  1.05 Mbits/sec  91
[  7]   8.00-9.00   sec   127 KBytes  1.04 Mbits/sec  90
[  7]   9.00-10.00  sec   129 KBytes  1.05 Mbits/sec  91
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Jitter    Lost/Total Datagrams
[  7]   0.00-10.00  sec  1.25 MBytes  1.05 Mbits/sec  0.000 ms  0/906 (0%)  sender
[  7]   0.00-10.00  sec  1.25 MBytes  1.05 Mbits/sec  0.038 ms  0/906 (0%)  receiver

iperf Done.
root@5GRailWP6:/home/student/DTUsCodeTesting/MyTopo/Mininet_Topologies/Considered_Scenario#
```

```
iperf3 -s -p 4

Server listening on 4
-----------------------------------------------------------
Accepted connection from 192.168.7.101, port 52580
[  7] local 192.168.7.104 port 4 connected to 192.168.7.101 port 44414
[ ID] Interval           Transfer     Bitrate         Jitter    Lost/Total Datagrams
[  7]   0.00-1.00   sec   129 KBytes  1.05 Mbits/sec  0.028 ms  0/91 (0%)
[  7]   1.00-2.00   sec   127 KBytes  1.04 Mbits/sec  0.039 ms  0/90 (0%)
[  7]   2.00-3.00   sec   129 KBytes  1.05 Mbits/sec  0.032 ms  0/91 (0%)
[  7]   3.00-4.00   sec   127 KBytes  1.04 Mbits/sec  0.088 ms  0/90 (0%)
[  7]   4.00-5.00   sec   129 KBytes  1.05 Mbits/sec  0.048 ms  0/91 (0%)
[  7]   5.00-6.00   sec   127 KBytes  1.04 Mbits/sec  0.034 ms  0/90 (0%)
[  7]   6.00-7.00   sec   129 KBytes  1.05 Mbits/sec  0.040 ms  0/91 (0%)
[  7]   7.00-8.00   sec   127 KBytes  1.04 Mbits/sec  0.044 ms  0/90 (0%)
[  7]   8.00-9.00   sec   129 KBytes  1.05 Mbits/sec  0.025 ms  0/91 (0%)
[  7]   9.00-10.00  sec   129 KBytes  1.05 Mbits/sec  0.038 ms  0/91 (0%)
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Jitter    Lost/Total Datagrams
[  7]   0.00-10.00  sec  1.25 MBytes  1.05 Mbits/sec  0.038 ms  0/906 (0%)  receiver
-----------------------------------------------------------
Server listening on 4
-----------------------------------------------------------
```

Figure 24. UDP data transmission from Train1 to RailServer.



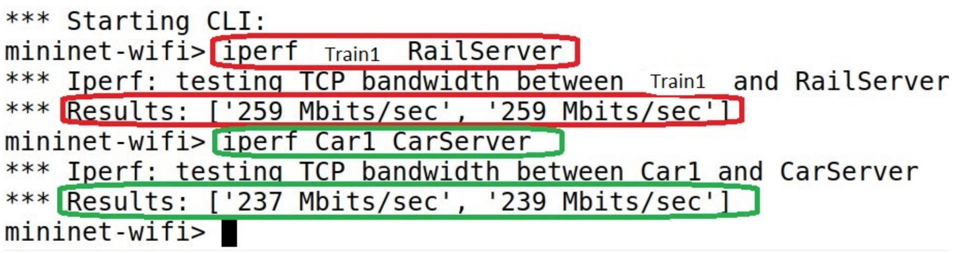
```
root@5GRailWP6:/home/student/DTUsCodeTesting/MyTopo/Mininet_Topologies/Consider
ed_Scenario# iperf3 -c 192.168.7.104 -p 4
Connecting to host 192.168.7.104, port 4
[  7] local 192.168.7.101 port 49220 connected to 192.168.7.104 port 4
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[  7]   0.00-1.00   sec  31.3 MBytes   262 Mbits/sec    0    249 KBytes
[  7]   1.00-2.03   sec  31.2 MBytes   254 Mbits/sec    0    321 KBytes
[  7]   2.03-3.05   sec  31.2 MBytes   258 Mbits/sec   13    322 KBytes
[  7]   3.05-4.01   sec  27.5 MBytes   240 Mbits/sec    0    322 KBytes
[  7]   4.01-5.03   sec  30.0 MBytes   248 Mbits/sec    1    322 KBytes
[  7]   5.03-6.04   sec  30.0 MBytes   248 Mbits/sec    8    322 KBytes
[  7]   6.04-7.02   sec  28.8 MBytes   245 Mbits/sec    7    322 KBytes
[  7]   7.02-8.01   sec  32.5 MBytes   276 Mbits/sec   11    322 KBytes
[  7]   8.01-9.00   sec  31.2 MBytes   265 Mbits/sec    1    322 KBytes
[  7]   9.00-10.01  sec  32.5 MBytes   270 Mbits/sec    0    322 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[  7]   0.00-10.01  sec   306 MBytes   257 Mbits/sec   41             sender
[  7]   0.00-10.01  sec   306 MBytes   257 Mbits/sec                  receiver

iperf Done.
root@5GRailWP6:/home/student/DTUsCodeTesting/MyTopo/Mininet_Topologies/Considered_Scenario#
```

```
-----------------------------------------------------------
Server listening on 4
-----------------------------------------------------------
Accepted connection from 192.168.7.101, port 49218
[  7] local 192.168.7.104 port 4 connected to 192.168.7.101 port 49220
[ ID] Interval           Transfer     Bitrate
[  7]   0.00-1.00   sec  31.2 MBytes   262 Mbits/sec
[  7]   1.00-2.00   sec  30.3 MBytes   254 Mbits/sec
[  7]   2.00-3.00   sec  30.6 MBytes   257 Mbits/sec
[  7]   3.00-4.00   sec  28.7 MBytes   241 Mbits/sec
[  7]   4.00-5.00   sec  29.6 MBytes   249 Mbits/sec
[  7]   5.00-6.00   sec  29.7 MBytes   249 Mbits/sec
[  7]   6.00-7.00   sec  29.1 MBytes   244 Mbits/sec
[  7]   7.00-8.00   sec  32.8 MBytes   276 Mbits/sec
[  7]   8.00-9.00   sec  31.6 MBytes   265 Mbits/sec
[  7]   9.00-10.00  sec  32.2 MBytes   270 Mbits/sec
[  7]  10.00-10.01  sec   386 KBytes   280 Mbits/sec
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate
[  7]   0.00-10.01  sec   306 MBytes   257 Mbits/sec                  receiver
-----------------------------------------------------------
Server listening on 4
-----------------------------------------------------------
```

Figure 25. TCP data transmission from Train1 to RailServer.

9.5. 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 26 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.



```
*** Starting CLI:
mininet-wifi> iperf  Train1  RailServer
*** Iperf: testing TCP bandwidth between  Train1  and RailServer
*** Results: ['259 Mbits/sec', '259 Mbits/sec']
mininet-wifi> iperf Car1 CarServer
*** Iperf: testing TCP bandwidth between Car1 and CarServer
*** Results: ['237 Mbits/sec', '239 Mbits/sec']
mininet-wifi>
```

Figure 26. Link capacity test.

9.6. 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 Train1 to RailServer. This is carried out by opening the xterm window for nodes Car1 and Train1 using the command **xterm Car1 Train1** from the Mininet–WiFi terminal. To measure the latency from Car1, the command *mtr -r -n -c 100 192.168.0.204 -u -P 3* is used, where 192.168.0.204 is the IP address of CarServer and is connected to port 3 of network switch

S22. Similarly, the command *mtr -r -n -c 100 192.168.7.104 -u -P 4* is executed from Train1 to measure the latency while sending 100 UDP data packets to RailServer. The IP address of the assigned RailServer is 192.168.7.104, and it is connected to port 4 of network switch S22. The latency for this network topology is in the range of 4.7 to 5.7 milliseconds. Figures 27 and 28 show the latency test for the selected network topology.

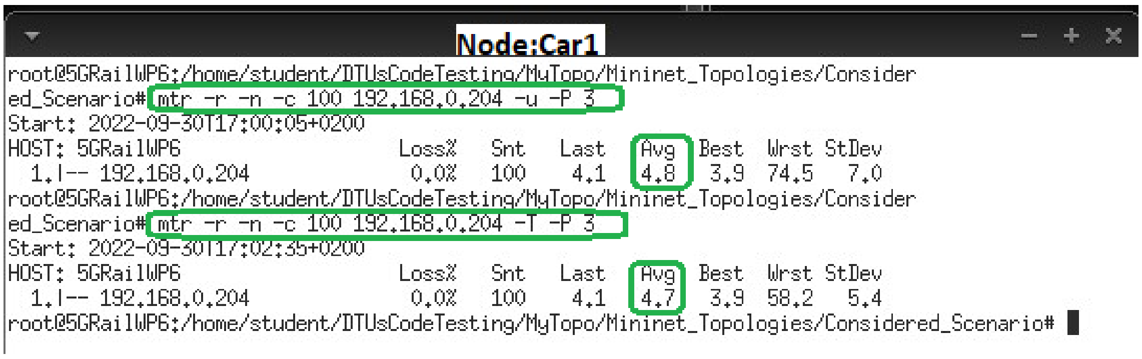

