SMART TRAFFIC MANAGEMENT SYSTEM BASED ON SOFTWARE DEFINED INTERNET OF THINGS ARCHITECTURE

A Project as a Course requirement for

Master of Technology in Optoelectronics and Communication

PRANAV GODWAY

17705

SRI SATHYA SAI INSTITUTE OF HIGHER LEARNING

(Deemed to be University)

Department of Physics

Prasanthi Nilayam Campus

March 2019

DEPARTMENT OF PHYSICS

Sri Sathya Sai Institute of Higher Learning,
Prasanthi Nilayam - 515134 (A.P)

CERTIFICATE

This is to certify that this Project Titled **SMART TRAFFIC MANAGEMENT SYSTEM BASED ON SOFTWARE DEFINED INTERNET OF THINGS ARCHITECTURE** submitted by Pranav Godway, 17705, Department of Physics, Prasanthi Nilayam Campus is a bonafide record of the original work done under my supervision as a Course requirement for the Degree of Master of Technology in Optoelectronics and Communication.

………………………………..
Dr. R Gowrishankar
Project Supervisor

Countersigned by
Dr. R Gowrishankar

Place:

………………………………...

Date:

Head of the Department

DECLARATION

The Project titled **SMART TRAFFIC MANAGEMENT SYSTEM BASED ON SOFTWARE DEFINED INTERNET OF THINGS ARCHITECTURE** was carried out by me under the supervision of Dr. R Gowrishankar, Department of Physics, Prasanthi Nilayam Campus as a Course requirement for the Degree of Master of Technology in optoelectronics and Communication and has not formed the basis for the award of any degree, diploma or any other such title by this or any other University

……………………………..

Place: Pranav Godway

Regd. No: 17705

II. M.Tech(OEC)

Date: Prasanthi Nilayam

DEDICATED AT THY LOTUS FEET

"END OF EDUCATION IS CHARACTER" – BHAGAWAN SRI SATHYA SAI BABA

Acknowledgements

I place this work as a humble offering to my Alma mater Bhagawan Sri Sathya Sai Baba, Revered Founder Chancellor, Sri Sathya Sai Institute of Higher Learning.

At the outset I thank my Project Supervisor Dr. GowriShankar Ramadurai, Head, Department of Physics, Sri Sathya Sai Institute of Higher Learning for his constant guidance and support all through my project.

I owe a deep sense of gratitude to my Industry Mentor Mr. Vikram Seshasai, Principal Engineer, Product Verification Team, Tejas Networks Pvt. Ltd for sparing his valuable time in guiding me through this exciting field of Virtualization, despite of his tight schedule. The work was done in cooperation with Tejas Networks. It helped me in getting the industrial work exposure. Its a genuine pleasure to express my deep sense of thanks and gratitude to my mentor Prof. S Siva Sankara Sai, Dean of Sciences, Sri Sathya Sai Institute of Higher Learning for his support and interest in the domain of Networking.

I wish to express my thankfulness to Mr. Avinash B V from Tejas Networks for providing his valuable time support and inputs and motivating me. Inspite of his busy schedule he has taken out his time and supported me.

I express my deepest sense of gratitude to my Parents for their love and ever indebted to them for their encouragement all through.

-Pranav Godway

Abstract

Rising traffic congestion is an inescapable condition in large and growing metropolitan cities leading to many problems for an individual or group. Movement of emergency vehicles are hampered due to traffic congestiion costing lives. This work aims to provide a solution to the Road Traffic Management System in order to avoid congestion and to provide services in an emergency situation. This work uses the emerging paradigm in the network called Software Defined Networks (SDN), also known as Network Automation and Network Programability, to solve traffic congestion. In this work, we have used a real - life model, borrowed from the state of Orissa and the city of Bhubaneshwar, to develop an application for managining the traffic congestion. This application is built using the Software Defined Internet of Things (SDIoT) architecture. Our work highlights the need for SDN in IoT.

Contents

List of Figures

List of Tables

Abbreviations

SDN	Software-Defined Networks
SDIoT	Software Defined Internet of Things
ACL	Access Control List
RTMS	Road Traffic Management System
API	Application Program Interface
NBI	Northbound Interface
SBI	Southbound Interface
OVS	Open vSwitch
NOS	Network Operating System
ONOS	Open Networking Operating System
GUI	Graphical User Interface
OVSDB	Open vSwitch Database
SNMP	Simple Network Management Protocol
I2RS	Interface to the Routing System
BFD	Bidirectional Forwarding Detection
CLI	Command Line Interface
IoT	Internet of Things
V2V	Vehicle to Vehicle
V2P	Vehicle to Pedestrian
V2I	Vehicle to Infrastructure
V2N	Vehicle to Network
RSU	Road Side Unit
RESTful	Representational State Transfer

Chapter 1

Introduction

" Networking is not about just connecting people. Its about connecting people with people, people with ideas, and people with opportunities." – Michele Jennae

1.1 Background

As time has moved on there is a significant change in the network that has taken place from a conventional IP system to the current emerging networking paradigm called SDN. Before discussing the new networking paradigm let us have a look at some of the limitations of the traditional network.