```
Node:Car1
root@5GRailWP6:/home/student/DTUsCodeTesting/MyTopo/Mininet_Topologies/Consider
ed_Scenario# mtr -r -n -c 100 192.168.0.204 -u -P 3
Start: 2022-09-30T17:00:05+0200
HOST: 5GRailWP6                    Loss%   Snt   Last   Avg  Best  Wrst StDev
  1.|-- 192.168.0.204               0.0%   100    4.1   4.8   3.9  74.5   7.0
root@5GRailWP6:/home/student/DTUsCodeTesting/MyTopo/Mininet_Topologies/Consider
ed_Scenario# mtr -r -n -c 100 192.168.0.204 -T -P 3
Start: 2022-09-30T17:02:35+0200
HOST: 5GRailWP6                    Loss%   Snt   Last   Avg  Best  Wrst StDev
  1.|-- 192.168.0.204               0.0%   100    4.1   4.7   3.9  58.2   5.4
root@5GRailWP6:/home/student/DTUsCodeTesting/MyTopo/Mininet_Topologies/Considered_Scenario#
```

Figure 27. Latency test from Car1.

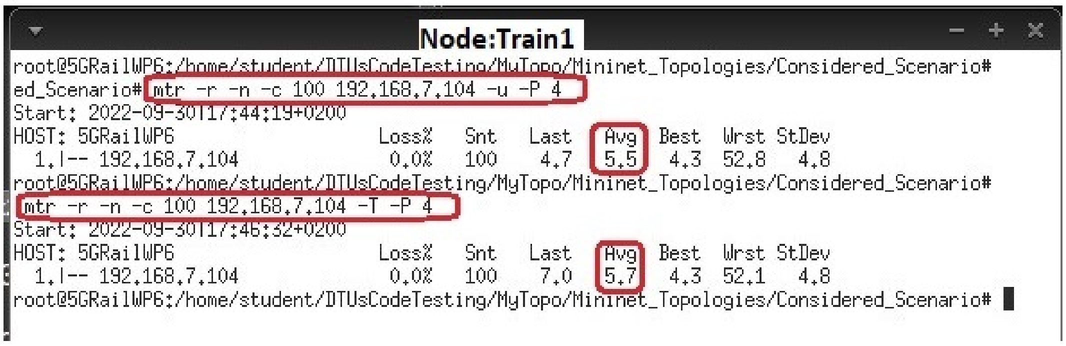
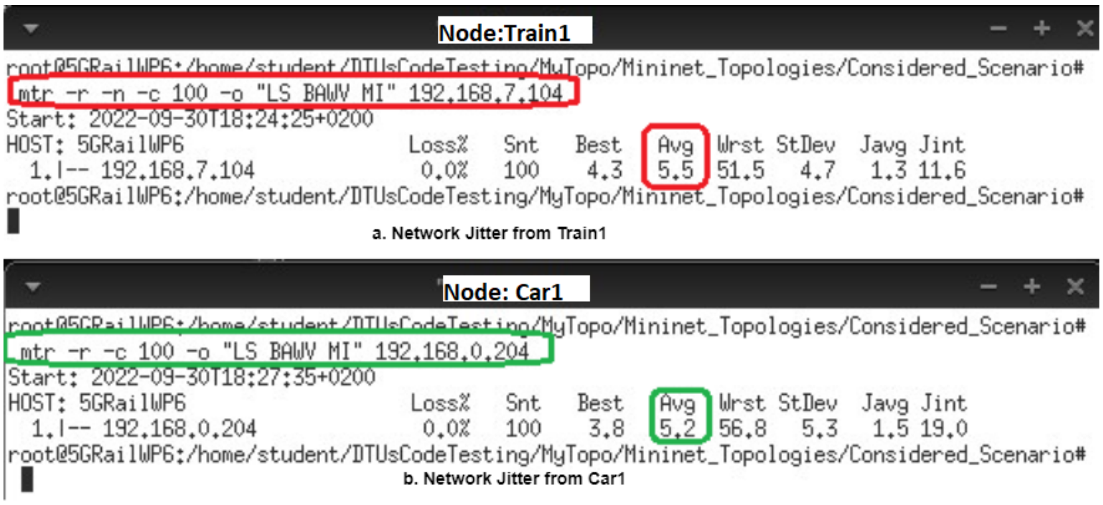

```
Node:Train1
root@5GRailWP6:/home/student/DTUsCodeTesting/MyTopo/Mininet_Topologies/Considered_Scenario#
ed_Scenario# mtr -r -n -c 100 192.168.7.104 -u -P 4
Start: 2022-09-30T17:44:19+0200
HOST: 5GRailWP6                    Loss%   Snt   Last   Avg  Best  Wrst StDev
  1.|-- 192.168.7.104               0.0%   100    4.7   5.5   4.3  52.8   4.8
root@5GRailWP6:/home/student/DTUsCodeTesting/MyTopo/Mininet_Topologies/Considered_Scenario#
 mtr -r -n -c 100 192.168.7.104 -T -P 4
Start: 2022-09-30T17:46:32+0200
HOST: 5GRailWP6                    Loss%   Snt   Last   Avg  Best  Wrst StDev
  1.|-- 192.168.7.104               0.0%   100    7.0   5.7   4.3  52.1   4.8
root@5GRailWP6:/home/student/DTUsCodeTesting/MyTopo/Mininet_Topologies/Considered_Scenario#
```

Figure 28. Latency test from Train1.

To measure the jitter of the network, the command *mtr -r -n -c 100 -o "LS BAWV MI" 192.168.7.104* is executed from the terminal of node Train1, and the command *mtr -r -n -c 100 -o "LS BAWV MI" 192.168.0.204* is executed from the terminal of node Car1. The information about commands and parameters used is given in Table 5. Figure 29 shows that the jitter of the network is in the range of 5.2 to 5.5 milliseconds.


```
Node:Train1
root@5GRailWP6:/home/student/DTUsCodeTesting/MyTopo/Mininet_Topologies/Considered_Scenario#
 mtr -r -n -c 100 -o "LS BAWV MI" 192.168.7.104
Start: 2022-09-30T18:24:25+0200
HOST: 5GRailWP6                    Loss%   Snt   Best   Avg  Wrst StDev  Javg Jint
  1.|-- 192.168.7.104               0.0%   100    4.3   5.5  51.5   4.7   1.3 11.6
root@5GRailWP6:/home/student/DTUsCodeTesting/MyTopo/Mininet_Topologies/Considered_Scenario#
```

a. Network Jitter from Train1

```
Node: Car1
root@5GRailWP6:/home/student/DTUsCodeTesting/MyTopo/Mininet_Topologies/Considered_Scenario#
 mtr -r -c 100 -o "LS BAWV MI" 192.168.0.204
Start: 2022-09-30T18:27:35+0200
HOST: 5GRailWP6                    Loss%   Snt   Best   Avg  Wrst StDev  Javg Jint
  1.|-- 192.168.0.204               0.0%   100    3.8   5.2  56.8   5.3   1.5 19.0
root@5GRailWP6:/home/student/DTUsCodeTesting/MyTopo/Mininet_Topologies/Considered_Scenario#
```

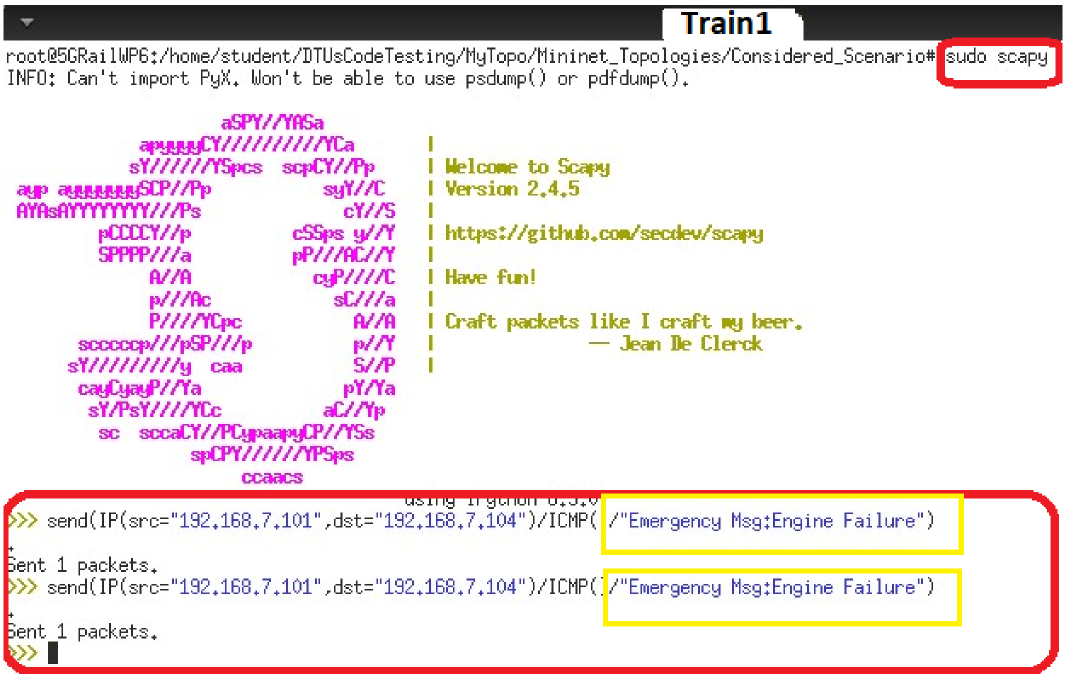
b. Network Jitter from Car1

Figure 29. Network jitter test.

9.7. Sending a Critical Message to the Assigned Server

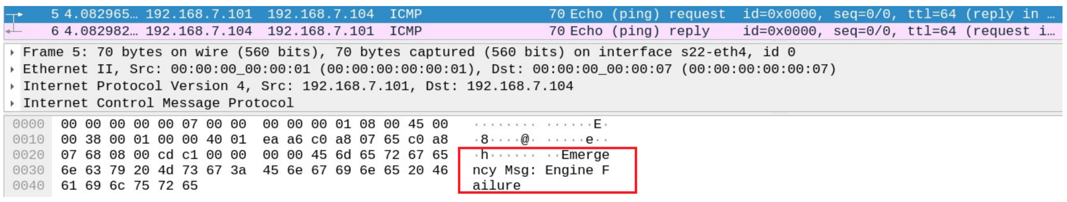
To send a critical message or information, the Scapy tool is used for the considered scenarios. Using this tool, a user-defined message is sent from any node/station to the assigned service server.

Figure 30 shows that an ICMP data packet is sent with a message *"Emergency Msg: Engine Failure"* from node Train1 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 31 shows the data packet captured using the Wireshark tool.



```
root@5GRailWP6:/home/student/DTUsCodeTesting/MyTopo/Mininet_Topologies/Considered_Scenario# sudo scapy
INFO: Can't import PyX. Won't be able to use psdump() or pdfdump().

                     aSPY//YASa
             apyyyyCY//////////YCa       |
            sY//////YSpcs  scpCY//Pp     | Welcome to Scapy
 ayp ayyyyyyySCP//Pp           syY//C    | Version 2.4.5
 AYAsAYYYYYYYY///Ps              cY//S   |
         pCCCCY//p          cSSps y//Y   | https://github.com/secdev/scapy
         SPPPP///a          pP///AC//Y   |
              A//A            cyP////C   | Have fun!
              p///Ac            sC///a   |
              P////YCpc           A//A   | Craft packets like I craft my beer.
       scccccp///pSP///p          p//Y   |               -- Jean De Clerck
      sY/////////y  caa           S//P   |
       cayCyayP//Ya              pY/Ya
        sY/PsY////YCc          aC//Yp
         sc  sccaCY//PCypaapyCP//YSs
                  spCPY//////YPSps
                       ccaacs
                                       using IPython 8.3.0
>>> send(IP(src="192.168.7.101",dst="192.168.7.104")/ICMP()/"Emergency Msg:Engine Failure")
.
Sent 1 packets.
>>> send(IP(src="192.168.7.101",dst="192.168.7.104")/ICMP()/"Emergency Msg:Engine Failure")
.
Sent 1 packets.
>>> 
```

Figure 30. Scapy: packet creation.



```
5 4.082965… 192.168.7.101  192.168.7.104  ICMP   70 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (reply in …
6 4.082982… 192.168.7.104  192.168.7.101  ICMP   70 Echo (ping) reply    id=0x0000, seq=0/0, ttl=64 (request i…
› Frame 5: 70 bytes on wire (560 bits), 70 bytes captured (560 bits) on interface s22-eth4, id 0
› Ethernet II, Src: 00:00:00_00:00:01 (00:00:00:00:00:01), Dst: 00:00:00_00:00:07 (00:00:00:00:00:07)
› Internet Protocol Version 4, Src: 192.168.7.101, Dst: 192.168.7.104
› Internet Control Message Protocol
0000  00 00 00 00 00 07 00 00  00 00 00 01 08 00 45 00   ········ ······E·
0010  00 38 00 01 00 00 40 01  ea a6 c0 a8 07 65 c0 a8   ·8····@· ·····e··
0020  07 68 08 00 cd c1 00 00  00 00 45 6d 65 72 67 65   ·h······ ··Emerge
0030  6e 63 79 20 4d 73 67 3a  45 6e 67 69 6e 65 20 46   ncy Msg: Engine F
0040  61 69 6c 75 72 65                                  ailure
```

Figure 31. Wireshark: Scapy packet with a message.

The Python scripts given below represent sending critical data from node Train1 to RailServer using ICMP protocol.

Listing 9. Sending a Critical Message Using Scapy with Python.

```
#! /usr/bin/env python
# The following line will import all Scapy modules
from scapy.all import *
i = 1
while i < 50:
          send(IP(src=''192.168.7.101'' ,dst=''192.168.7.104'')/ICMP()
          /''Emergency_Msg:Engine_Failure'')

    i += 1
    print(''Train1_is_sending_msg_to_RailServer'')
```

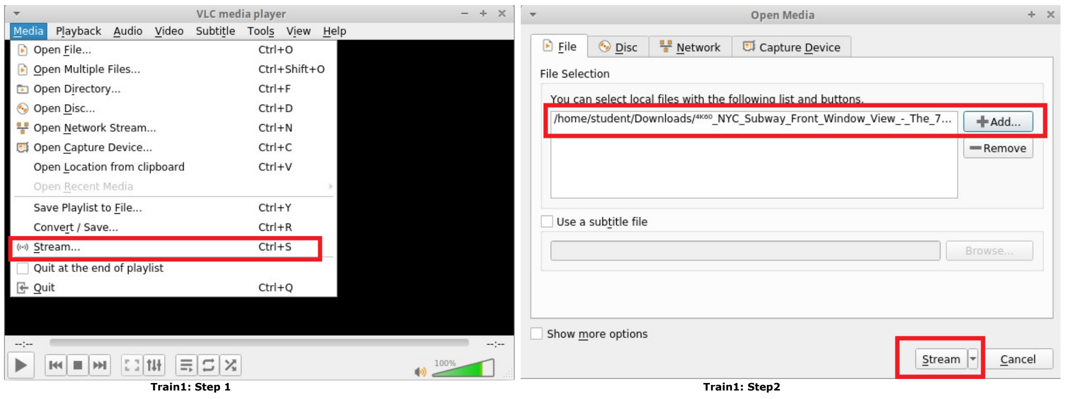
9.8. Video Transmission Test

To demonstrate the video transmission for railway and road coexistence scenarios, VLC player is used. A detailed procedure is elaborated below with figures to transmit a video file from node Train1 to RailServer to manifest video communication or critical video communication. This is implemented by opening the xterm window for Train1 and Railserver using the command *xterm Train1 Railserver* and running the command *vlc-wrapper &* from both instances of xterm. Figure 32 shows the initial window after running the command, where Train1 is configured to transmit the video to the node RailServer.

- **VLC player configuration at Train1:** Train1 is acting as the video transmitter. Click on the option "Media", select the option "Stream", and add the video file as shown in Figure 33. After selecting the option "Stream", a new window will pop up that provides the information about the selected video file; select the streaming method as "RTP/MPEP Transport Stream", as shown in Figure 34.
  Add the destination host's IP address. Here, the destination IP address is 192.168.7.104, which belongs to RailServer. In the next window, as shown in Figure 35, select the transcoding option "video-H.256 + MP3(MP4)"; other options can be selected based on the user's transmission video format.
  Now, select the option "Stream", as shown in Figure 36 and start streaming the video from Train1 to RailServer.
- **VLC player configuration at RailServer:** From "Media", select option "Open Network Stream", and a new window will pop up; add the network URL as "rtp://@:5004", as shown in Figure 37.

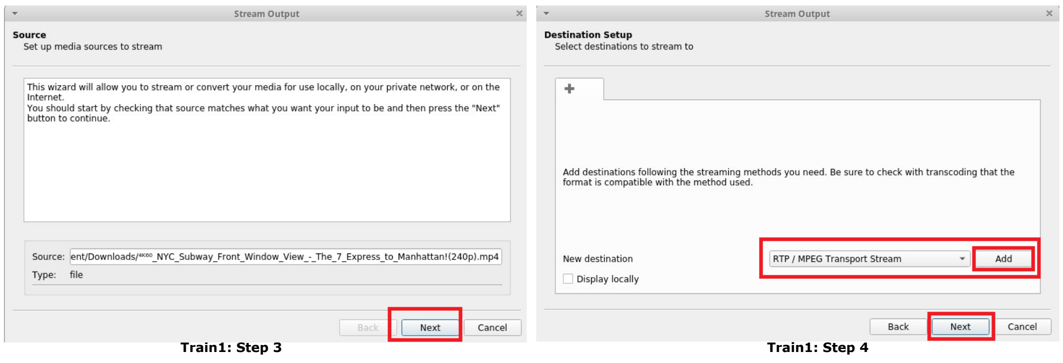
Figure 38 shows that Train1 is streaming the video and RailServer is receiving the video. Therefore, it can be concluded that by using the VLC player, a user can demonstrate video transmission from one host to another host in a railway and road coexistence scenario.

Figure 32. VLC player execution At Train1 and RailServer.


VLC media player
Media Playback Audio Video Subtitle Tools View Help
Open File... Ctrl+O
Open Multiple Files... Ctrl+Shift+O
Open Directory... Ctrl+F
Open Disc... Ctrl+D
Open Network Stream... Ctrl+N
Open Capture Device... Ctrl+C
Open Location from clipboard Ctrl+V
Open Recent Media
Save Playlist to File... Ctrl+Y
Convert / Save... Ctrl+R
Stream... Ctrl+S
Quit at the end of playlist
Quit Ctrl+Q
Train1: Step 1
Open Media
File Disc Network Capture Device
File Selection
Add...
Remove
Use a subtitle file
Browse...
Show more options
Stream
Cancel
Train1: Step2

You can select local files with the following list and buttons.
/home/student/Downloads/4K60_NYC_Subway_Front_Window_View_-_The_7...

Figure 33. Train1: adding the video file to transmit.



Source
Set up media sources to stream

This wizard will allow you to stream or convert your media for use locally, on your private network, or on the Internet.
You should start by checking that source matches what you want your input to be and then press the "Next" button to continue.

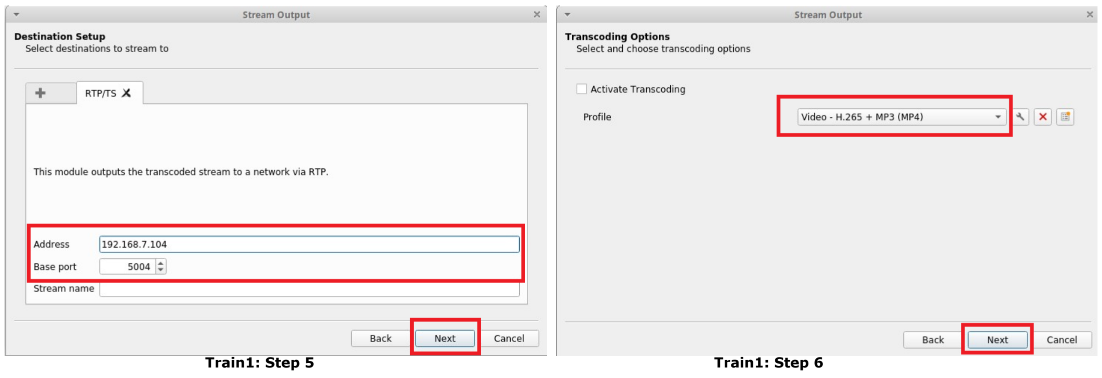
Source: ent/Downloads/⁴ᴷ⁶⁰_NYC_Subway_Front_Window_View_-_The_7_Express_to_Manhattan!(240p).mp4
Type: file

Destination Setup
Select destinations to stream to

Add destinations following the streaming methods you need. Be sure to check with transcoding that the format is compatible with the method used.

New destination: RTP / MPEG Transport Stream
Display locally

Figure 34. Train1: selecting the streaming method.



Destination Setup
Select destinations to stream to

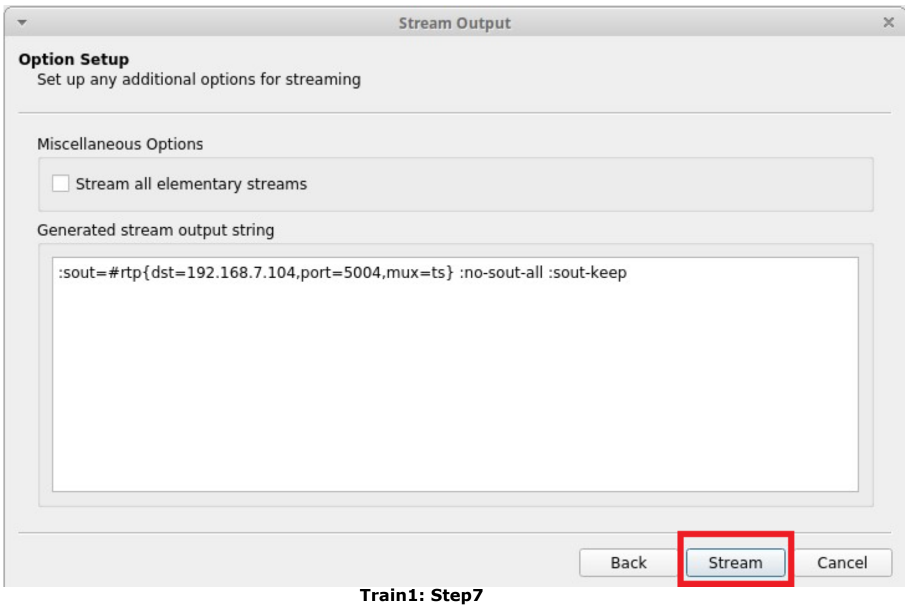
This module outputs the transcoded stream to a network via RTP.

Address: 192.168.7.104
Base port: 5004
Stream name

Transcoding Options
Select and choose transcoding options

Activate Transcoding
Profile: Video - H.265 + MP3 (MP4)

Figure 35. Train1: adding the destination IP address of RailServer.



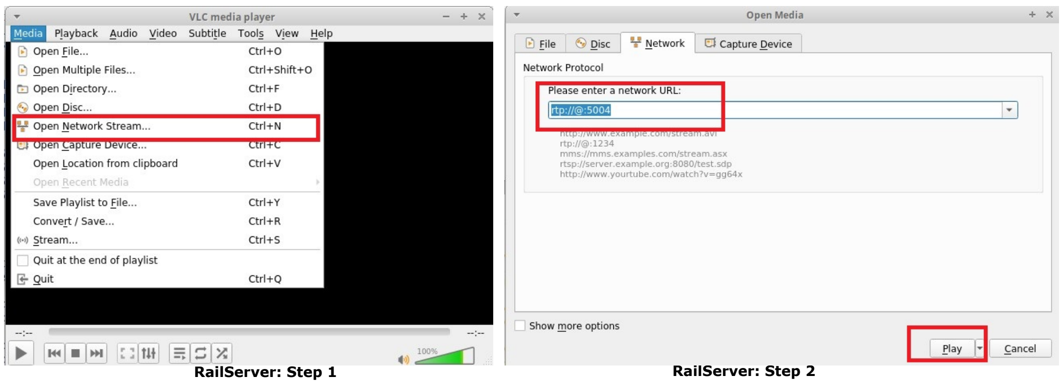
Option Setup
Set up any additional options for streaming

Miscellaneous Options
Stream all elementary streams

Generated stream output string