- Conventional IP networks are complex and difficult to manage
- The system administrator needs to configure each device separately by using vendor dependent commands or using low-level language.
- Currently, there is no automatic reconfiguration and reaction system in current IP networks.
- The control plane (that chooses how to deal with system traffic) and the information plane (that advances traffic as indicated by the choices made by the control plane) are packaged inside the networking devices, decreasing adaptability and obstruct innovations and development of the networking framework [1].

Demands on networks are growing rapidly both in volume and variety. Adding more switches and transmission capacity, involving multiple vendor equipment, is difficult because of the complex, static nature of the network. To respond for demands such as differing levels of QoS, high and fluctuating traffic volumes, and security requirements, networking technology has grown more complex and difficult to manage. The increasing number of devices, the growing volume and velocity of traffic, big data and cloud computing are some of the problems that the traditional internet is facing deal with. Software Defined Networking (SDN) provides a solution to these challenges by making the internet flexible and programmable.

1.2 Motivation

The objectives of SDN is to incorporate the network innovations faster and to simplify and automate management of the complex network. SDN, being a flexible and programmable network, provides an effective solution to many problems faced by the current internet architecture. It makes it easier to test and deploy new protocols, it has increased the rate of innovation in the networks, and it decreases the barrier for a new technology to be deployed at large scale. These has led to some major players like Google and Microsoft to switch to SDN for their cloud and data center operations. This has led to an interest in the community of vendors and Internet Service Provodere ISPs. Vendors like Cisco, Ericsson, Nokia are actively involved with the Open Networking Foundation (ONF) for research and development of SDN.

Software-defined networking has steamed as a new network architecture that increases programmability and centralizes the intelligence of the network, breaking the control plane out of the switch itself and delegating this control to a central equipment increasing the flexibility of the network. SDN aims at changing the manner in which networks are structured and managed over the past years.

The following points will make it clear that why we need SDN

1. **Virtualization**: Use network resource without worrying about where it is physically located, how much it is, how it is organized, etc.

2. **Orchestration**: Should be able to control and manage thousands of devices with one command.
3. **Programmable**: Should be able to change behavior on the fly.
4. **Dynamic Scaling**: Should be able to change size, quantity.
5. **Visibility**: Monitor resources, connectivity.

1.3 Problem Definition

Rising traffic congestion is an inescapable condition in large and growing metropolitan cities in the country. Which lead to many problems for an individual or to a group. Many have lost their lives due to the traffic congestion. This works aims at providing an efficient traffic congestion management and to ensure smooth flow of emergency/VIP vehicles.

We make use of the emerging paradigm in the network called as Software Defined Networks (SDN) which is also known as Network Automation and Network Programmability to solve the problem of the Traffic Congestion. This work also brings out the need for SDN in the Internet of Things called Software Defined Internet of things (SDIoT) which will be discussed in the upcoming chapters and proposes the solution using SDN as a framework. We aim to simulate the real-life scenario and provide the necessary solutions for the easy management of traffic congestion so as to avoid the inconvenience faced by the citizen. We make use of ACL (Access Control List) in order to avoid congestion which will be disussed in detail in the upcoming chapters.

The important tools used in this work are Open Networking Operating System also called ONOS which is the SDN controller, Mininet Network Emulator and Wireshark. These are opensource software that are easily available. These will be discussed briefly in the upcoming chapters.

1.4 Thesis Outline

The Thesis is structured as follows.

In **chapter 1**, we start with limitations of the traditional network then discuss how SDN will overcome this limitations. We as well discuss why we need SDN. Thereafter, we defined the problem statement which is related to Road Traffic Management System (RTMS).

In **chapter 2** we will discuss State of the Art i.e SDN, its architecture, the switches in the SDN for the Data plane, limiting to Open Vswitch (OVS). Then we will discuss SDN controller for the Control Plane again specific to ONOS. We then enumerate the prtocols for communication in SDN and conclude the chapter by explaining the Mininet Network emulator for the creation of Network topologies.

In **chapter 3** we give the need for SDN in Internet of things(IoT), then discuss briefly about the SDIoT Architecture. Next we compare the RTMS architecture with and without controller.

Chapter 4 we explain the model that we have developed in order to simulate and overcome traffic congestion in real-life road scenario.

And finally in **chapter 5** we give the results, conclusion and future possible work.