```
:sout=#rtp{dst=192.168.7.104,port=5004,mux=ts} :no-sout-all :sout-keep
```

Figure 36. Train1: stream the video.


VLC media player
Media Playback Audio Video Subtitle Tools View Help
Open File... Ctrl+O
Open Multiple Files... Ctrl+Shift+O
Open Directory... Ctrl+F
Open Disc... Ctrl+D
Open Network Stream... Ctrl+N
Open Capture Device... Ctrl+C
Open Location from clipboard Ctrl+V
Open Recent Media
Save Playlist to File... Ctrl+Y
Convert / Save... Ctrl+R
Stream... Ctrl+S
Quit at the end of playlist
Quit Ctrl+Q
RailServer: Step 1
Open Media
File Disc Network Capture Device
Network Protocol
Show more options
Play Cancel
RailServer: Step 2

Please enter a network URL: rtp://@:5004

Figure 37. Train1: stream the video.

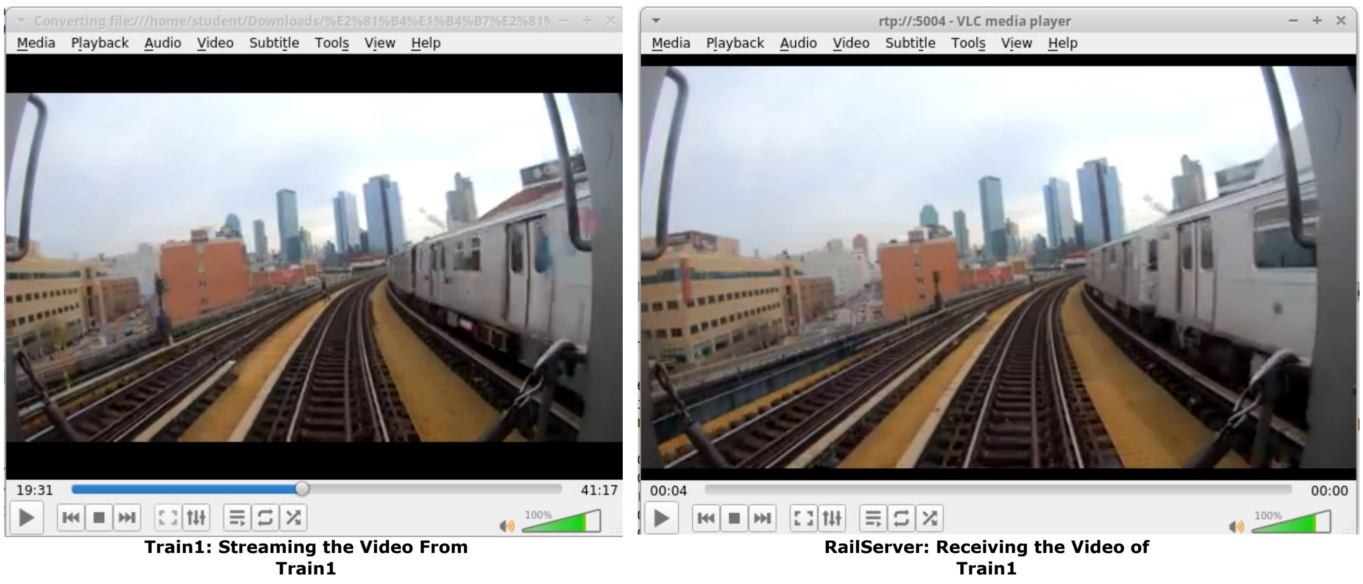

Converting file:///home/student/Downloads/%E2%81%B4%E1%B4%B7%E2%81%
Media Playback Audio Video Subtitle Tools View Help
19:31
41:17
Train1: Streaming the Video From Train1
rtp://:5004 - VLC media player
Media Playback Audio Video Subtitle Tools View Help
00:04
00:00
RailServer: Receiving the Video of Train1

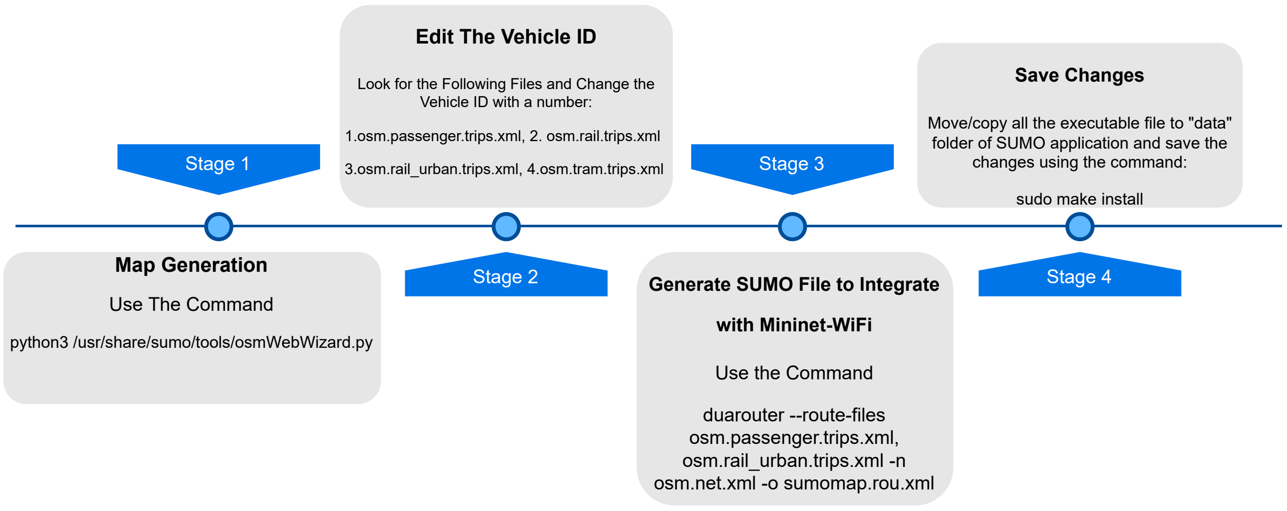
Figure 38. Video data transmission and reception [39].

10. SUMO Integration

In this section, a detailed description of SUMO integration with Mininet–WiFi is given to showcase the graphical representation of railway and road coexistence scenarios. Figure 39 shows the steps to generate the SUMO map from open street map and integration with Mininet–WiFi.

- **Map Generation:** To show the coexistence scenario of railways and roads in a pictorial form, user can design the desired map. A detailed tutorial is presented in [40]. For this practical work, we download the desired Google map file. To generate the desired emulation map on Google street map, run the command *python3 /usr/share/sumo/-tools/osmWebWizard.py* from the folder where you want to download the map files. After running the command, the "OSM Web Wizard for SUMO" page will open. Go to the "Generate Scenario" option given on the right side of the page and select the map location by providing the city or place name or by providing the GPS coordinates of the region. Figure 40 shows the OSM Web Wizard home page. To demonstrate SUMO integration, we select the place "Puente De Santiago" since it has tracks parallel to roads. The map area is selected using the option "Select Area". Figure 41 shows the selected map area and selected parameters to generate the vehicle traffic. Using the parameter "Through Traffic Factor", the user can define the number of vehicles that depart and arrive at the selected simulation boundary area.

After selecting the required parameters, by clicking on the "Generate Scenario" option, all the map files will be downloaded to the selected folder. After completion of map file generation, the user will get the messages shown in Figure 42. If all the steps are correct and the intended map files are downloaded, a new window will pop up and start running the simulation, as shown in Figure 43.

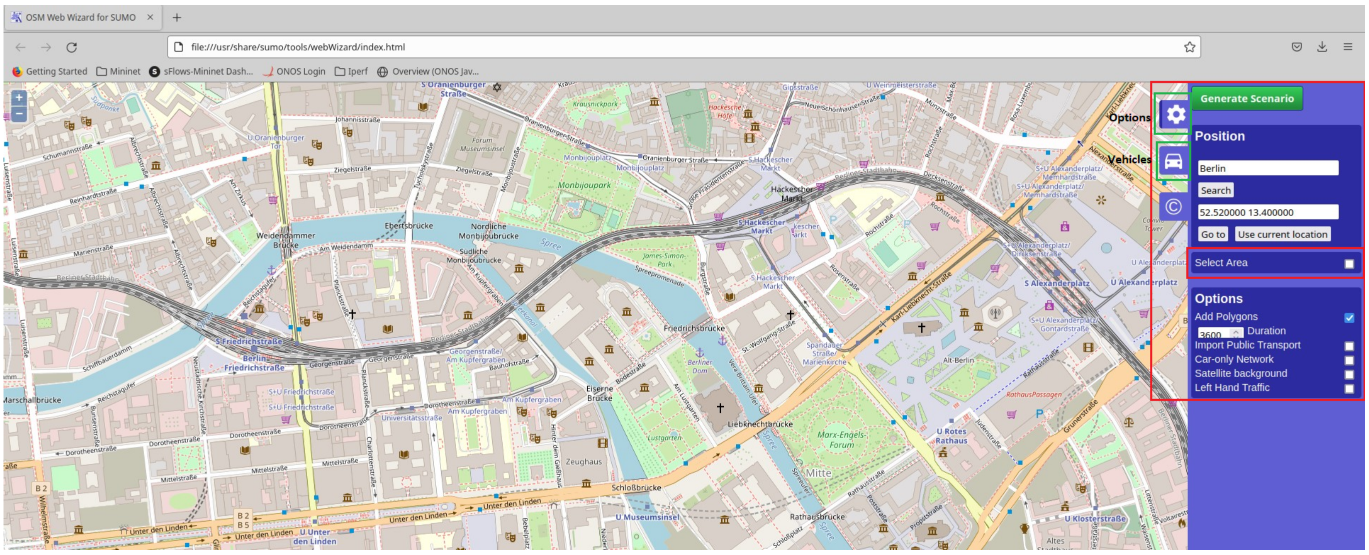

Stage 1
Map Generation
Use The Command
Stage 2
Edit The Vehicle ID
Look for the Following Files and Change the Vehicle ID with a number:
1.osm.passenger.trips.xml, 2. osm.rail.trips.xml
3.osm.rail_urban.trips.xml, 4.osm.tram.trips.xml
Stage 3
Generate SUMO File to Integrate with Mininet-WiFi
Use the Command
Stage 4
Save Changes
Move/copy all the executable file to "data" folder of SUMO application and save the changes using the command:

```
python3 /usr/share/sumo/tools/osmWebWizard.py
```

```
duarouter --route-files osm.passenger.trips.xml, osm.rail_urban.trips.xml -n osm.net.xml -o sumomap.rou.xml
```

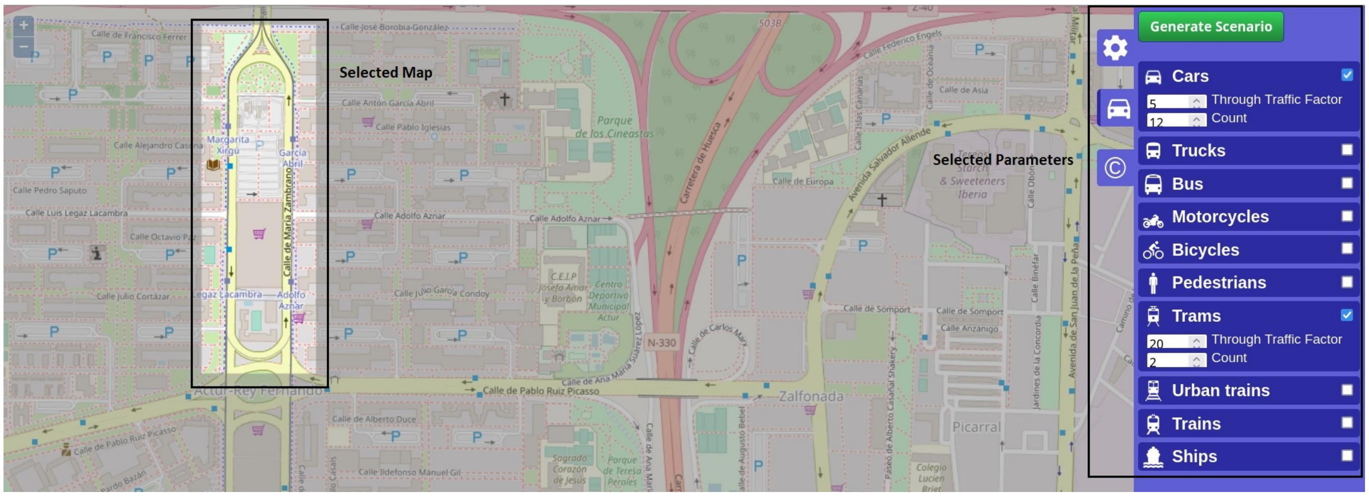
```
sudo make install
```

Figure 39. SUMO integration steps.



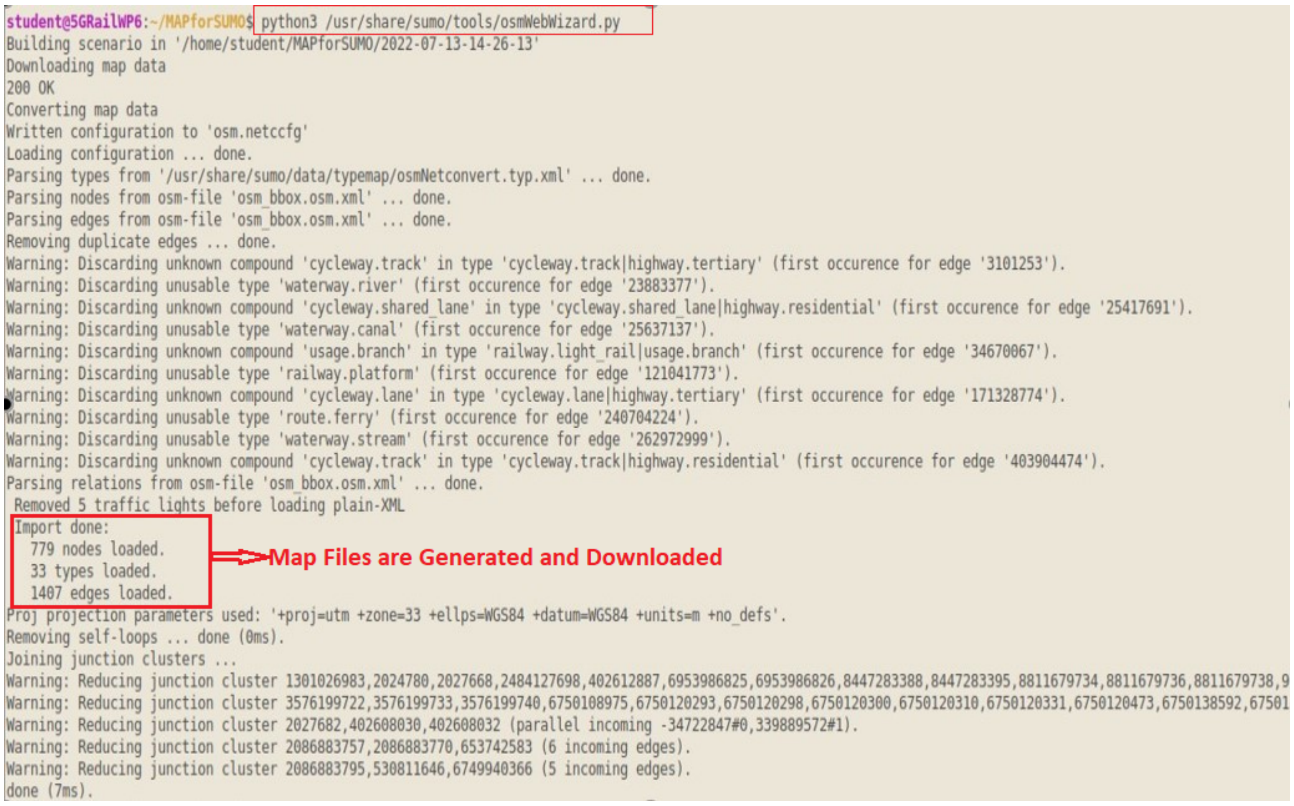
Berlin
52.520000 13.400000
3600

Figure 40. OSM Web Wizard home page.



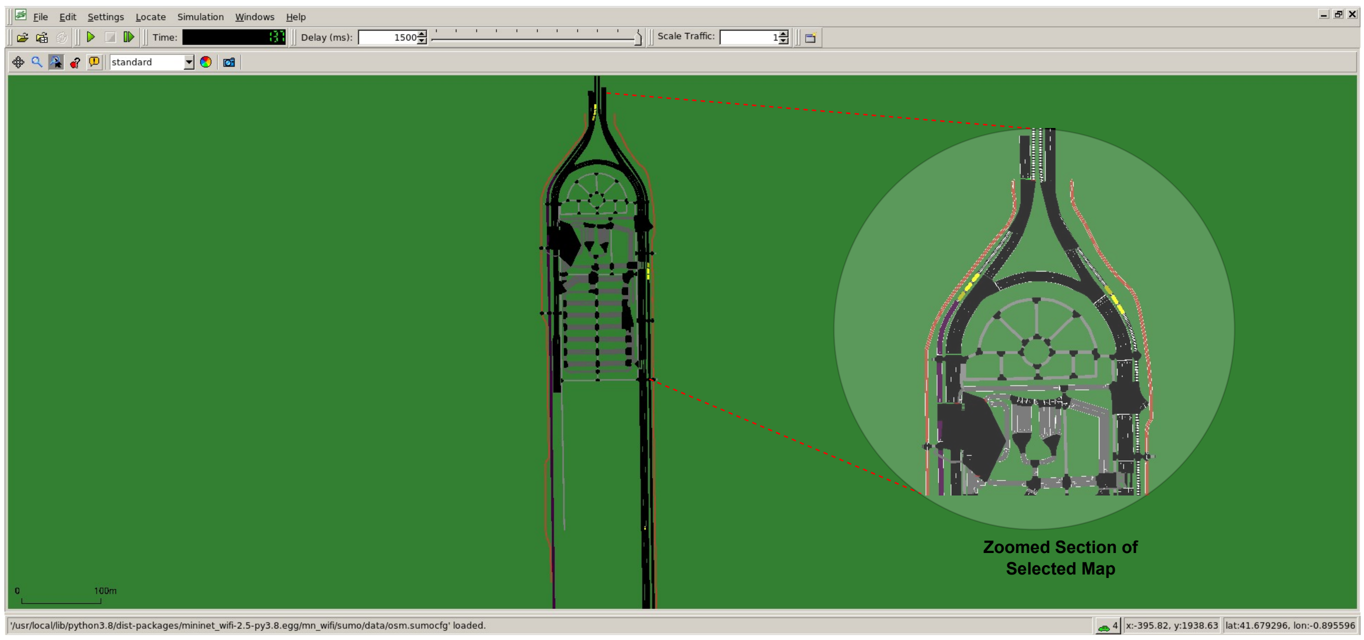
Cars ✓
5 Through Traffic Factor
12 Count
Trams ✓
20 Through Traffic Factor
2 Count

Figure 41. Selected map: Puente de Santiago.