Chapter 2

Emerging Networking Paradigm SDN

“ Think of it as a general language or an instruction set that lets me write a control program for the network rather than having to rewrite all of code on each individual router”. – Scott Shenker

2.1 Software Defined Networks

One of the main features that Software Defined Networks also known as Network Programmability and Network Automation mainly focuses on, is the separation of data plane and control plane. In SDN, the control plane is centralized, controls a distributed data plane and can be implemented completely in software and installed on hardware. Therefore, a SDN is characterized as a programmable network. It refers to networks in which the behavior of network devices is handled by software. Control plane refers to the logic of controlling and forwarding behavior. Data plane (also known as forwarding or user plane) on the other hand refers to the network part that forward user traffic. Forwarding is based on rules as set by the control plane.

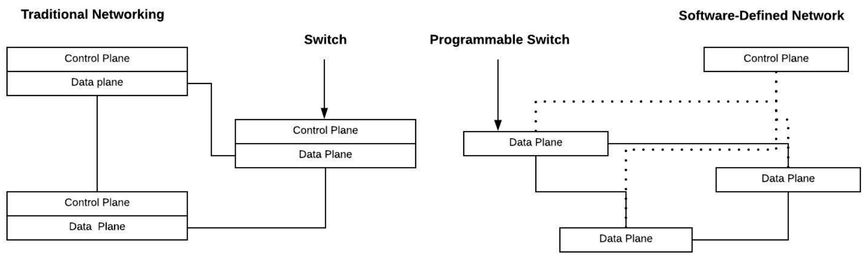


FIGURE 2.1: Comparison between Traditional Network and SDN
Redrawn: Image source:https://www.bing.com/images/search/slidesharecdn.com

SDN is a model which is based on the idea of moving from the traditional fully distributed model to a more centralized approach. Before we begin to discuss the architecture as shown in figure 2.2 let us have a look at the few important terminology involved in SDN.

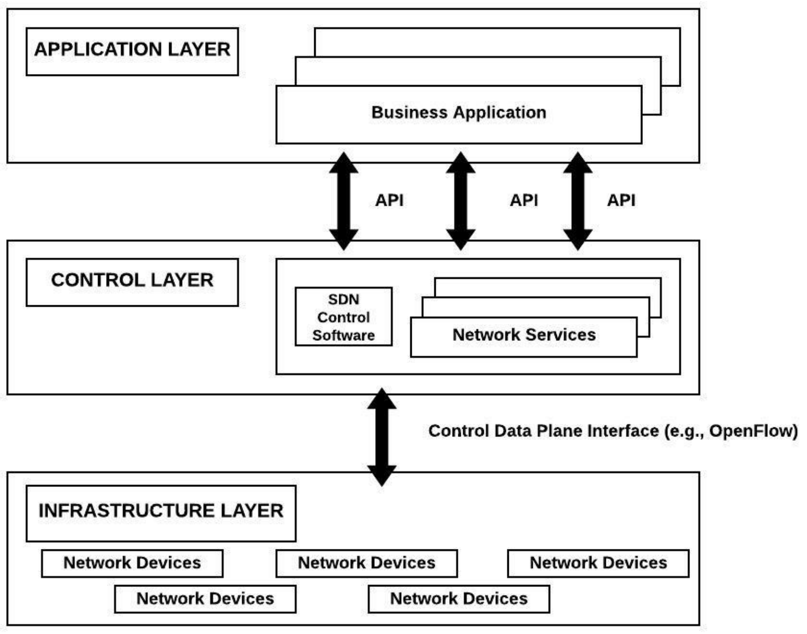


FIGURE 2.2: SDN Architecture
Redrawn: Image source:https://www.sdxcentral.com/sdn/definitions/inside-sdn

2.1.1 Important Terminology involved in SDN

- **Application**: Also known as App, an SDN application is a software program designed to perform a task in a software-defined networking (SDN) environment.

- **API**: Application programming Interface, allows applications to communicate with one another.
- **Forwarding Plane**: Responsible for forwarding the traffic in the network.
- **Control Plane**: Instructs network devices with respect to how to process and forward packets.
- **Operational Plane**: The collection of resources responsible for managing the overall operation of individual network devices [2].

2.1.2 SDN Architecture

SDN Architecture is a layered architecture as shown in Figure 2.2, at the bottom most layer is the Infrastructure layer where the networking devices, like switches, routers etc. The important actions performed by the plane includes forwarding and dropping the packets, alter the packet header and so on. This layer is responsible for forwarding the traffic based on the instructions received from the control plane. The network intelligence is expelled from the information plane devices to a logically-centrallized control system. Network devices can be implemented in hardware or software and can be either physical or virtual.