```
student@5GRailWP6:~/MAPforSUMO$ python3 /usr/share/sumo/tools/osmWebWizard.py
Building scenario in '/home/student/MAPforSUMO/2022-07-13-14-26-13'
Downloading map data
200 OK
Converting map data
Written configuration to 'osm.netccfg'
Loading configuration ... done.
Parsing types from '/usr/share/sumo/data/typemap/osmNetconvert.typ.xml' ... done.
Parsing nodes from osm-file 'osm_bbox.osm.xml' ... done.
Parsing edges from osm-file 'osm_bbox.osm.xml' ... done.
Removing duplicate edges ... done.
Warning: Discarding unknown compound 'cycleway.track' in type 'cycleway.track|highway.tertiary' (first occurence for edge '3101253').
Warning: Discarding unusable type 'waterway.river' (first occurence for edge '23883377').
Warning: Discarding unknown compound 'cycleway.shared_lane' in type 'cycleway.shared_lane|highway.residential' (first occurence for edge '25417691').
Warning: Discarding unusable type 'waterway.canal' (first occurence for edge '25637137').
Warning: Discarding unknown compound 'usage.branch' in type 'railway.light_rail|usage.branch' (first occurence for edge '34670067').
Warning: Discarding unusable type 'railway.platform' (first occurence for edge '121041773').
Warning: Discarding unknown compound 'cycleway.lane' in type 'cycleway.lane|highway.tertiary' (first occurence for edge '171328774').
Warning: Discarding unusable type 'route.ferry' (first occurence for edge '240704224').
Warning: Discarding unusable type 'waterway.stream' (first occurence for edge '262972999').
Warning: Discarding unknown compound 'cycleway.track' in type 'cycleway.track|highway.residential' (first occurence for edge '403904474').
Parsing relations from osm-file 'osm_bbox.osm.xml' ... done.
 Removed 5 traffic lights before loading plain-XML
 Import done:
   779 nodes loaded.
   33 types loaded.
   1407 edges loaded.
Proj projection parameters used: '+proj=utm +zone=33 +ellps=WGS84 +datum=WGS84 +units=m +no_defs'.
Removing self-loops ... done (0ms).
Joining junction clusters ...
Warning: Reducing junction cluster 1301026983,2024780,2027668,2484127698,402612887,6953986825,6953986826,8447283388,8447283395,8811679734,8811679736,8811679738,9
Warning: Reducing junction cluster 3576199722,3576199733,3576199740,6750108975,6750120293,6750120298,6750120300,6750120310,6750120331,6750120473,6750138592,67501
Warning: Reducing junction cluster 2027682,402608030,402608032 (parallel incoming -34722847#0,339889572#1).
Warning: Reducing junction cluster 2086883757,2086883770,653742583 (6 incoming edges).
Warning: Reducing junction cluster 2086883795,530811646,6749940366 (5 incoming edges).
done (7ms).
```

Figure 42. Map generation.



'/usr/local/lib/python3.8/dist-packages/mininet_wifi-2.5-py3.8.egg/mn_wifi/sumo/data/osm.sumocfg' loaded.

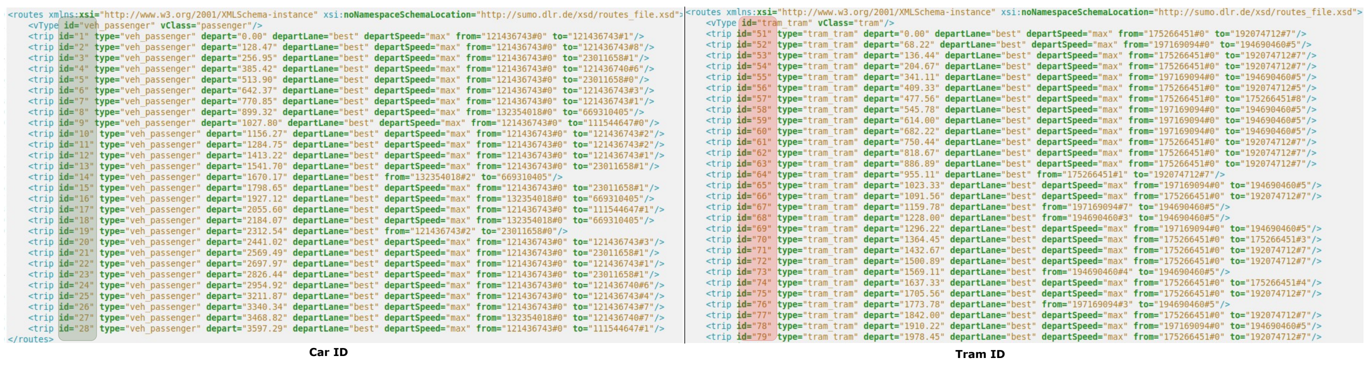
Figure 43. Simulation: Puente de Santiago

- **Edit the Vehicle Id, i.e., Car ID and Train/Tram ID:** To show the vehicles in SUMO and to interface with Mininet–WiFi, the selected vehicles/nodes should have different IDs. Go to the folder where map files are generated and look for the files that have the car and the tram. Look for the following emulation files:
    1. **osm.passenger.trips.xml**: contains the car ID;
    2. **osm.rail.trips.xml**: contains train ID (long route trains);
    3. **osm.rail_urban.trips.xml**: contains urban train ID;
    4. **osm.tram.trips.xml**: contains tram ID.

  In this selected map, a car and a tram are simulated as network nodes. Open the files *osm.passenger.trips.xml* and *osm.tram.trips.xml* and change the IDs with numerical numbers such as 1, 2, 3, etc.; the number should not be repeated; every entity should have a different number, as shown in Figure 44. After that, to build a single executable file, open a terminal in the folder where all these files are saved and run the command:

  duarouter - -route-files osm.passenger.trips.xml, osm.rail_urban.trips.xml -n osm.net.xml -o sumomap.rou.xml

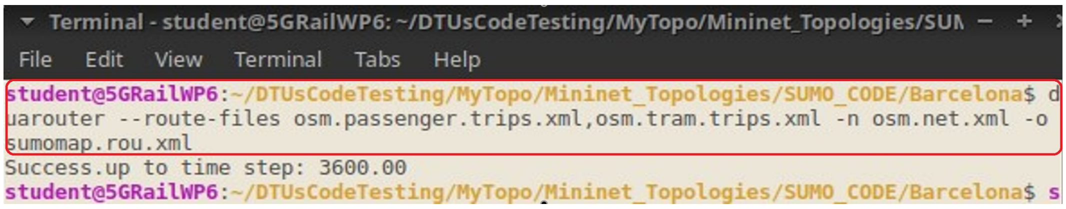
  where *sumomap.rou.xml* is the executable file. Figure 45 shows that the *sumomap.rou.xml* file is generated.



```
<routes xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="http://sumo.dlr.de/xsd/routes_file.xsd">
    <vType id="veh_passenger" vClass="passenger"/>
    <trip id="1" type="veh_passenger" depart="0.00" departLane="best" departSpeed="max" from="121436743#0" to="121436743#1"/>
    <trip id="2" type="veh_passenger" depart="128.47" departLane="best" departSpeed="max" from="121436743#0" to="121436743#8"/>
    <trip id="3" type="veh_passenger" depart="256.95" departLane="best" departSpeed="max" from="121436743#0" to="23011658#1"/>
    <trip id="4" type="veh_passenger" depart="385.42" departLane="best" departSpeed="max" from="121436743#0" to="121436740#6"/>
    <trip id="5" type="veh_passenger" depart="513.90" departLane="best" departSpeed="max" from="121436743#0" to="23011658#0"/>
    <trip id="6" type="veh_passenger" depart="642.37" departLane="best" departSpeed="max" from="121436743#0" to="121436743#3"/>
    <trip id="7" type="veh_passenger" depart="770.85" departLane="best" departSpeed="max" from="121436743#0" to="121436743#1"/>
    <trip id="8" type="veh_passenger" depart="899.32" departLane="best" departSpeed="max" from="132354018#0" to="669310405"/>
    <trip id="9" type="veh_passenger" depart="1027.80" departLane="best" departSpeed="max" from="121436743#0" to="111544647#0"/>
    <trip id="10" type="veh_passenger" depart="1156.27" departLane="best" departSpeed="max" from="121436743#0" to="121436743#2"/>
    <trip id="11" type="veh_passenger" depart="1284.75" departLane="best" departSpeed="max" from="121436743#0" to="121436743#2"/>
    <trip id="12" type="veh_passenger" depart="1413.22" departLane="best" departSpeed="max" from="121436743#0" to="121436743#1"/>
    <trip id="13" type="veh_passenger" depart="1541.70" departLane="best" departSpeed="max" from="121436743#0" to="23011658#1"/>
    <trip id="14" type="veh_passenger" depart="1670.17" departLane="best" from="132354018#2" to="669310405"/>
    <trip id="15" type="veh_passenger" depart="1798.65" departLane="best" departSpeed="max" from="121436743#0" to="23011658#1"/>
    <trip id="16" type="veh_passenger" depart="1927.12" departLane="best" departSpeed="max" from="132354018#0" to="669310405"/>
    <trip id="17" type="veh_passenger" depart="2055.60" departLane="best" departSpeed="max" from="121436743#0" to="111544647#1"/>
    <trip id="18" type="veh_passenger" depart="2184.07" departLane="best" departSpeed="max" from="132354018#0" to="669310405"/>
    <trip id="19" type="veh_passenger" depart="2312.54" departLane="best" from="121436743#2" to="23011658#0"/>
    <trip id="20" type="veh_passenger" depart="2441.02" departLane="best" departSpeed="max" from="121436743#0" to="121436743#3"/>
    <trip id="21" type="veh_passenger" depart="2569.49" departLane="best" departSpeed="max" from="121436743#0" to="23011658#1"/>
    <trip id="22" type="veh_passenger" depart="2697.97" departLane="best" departSpeed="max" from="121436743#0" to="121436743#1"/>
    <trip id="23" type="veh_passenger" depart="2826.44" departLane="best" departSpeed="max" from="121436743#0" to="23011658#1"/>
    <trip id="24" type="veh_passenger" depart="2954.92" departLane="best" departSpeed="max" from="121436743#0" to="121436740#6"/>
    <trip id="25" type="veh_passenger" depart="3211.87" departLane="best" departSpeed="max" from="121436743#0" to="121436743#4"/>
    <trip id="26" type="veh_passenger" depart="3340.34" departLane="best" departSpeed="max" from="121436743#0" to="121436743#7"/>
    <trip id="27" type="veh_passenger" depart="3468.82" departLane="best" departSpeed="max" from="132354018#0" to="121436740#7"/>
    <trip id="28" type="veh_passenger" depart="3597.29" departLane="best" departSpeed="max" from="121436743#0" to="111544647#1"/>
</routes>
```

```
<routes xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="http://sumo.dlr.de/xsd/routes_file.xsd">
    <vType id="tram_tram" vClass="tram"/>
    <trip id="51" type="tram_tram" depart="0.00" departLane="best" departSpeed="max" from="175266451#0" to="192074712#7"/>
    <trip id="52" type="tram_tram" depart="68.22" departLane="best" departSpeed="max" from="197169094#0" to="194690460#5"/>
    <trip id="53" type="tram_tram" depart="136.44" departLane="best" departSpeed="max" from="175266451#0" to="192074712#7"/>
    <trip id="54" type="tram_tram" depart="204.67" departLane="best" departSpeed="max" from="175266451#0" to="192074712#7"/>
    <trip id="55" type="tram_tram" depart="341.11" departLane="best" departSpeed="max" from="197169094#0" to="194690460#5"/>
    <trip id="56" type="tram_tram" depart="409.33" departLane="best" departSpeed="max" from="175266451#0" to="192074712#5"/>
    <trip id="57" type="tram_tram" depart="477.56" departLane="best" departSpeed="max" from="175266451#0" to="175266451#8"/>
    <trip id="58" type="tram_tram" depart="545.78" departLane="best" departSpeed="max" from="197169094#0" to="194690460#5"/>
    <trip id="59" type="tram_tram" depart="614.00" departLane="best" departSpeed="max" from="197169094#0" to="194690460#5"/>
    <trip id="60" type="tram_tram" depart="682.22" departLane="best" departSpeed="max" from="197169094#0" to="194690460#5"/>
    <trip id="61" type="tram_tram" depart="750.44" departLane="best" departSpeed="max" from="175266451#0" to="192074712#7"/>
    <trip id="62" type="tram_tram" depart="818.67" departLane="best" departSpeed="max" from="175266451#0" to="192074712#7"/>
    <trip id="63" type="tram_tram" depart="886.89" departLane="best" departSpeed="max" from="175266451#0" to="192074712#7"/>
    <trip id="64" type="tram_tram" depart="955.11" departLane="best" from="175266451#1" to="192074712#7"/>
    <trip id="65" type="tram_tram" depart="1023.33" departLane="best" departSpeed="max" from="197169094#0" to="194690460#5"/>
    <trip id="66" type="tram_tram" depart="1091.56" departLane="best" departSpeed="max" from="175266451#0" to="192074712#7"/>
    <trip id="67" type="tram_tram" depart="1159.78" departLane="best" from="197169094#7" to="194690460#5"/>
    <trip id="68" type="tram_tram" depart="1228.00" departLane="best" from="194690460#3" to="194690460#5"/>
    <trip id="69" type="tram_tram" depart="1296.22" departLane="best" departSpeed="max" from="197169094#0" to="194690460#5"/>
    <trip id="70" type="tram_tram" depart="1364.45" departLane="best" departSpeed="max" from="175266451#0" to="175266451#3"/>
    <trip id="71" type="tram_tram" depart="1432.67" departLane="best" departSpeed="max" from="175266451#0" to="192074712#7"/>
    <trip id="72" type="tram_tram" depart="1500.89" departLane="best" departSpeed="max" from="175266451#0" to="192074712#7"/>
    <trip id="73" type="tram_tram" depart="1569.11" departLane="best" from="194690460#4" to="194690460#5"/>
    <trip id="74" type="tram_tram" depart="1637.33" departLane="best" departSpeed="max" from="175266451#0" to="175266451#4"/>
    <trip id="75" type="tram_tram" depart="1705.56" departLane="best" departSpeed="max" from="175266451#0" to="192074712#7"/>
    <trip id="76" type="tram_tram" depart="1773.78" departLane="best" from="197169094#3" to="194690460#5"/>
    <trip id="77" type="tram_tram" depart="1842.00" departLane="best" departSpeed="max" from="175266451#0" to="192074712#7"/>
    <trip id="78" type="tram_tram" depart="1910.22" departLane="best" departSpeed="max" from="197169094#0" to="194690460#5"/>
    <trip id="79" type="tram_tram" depart="1978.45" departLane="best" departSpeed="max" from="175266451#0" to="192074712#7"/>
```

Figure 44. Change IDs of vehicles.



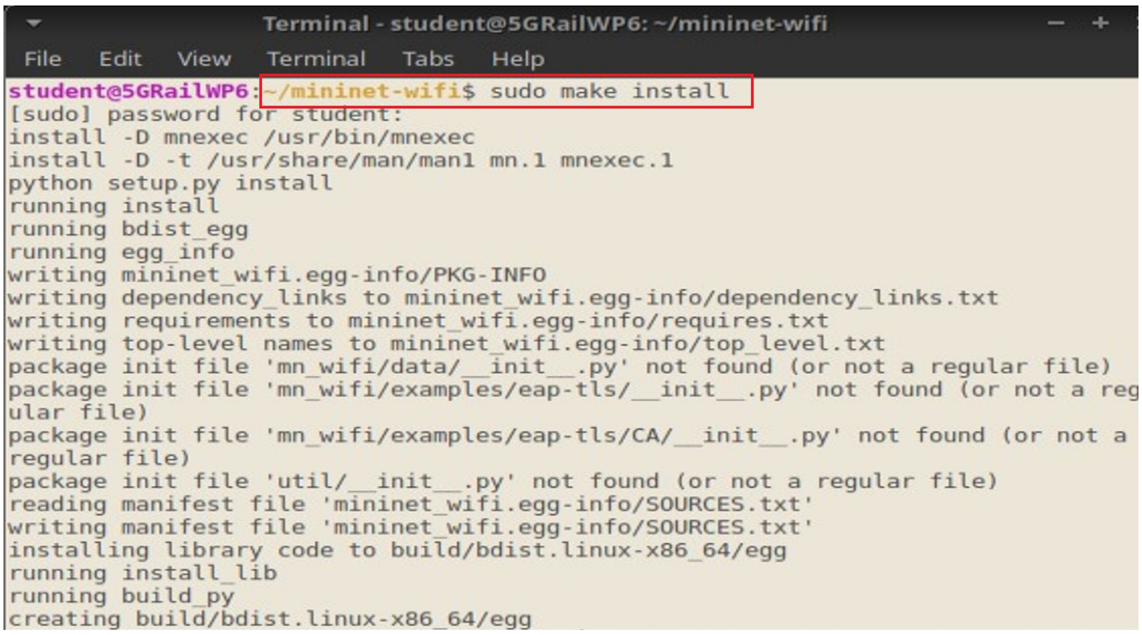
```
Terminal - student@5GRailWP6:~/DTUsCodeTesting/MyTopo/Mininet_Topologies/SUN
File  Edit  View  Terminal  Tabs  Help
student@5GRailWP6:~/DTUsCodeTesting/MyTopo/Mininet_Topologies/SUMO_CODE/Barcelona$ d
uarouter --route-files osm.passenger.trips.xml,osm.tram.trips.xml -n osm.net.xml -o
sumomap.rou.xml
Success.up to time step: 3600.00
student@5GRailWP6:~/DTUsCodeTesting/MyTopo/Mininet_Topologies/SUMO_CODE/Barcelona$ s
```

Figure 45. Change IDs of vehicles.

- **Edit the File *osm.sumocfg*:** After generating the map file *sumomap.rou.xml*, open the file *osm.sumocfg* and change the following information:

  <route-files value="sumomap.rou.xml"/>

- **Move or Copy the Map Files:** By default, Mininet–WiFi accesses the SUMO files from the "data" folder of the SUMO application. Therefore, after completing all the steps mentioned above, move or copy the map files in the /home/student/mininet-wifi/mn_wifi/
  sumo/data folder.
- **Save all Changes:** Now, go to the folder for Mininet–WiFi, open a terminal, and run the following command *sudo make install* to save all the changes done with SUMO files, as shown in Figure 46.



```
Terminal - student@5GRailWP6: ~/mininet-wifi
File  Edit  View  Terminal  Tabs  Help
student@5GRailWP6:~/mininet-wifi$ sudo make install
[sudo] password for student:
install -D mnexec /usr/bin/mnexec
install -D -t /usr/share/man/man1 mn.1 mnexec.1
python setup.py install
running install
running bdist_egg
running egg_info
writing mininet_wifi.egg-info/PKG-INFO
writing dependency_links to mininet_wifi.egg-info/dependency_links.txt
writing requirements to mininet_wifi.egg-info/requires.txt
writing top-level names to mininet_wifi.egg-info/top_level.txt
package init file 'mn_wifi/data/__init__.py' not found (or not a regular file)
package init file 'mn_wifi/examples/eap-tls/__init__.py' not found (or not a reg
ular file)
package init file 'mn_wifi/examples/eap-tls/CA/__init__.py' not found (or not a
regular file)
package init file 'util/__init__.py' not found (or not a regular file)
reading manifest file 'mininet_wifi.egg-info/SOURCES.txt'
writing manifest file 'mininet_wifi.egg-info/SOURCES.txt'
installing library code to build/bdist.linux-x86_64/egg
running install_lib
running build_py
creating build/bdist.linux-x86_64/egg
```

Figure 46. Save all the changes.

- **SUMO Map Integration with Selected Network Topology:** To generate the network topology integrated with SUMO maps, a Python script is used utilizing all the methods and functions described in Section 6.1. The user has to add the following line of code mentioned in List 10 to the network topology Python script. The command *net.useExternalProgram* is used to integrate the SUMO application with Mininet–WiFi. The entire Python script (*SUMO_Aug17.py*) for network topology creation with SUMO map integration is available at [26].

Listing 10. SUMO Integration with Mininet–WiFi.

```
info(''****Starting_network_and_connecting_to_traci'')
    info('Connecting_to_traci_sumo')