Next layer is the Control layer which is the brain of the SDN, responsible for making decisions on how packets should be forwarded by one or more network devices and forwarding such decisions down to the network devices for execution. This is where the SDN Controller resides and is entity that implements the control plane functionalities. Controller gathers information about the network from the hardware devices and revert back to the SDN Applications with an abstract view of the network, including statistics and events about what is happening. The important control-plane functionalities includes:

- Topology discovery and maintenance
- Packet route selection
- Path failover mechanism

The SDN controller is responsible for overseeing and controlling the whole network through the Network Operating System from a central point with a global view of all the network resources. The fundamental work of Networking Operating System is to provide abstraction and a common API to developers.

Application plane is where applications and services that define network behavior reside. The application can be a SDN application, networking management or business application. SDN Applications are programs that convey behaviors and needed resources with the SDN Controller by means of application programming interfaces (APIs). Applications can assemble a preoccupied perspective on the system by gathering data from the controller for decision-making purposes. In SDN, the control plane gives a dynamic view and resource information of the whole network components to SDN applications. It is not necessary that all SDN applications will be completely new. A lot of them replicate or improve applications that are currently running on routers and switches (control and data plane).

The next question that comes to mind is how the layers communicate with each other. There are several standards protocol and API's that makes communication possible between the layers. The communication between Infrastructure and the Control layer takes plcae via what is called Northbound Interface (NBI) and the communication between Control plane and Application plane takes place via Southbound Interface (SBI). Let us now discuss briefly what is NBI and SBI.

- **Northbound Interface**: Northbound communication refers to the communication between the application layer and the control layer. A northbound API is used to implement and develop vendor independent applications for network management and monitoring. Another advantage of northbound APIs is that they are easy modifiable using high level languages like Python, Java, C++ etc. The northbound interface is defined completely in software, while controller-switch interactions must be enabled by the hardware implementation. Currently there is no common northbound interface, it is still an open issue. Existing controllers such as floodlight, NOX, Opendaylight have there own northbound API. Current controller offer a variety of northbound API's such as ad-hoc API's, RESTful API's.

- **Southbound Interface**: They provide a common interface for the upper layers, while allowing a control platform to use different southbound APIs e.g., OpenFlow, OVSDB, ForCES and protocol plugins to manage existing or new physical or virtual devices e.g., SNMP, BGP, NetConf. Many controllers support only openFlow as a southbound API. Still, a few of them, such as OpenDaylight, HP-VAN SDN controller, offer a wider range of Southbound API's and/or protocol plugins. OpenFlow, which was developed by the ONF, is the first and probably most well-known southbound interface which we will discuss in detail in the upcoming section. It is an industry standard that defines the way the SDN controller should interact with the forwarding plane to adjust the network, so as to adapt to changing business requirements.

- **Eastbound and Westbound Interface**: These are the special case of interfaces required by distributed controllers. At present every controller implements its own east/westbound API. The function of these interfaces incorporates import/export data among controllers and observing/notification abilities e.g., check if a controller is up or advise a take over on a lot of forwarding devices [1].

2.2 Software Switch for Data plane

As mentioned earlier, the data plane of SDN is made up of simple forwarding devices, these can be either implemented in hardware or software. In this section, we are going to discuss mainly the software switches with more attention to Open vSwitch. The real benefit of a software switch is that they provide a platform for rapid testing. Moreover, their predominantly open source nature has increased the rate of innovation in networking. Many industry-leading partners and vendors have jumped into soft-switch an abbreviation for software switches. Some of the notable ones being VMWare, Cisco, Big Switch Networks. Various research groups have also come up with their open source projects for soft-switches like the CPqD OfSoftSwitch. For a long time, the software switches have struggled to provide as good a performance as their hardware counterparts.

Open vSwitch is one of the most popular software switches and is backed by the Linux Foundation. It is considered by many as the de facto software switch for SDN. OVS was created by Nicira which was later acquired by VMWare. It is feature rich and supports all the major protocols and technologies like VLAN tagging, NetFlow, Port Mirroring, etc. OVS uses OpenFlow protocol which we will be discussing in the upcoming section, for communication with the controller and uses the Open vSwitch Database (OVSDB) for configuration management.

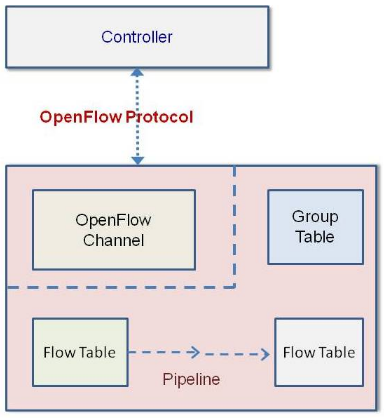


FIGURE 2.3: OpenFlow switch components
Redrawn:Image source:OpenFlow Switch Specification, version 1.3.2,April 25,2013

Figure 2.3 Shows segments of an OpenFlow switch, consists of stream of flow tables and a group table, which perform packet lookups and forwarding, and an OpenFlow channel to an outside controller. The switch communicates with the controller and the controller deals with the switch by means of OpenFlow protocol which we will discuss in the next section.

The switches refer to the flow table to match an incoming flow and then apply the specified action for the given flow type. If a matching entry is found, the instructions associated with the specific flow entry are executed. If no match is found in a flow table, then the following actions take place

- Packet may be sent to the controller over the OpenFlow channel
- The packet may be dropped
- May proceed to the next flow table

Let's have a quick look through a flowchart to see how a packet matches. Figure 2.4 shows the flowchart of a packet flow. On receipt of a packet, the switch begins by

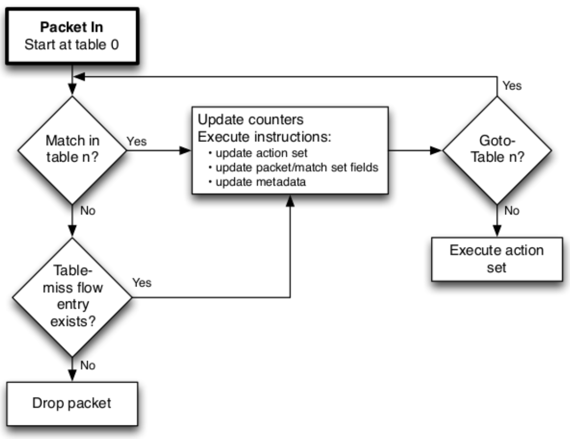
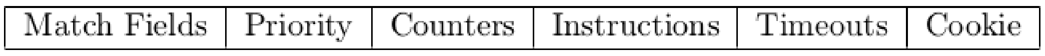


FIGURE 2.4: Flowchart of a packet flow through an OpenFlow switch
Redrawn:Image source:OpenFlow Switch Specification, version 1.3.2,April 25,2013

performing a table lookup within the first flow table, and based on pipeline processing, may perform table lookups in other flow tables. A pipeline is fundamentally a sequence of flow tables. Packet match fields are extracted from the packet. Packet match fields used for table lookups depend on the packet type and typically include different packet header fields, such as Ethernet source address or IPv4 destination address. In expansion to packet headers, matches can too be performed against the ingress port.

Figure 2.5 shows the flow entries involved in the flow table. Each flow table will contain the following entries.



Match Fields	Priority	Counters	Instructions	Timeouts	Cookie

FIGURE 2.5: Components of a flow entry in a flow table.
Redrawn:Image source:OpenFlow Switch Specification, version 1.3.2,April 25,2013

When a new flow arrives at the switch, the switch matches it with the available match fields. If it matches any of the match fields available in the flow table, the corresponding instructions are applied to the flow according to the defined priority and corresponding counters are incremented. If the new flow doesnt match any existing match field, the request is sent to the controller and controller then installs a new entry in the table [3].

2.3 Protocol for Communication

To turn the concept of SDN into practical implementation, two requirements must be met. First, there must be a common logocal architecture in all switches, routers, and other network devices to be managed by an SDN Controller. This logical architecture may be implemented in different ways on a different vendor equipment and in different types of network devices, so long as the controller sees a uniform logical switch function. Second, a standard, secure protocol is needed between the SDN controller and the network devices.

Both of these requirements are addressed by ***OpenFlow***, which is a both a protocol between SDN controller and network devices, as well as a specification of the logical structure of the network switch function. It was originally developed at Stanford University and currently under active standars development through Open Networking Foundation. ONF is a consortium of software providers, content delivery network, and networking equipment vendors whose purpose is to promote Software-Defined Netwoking. Table 2.1 [1] shows various Hardware and Software switches OpenFlow enable devices.

Table 2.1: OpenFlow enabled Hardware and Software Devices

Group	Product	Type	Maker/Developer	Version	Short description
Hardware	Arista 7150 Series	switch	Arista Network	v1.0	Data centres hybrid Ethernet/OpenFlow switches
	NoviSwitch 1248	switch	NoviFlow	v1.3	High-Performance OpenFlow switch
	Pica8 3920	switch	Pica8	v1.0	Hybrid Ethernet/OpenFlow switches
	RackSwitch G8264	switch	IBM	v1.0	Data centre switches supporting OpenFlow
Software	Open vSwitch	switch	Open Community	v1.0-1.3	Switch platform designed for virtualized server environment.
	ofsoftswitch13	switch	Ericsson, CPqD	v1.3	OF 1.3 compatible user-space software switch implementation.
	OpenFlow Reference	switch	Stanford	v1.0	OF Switching capability to a Linux PC with multiple NICs
	Switch Light	switch	Big Swiych	v1.0	Thin switching software platform for physical/virtual switches

OpenFlow isn't the only available protocol for SDN. There are other protocols such as Interface to the Routing System (I2RS), Simple Network Management Protocol (SNMP), Bidirectional Forwarding Detection (BFD). But OpenFlow is the foremost broadly acknowledged and deployed open southbound standard for SDN.