    net.useExternalProgram(program=sumo, port=8813,
        config_file='osm.sumocfg',
        extra_params=[''--start --delay 1500''])
```

After running the script *SUMO_Aug17.py*, network emulation with the SUMO map as shown in Figure 43 will be generated. Figure 47 shows the location of the assigned access points and the movement of cars and trams, where assigned access points are ap1 to ap17, C2 represents Car2, and T77 represents Tram77. When a car/tram moves from the coverage range of one access point to another, we sometimes observe that there is no automatic connection to the nearest access point. In that case, the connection can be established using the command <*node name*> *iw dev* <*node name*>*-wlan0 connect* <*SSID name*>.

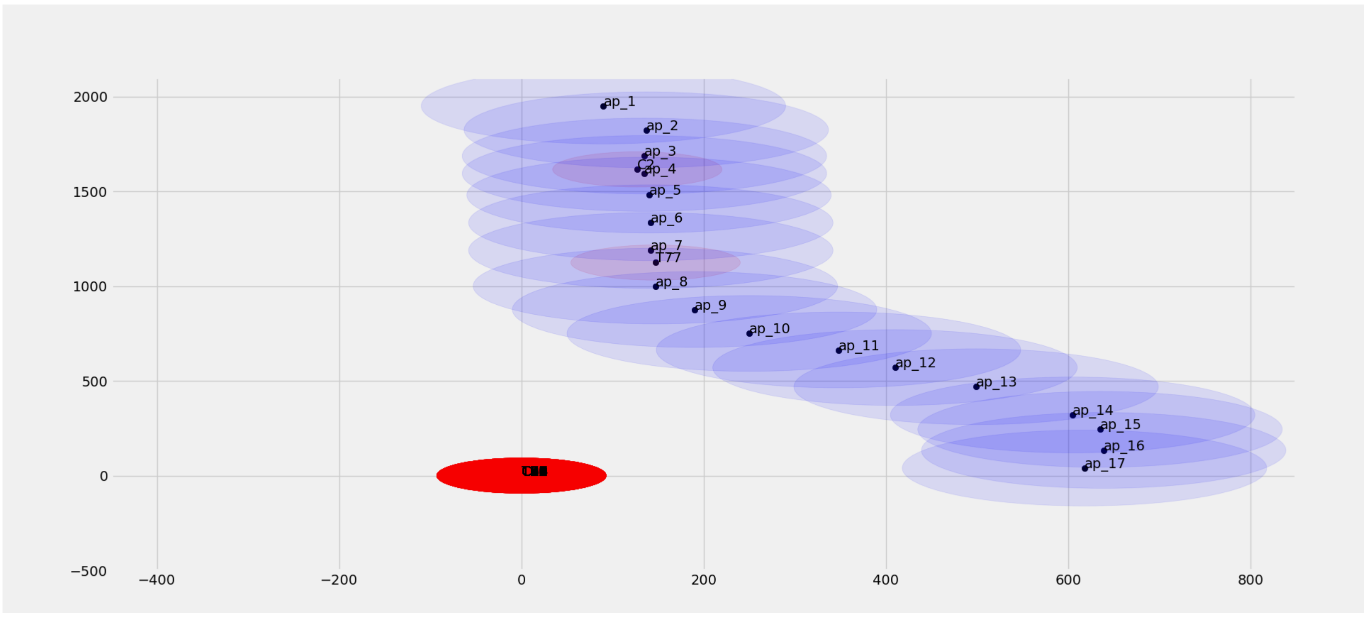


Figure 47. Movement of trams and cars on SUMO map in the range of assigned access points.

11. Conclusions

In this paper, we have provided a detailed tutorial to emulate the coexistence of railway and road services by sharing telecommunication infrastructure using the tools Mininet–WiFi, ONOS-SDN Controller, and SUMO. Based on the validation test carried out in Section 9, it can be concluded that by using Mininet–WiFi, a user can develop different network topologies with nodes having moving capabilities and wireless access points intended for railway and road coexistence scenarios. Therefore, moving hosts are able to replicate cars, rails, and trams in considered virtual space. As we have stated, Mininet–WiFi does not support 5G-based network emulation. In our previous work [16], we presented a network emulator "Emu5GNet". This emulation tool allows the emulation of 5G networks with complex applications. The developed ONOS SDN application is able to differentiate data traffic based on the VLAN tag. Along with this, the developed SDN application is able to handle the handover scenarios. The visualization tool SUMO is capable of representing the simulation of a considered scenario in a pictorial/graphical way. After executing the Mininet–WiFi network emulation file with SUMO, there are sometimes no automatic connections of nodes to the nearest WiFi access points when they move from one access point to another. This may be because of a coverage range problem where vehicles are moving. Since we have extracted the SUMO map from open street map, we do not know exactly the design and shape of the road. This is subjected to only SUMO integration. To mimic real data traffic for the considered coexistence scenarios for railways and roads, iperf3, Scapy, and VLC player are considered. Using iperf3, standard data communication is demonstrated by sending and receiving UDP and TCP packets between nodes. Using Scapy, messaging and critical data communication are manifested. Video transmission from one network node to another node is demonstrated using VLC Player. The MTR tool is used to measure the network parameters of latency, packet loss, and jitter. Therefore, it can be concluded that the considered tools have the potential to emulate the considered scenarios for railway and road coexistence environments.

Author Contributions: Conceptualization, J.S., R.S., T.S., L.M. and M.B.; methodology, J.S. and R.S.; software, R.S. and J.S.; validation, J.S., R.S., T.S., L.M. and M.B.; formal analysis, R.S. and J.S.; investigation, R.S. and J.S.; resources, J.S., R.S., T.S., L.M. and M.B.; data curation, J.S., R.S., T.S., L.M. and M.B.; writing—original draft preparation, R.S. and J.S.; writing—review and editing, J.S., R.S., T.S., L.M. and M.B.; visualization, R.S. and J.S.; supervision, J.S.; project administration, J.S., L.M. and M.B.; funding acquisition, J.S., L.M. and M.B. All authors have read and agreed to the published version of the manuscript.

Funding: This empirical work is part of the project "5G for future RAILway mobile communication system" (5GRAIL). It is funded by the European Union's Horizon 2020 research and innovation program. The grant agreement number is 951725.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Sylla, T.; Mendiboure, L.; Maaloul, S.; Aniss, H.; Chalouf, M.A.; Delbruel, S. Multi-Connectivity for 5G Networks and Beyond: A Survey. *Sensors* **2022**, *22*, 7591. [CrossRef] [PubMed]
2. ETSI TR 103 333—System Reference Document (SRDoc); GSM-R Networks Evolution. February 2017. Available online: https://www.etsi.org/deliver/etsi_tr/103300_103399/103333/01.01.01_60/tr_103333v010101p.pdf (accessed on 5 October 2022).
3. UIC FU-7100—FRCMS User Requirements Specification. February 2020. Available online: https://uic.org/IMG/pdf/frmcs_user_requirements_specification-fu_7100-v5.0.0.pdf (accessed on 5 October 2022).
4. UIC- MG-7900—FRCMS Use Cases. February 2020. Available online: https://uic.org/IMG/pdf/frmcs_use_cases-mg_7900-v2.0.0.pdf (accessed on 5 October 2022).
5. Mandoc, D. FRMCS Definition, Specification and Standardization Activities ERA CCRCC 2019, Slide 3. Available online: https://www.era.europa.eu/sites/default/files/events-news/docs/ccrcc_2019/4-3_dan_mandoc_era_uic_frmcs_definition_16102019_en.pdf (accessed on 12 November 2021).
6. 5GRail Info on 5G PPP. Available online: https://5g-ppp.eu/5grail/ (accessed on 1 October 2022).
7. 5GRail Deliverable D6.1. Scenarios for Rail and Road Communication System Coexistence. Available online: https://5grail.eu/wp-content/uploads/2021/07/5GRAIL_202107023_R_PU_D6.1_RV1_UNI_EIFFEL_Scenarios_Rail_and_road.pdf (accessed on 10 September 2022).
8. Fontes, R.R.; Afzal, S.; Brito, S.H.B.; Santos, M.A.S.; Rothenberg, C.E. Mininet-WiFi: Emulating software-defined wireless networks. In Proceedings of the 2015 11th International Conference on Network and Service Management (CNSM), Barcelona, Spain, 9–13 November 2015; pp. 384–389. [CrossRef]
9. Zang, M.; Dittmann, L.; Yan, Y. A Performance Study of Flow-Based Monitoring in Internet of Vehicles. In Proceedings of the 2021 12th International Conference on Network of the Future (NoF), Coimbra, Portugal, 6–8 October 2021; pp. 1–5. [CrossRef]
10. Smida, K.; Tounsi, H.; Frikha, M.; Song, Y.-Q. Efficient SDN Controller for Safety Applications in SDN-Based Vehicular Networks: POX, Floodlight, ONOS or OpenDaylight? In Proceedings of the 2020 IEEE Eighth International Conference on Communications and Networking (ComNet), Hammamet, Tunisia, 27–30 October 2020; pp. 1–6. [CrossRef]
11. Fontes, R.D.R.; Campolo, C.; Rothenberg, C.E.; Molinaro, A. From Theory to Experimental Evaluation: Resource Management in Software-Defined Vehicular Networks. *IEEE Access* **2017**, *5*, 3069–3076. [CrossRef]
12. Acharya, S.; Devanahalli, S.S.S.; Rawat, A.; Kuruvilla, V.P.; Sharma, P.; Amrutur, B.; Joglekar, A.; Krishnapuram, R.; Simmhan, Y.; Tyagi, H. Network Emulation For Tele-driving Application Development. In Proceedings of the 2021 International Conference on COMmunication Systems & NETworkS (COMSNETS), Bangalore, India, 5–9 January 2021; pp. 109–110. [CrossRef]
13. Abdelkader, G.; Elgazzar, K.; Khamis, A. Connected Vehicles: Technology Review, State of the Art, Challenges and Opportunities. *Sensors* **2021**, *21*, 7712. [CrossRef] [PubMed]
14. Monir, N.; Toraya, M.M.; Vladyko, A.; Muthanna, A.; Torad, M.A.; El-Samie, F.E.A.; Ateya, A.A. Seamless Handover Scheme for MEC/SDN-Based Vehicular Networks. *J. Sens. Actuator Netw.* **2022**, *11*, 9. [CrossRef]

15. Ali, H.D.; Abdulqader, A.H. Using Software Defined Network (SDN) Controllers to Enhance Communication between Two Vehicles in Vehicular AD HOC Network (VANET). In Proceedings of the 2021 7th International Conference on Contemporary Information Technology and Mathematics (ICCITM), Mosul, Iraq, 25–26 August 2021; pp. 106–111. [CrossRef]
16. 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.
17. Open Network Operating System (ONOS®). Available online: https://opennetworking.org/onos/ (accessed on 10 October 2022).
18. Mininet-WiFi Emulator for Software Defined Network. Available online: https://mininet-wifi.github.io/ (accessed on 10 September 2022).
19. 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, Barcelona, Spain, 23–29 October 2011.
20. Mininet An Instant Virtual Network on Your Laptop (or other PC). Available online: http://mininet.org/ (accessed on 15 October 2022).
21. mac80211_hwsim. Available online: https://wireless.wiki.kernel.org/en/users/drivers/mac80211_hwsim (accessed on 15 October 2022).
22. The User Manual. Available online: https://usermanual.wiki/Pdf/mininetwifidraftmanual.297704656.pdf (accessed on 11 September 2022).
23. OSMWebWizard. Available online: https://sumo.dlr.de/docs/Tutorials/OSMWebWizard.html#getting_started (accessed on 25 October 2022).
24. 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.
25. 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.
26. DTU5GRail /5GRail_WP6. Available online: https://github.com/DTU5GRail/5GRail_WP6/tree/main/DTU_Code/Considerd_Scenario/ (accessed on 18 September 2022).
27. Wireless B vs. G vs N vs AC | What is The Difference? Available online: https://homenetworkadmin.com/wireless-b-vs-g-vs-n-vs-ac-difference/ (accessed on 20 October 2022).
28. Iperf. Available online: https://en.wikipedia.org/wiki/Iperf (accessed on 19 September 2022).
29. What is iPerf/iPerf3? Available online: https://iperf.fr/ (accessed on 19 September 2022).
30. Introduction to Scapy? Available online: https://santandergto.com/en/guide-using-scapy-with-python/ (accessed on 21 September 2022).
31. Scapy Packet Crafting for Python2 and Python3. Available online: https://scapy.net/ (accessed on 21 September 2022).
32. Download and Installation. Available online: https://scapy.readthedocs.io/en/latest/installation.html (accessed on 21 September 2022).
33. How to do Video Streaming Using VLC Player. Available online: https://headendinfo.com/video-streaming-using-vlc-player/ (accessed on 25 September 2022).
34. VLC Features. Available online: https://www.videolan.org/vlc/features.html (accessed on 25 September 2022).
35. What is MTR & How to Use to Troubleshoot & Test your Connections. Available online: https://www.pcwdld.com/what-is-mtr-and-howto-troubleshoot-connections#wbounce-modal (accessed on 8 September 2022).
36. What is MTR and Why is it Useful? Available online: https://www.comparitech.com/net-admin/what-is-mtr/ (accessed on 8 September 2022).
37. ONOS Java API (2.5.0). Available online: https://api.onosproject.org/2.5.0/apidocs/ (accessed on 5 September 2022).
38. What Does VLAN Tagging Mean? Available online: https://www.techopedia.com/definition/32105/vlan-tagging (accessed on 7 September 2022).
39. NYC Subway Front Window View—The 7 Express to Manhattan! Available online: https://www.youtube.com/watch?v=ukOHqdPbYYg&list=WL&index=1&t=180s&ab_channel=DJHammersTrains (accessed on 12 October 2022).
40. Tutorials. Available online: https://sumo.dlr.de/docs/Tutorials/index.html (accessed on 5 October 2022).