2.4 SDN Controller for Control Plane

The control plane of an SDN constitutes the brain of the network and all the decision regarding routing and policies etc. are taken at the control plane of an SDN. It is the middle man between the applications connected to the northbound interface and the network devices connected to the southbound interface. Control plane consists of two main components;

- **Application and**
- **Network Operating System a.k.a the controller.**

There may be more than one controller in a network, and they must coordinate to create a consistent network state. This is usually done by assigning one controller as the master controller and others as backup. Even in the presence of multiple controllers, the control should logically appear to be centralized. The NBI and the SBI are the two important abstractions provided by the controllers. The applications sitting above the controller need to bother about the internal architecture and functioning of the layers below it and the devices below the controller on the SBI need not bother about the layers above it. R. Masoudi et al.[7] categorizes the area of concerns for controller design into following categories:

- **State Consistency** : A real life network would be too large to be handled by a single controller. It will also be too big a risk as it would be a single point of failure. So, usually, a network has a multiple controller managing a group of network devices. There is redundancy, i.e., a switch might be connected to multiple controller in master and backup mode for fault tolerance.
- **Scalibility**: Scalability of the SDN controllers become a critical problem with the increase in the popularity of SDN. One way to achieve scalability is to design the controller with a distributed architecture. ONOS is one such distributed network operating system and hence has been the most widely accepted production, ready controller which we will be discussing shortly.
- **High Availibility and Low Latency** : Tolerating failure and recovering from link and switches faults are some of the important consideration of controller design. A distributed system must ensure high availibility even in the presence of faults.

Since the inception of SDN, there has been increasing interest in the research and industrial community towards creating an SDN controller that can be deployed in real world networks. The following Table 2.2 [1] shows a list of the most notables controllers, the language it is implemented in and the developers and various others parameters.

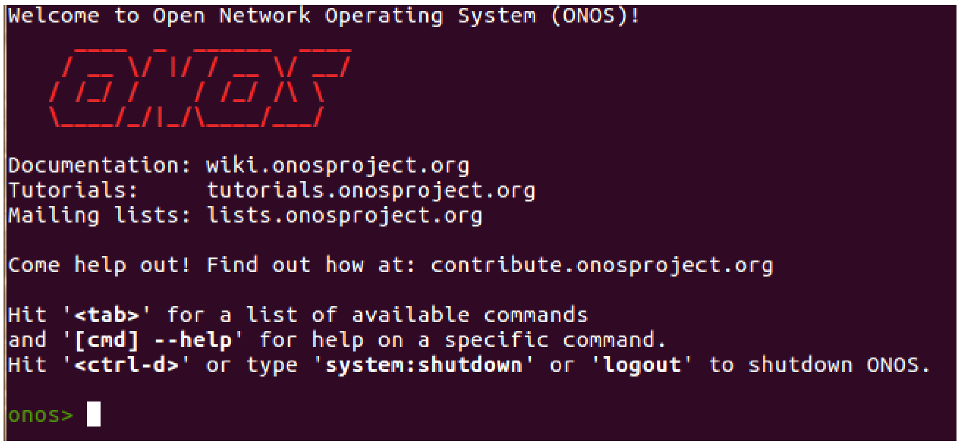
TABLE 2.2: Controllers Classification

Name	Architecture	Northbound API	Consistency	Faults	License	Prog. language	Version
Opendaylight	distributed	REST, RESTCONF	weak	no	EPL v1.0	Java	v1.{0,3}
ONOS	distributed	RESTful API	weak, strong	yes		Java	v1.0{0,4}
NOX	centralized	ad-hoc API	no	no	GPLv3	C++	v1.0
POX	centralized	ad-hoc API	no	no	GPLv3	Python	v1.0
HP VAN SDN	distributed	RESTful API	weak	yes		C++	v1.0

In our work, we have chosen Open Networking Operating System (ONOS) as the controller. ONOS mission is to produce the Open Source Network Operating System that will enable service providers to build real Software Defined Networks . ONOS targets support of multiple protocols (Openflow, NetConf, etc.), at the southbound interface to communicate with diverse devices, and expose APIs at the northbound interface to accommodate the needs of service provider use cases and application developers. It provides the control plane for a Software-Defined Network (SDN).

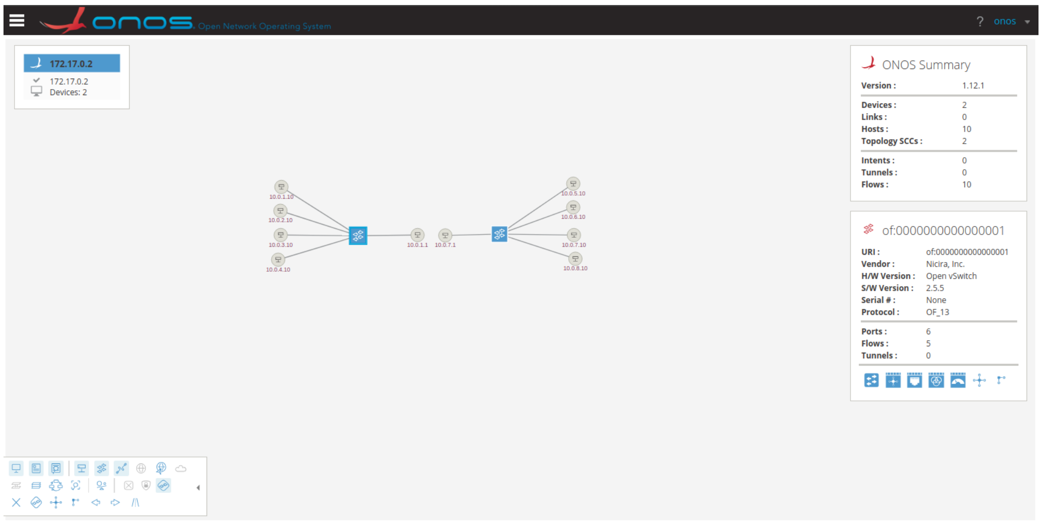
ONOS is a scalable and distributed controller platform that targets service providers networks and service provider's requirements, such as policy-driven network programmability and being operator-friendly. ONOS supports multiple protocols at the southbound interface to communicate with the devices. ONOS project is an open source community hosted by The Linux Foundation. The aim of the project is to create a software-defined networking operating system for communications service providers that is designed for scalability, high performance and high availability which makes it perfect choice for building next-generation SDN solution [4].

ONOS provides the network graph, and the view of the entire network, as the northbound abstraction. The ONOS platform is designed to support various application categories such as control, configuration and management applications. Figure 2.6 and 2.7 shows the ONOS CLI and GUI.



```
Welcome to Open Network Operating System (ONOS)!
     ____  _  ______  ____
    / __ \/ |/ / __ \/ __/
   / /_/ /    / /_/ /\ \
   \____/_/|_/\____/___/

Documentation: wiki.onosproject.org
Tutorials:     tutorials.onosproject.org
Mailing lists: lists.onosproject.org

Come help out! Find out how at: contribute.onosproject.org

Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.
Hit '<ctrl-d>' or type 'system:shutdown' or 'logout' to shutdown ONOS.

onos>
```

Figure 2.6: ONOS CLI
Screenshot from PC



172.17.0.2
172.17.0.2
Devices: 2

ONOS Summary

```
Version :         1.12.1
Devices :         2
Links :           0
Hosts :           10
Topology SCCs :   2
Intents :         0
Tunnels :         0
Flows :           10
```

of:0000000000000001

```
URI :           of:0000000000000001
Vendor :        Nicira, Inc.
H/W Version :   Open vSwitch
S/W Version :   2.5.5
Serial # :      None
Protocol :      OF_13
Ports :         6
Flows :         5
Tunnels :       0
```

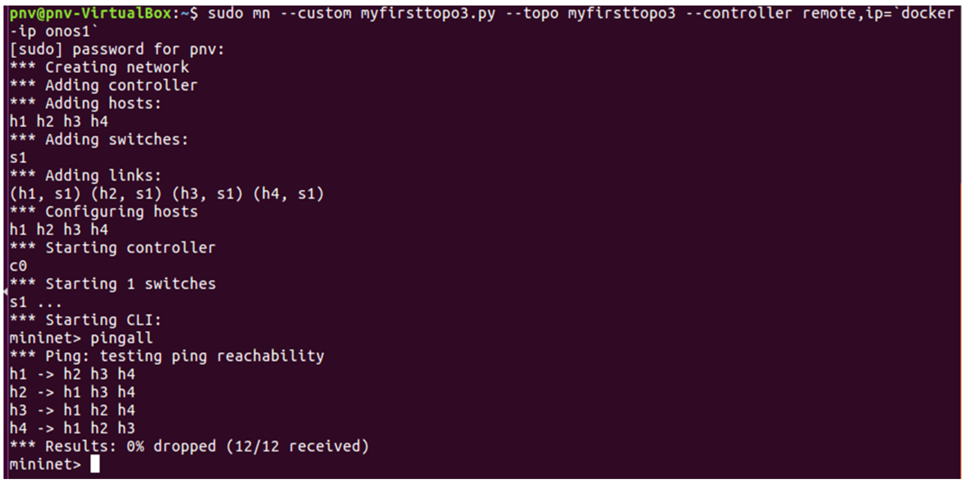
Figure 2.7: ONOS Web Interface
Image source: Screenshot from PC

ONOS suppports multiple southbound protocols like OpenFlow, NetConf, etc., for communication with a variety of network devices. The ONOS kernel and core services, as well as ONOS applications, are composed in Java as bundles that are stacked into the Karaf OSGi container. OSGi is a component framework for Java that permits modules to be introduced and run dynamically in a single JVM. Since ONOS runs in the JVM, it can run on several underlying OS platforms such as Ubuntu or OS X [5].

2.5 Mininet Network Emulator

Mininet is open source software that is used to simulate a software defined network (SDN) and its compatible controllers, switches and hosts. Mininet switches support OpenFlow for highly flexible custom routing and SDN. Mininet supports research improvement, learning, prototyping, testing, investigating, and any other errands that seem advantage from having a total experimental network on a portable workstation or other PC. Some of the main features of mininet are:

- Provides a basic network testbed for creating OpenFlow applications.
- Empowers complex topology testing, without the ought to wire up a physical network.
- It provides built in topology such as single, linear and tree other than these topologies we need to build a Python script [6].



```
pnv@pnv-VirtualBox:~$ sudo mn --custom myfirsttopo3.py --topo myfirsttopo3 --controller remote,ip=`docker
-ip onos1`
[sudo] password for pnv:
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s1)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
mininet> 
```

Figure 2.8: Mininet CLI
Image source:Screenshot from PC

Chapter 3

Software Defined Internet of Things (SDIoT)

" Its going to be interesting to watch, and to see whether networking companies are going to be software companies, not everybody finds BGP and OSPF as exciting as I do.". – David Ward, Cisco

3.1 Introduction to SDIoT

With the number of devices connected to the Internet of Things (IoT) projected to expand to somewhere between 20 and 46 billion by 2020, our future will connect nearly everything, from traditional communication tools, such as laptops and smartphones, to home appliances, such as refrigerators and garage doors, and from industrial systems to actual people.

Although several solutions have been proposed and implemented to deal with a steady increase in data consumption and number of devices (e.g., the introduction of IPv6), they were not designed with billions of new users/devices in mind, who would join the network in a short period of time. This projected growth means that current wireless and mobile networks should evolve to become more intelligent, efficient, secure and, most importantly, extremely scalable to deal with a torrent of data communications that are extremely diverse in nature. Software defined networking (SDN) is the most prominent technologies to serve as key enablers for the IoT networks of the near future [7].

3.2 Why SDN in IoT?

IoT applications are drastically different from the legacy ones. IoT intends to connect billions of devices. Apart from the dimensionality challenge, complex operational issues also exist for IoT. These issues can be resolved by introducing a high level of automation in the delivery and operations of IoT applications. This is where SDN can facilitate the IoT domain. Following are some challenges or issues in IoT and how SDN programmability can help in these:

- **Massive device Connectivity**: IoT will comprise of billions of multi-vendor devices, which might be deployed in a very uncoordinated manner. Adding routing/forwarding information pertaining to these devices wil require the automatic discovery of these devices and dynamic route computation mechanism across the network. SDN controller can be used to interface with switches in the forwarding plane to set the traffic flow across the network.
- **Rapid changes in the Network**: IoT devices are constrained in terms of energy and CPU. They might often get pruned from the network due to low battery or CPU overloading. Thet also operate on various wireless technology which might have considerable failure rates. IoT networking infrastructure might have to handle rapid changes- thus needing change in the routing or flow information on the network elements. SDN can handle such scenarios by dynamic topology maintenance.
- **Low Power Endpoints**: IoT endpoints have a very low computing power and need to manage battery drain issues. Hence, they can not implement complex routing or network control protocols.
- **Resource utilization**: The network performance can be reduced under or over utilization, which in turn minimizes the network utility. Therefore, efficient user request mapping is required to enhance resource use to maximize network utility. In SDN, flow - rule based forwarding helps to improve the use of network resources.

Requests from multiple users can therefore be transmitted via the desired path according to the SDN controller's flow rules [8].

3.3 SDN based IoT Architecture

Figure 3.1 shows SDN bases IoT architecture [9]. This is a layered architecture which consists of four layers. We will discuss each layer briefly.

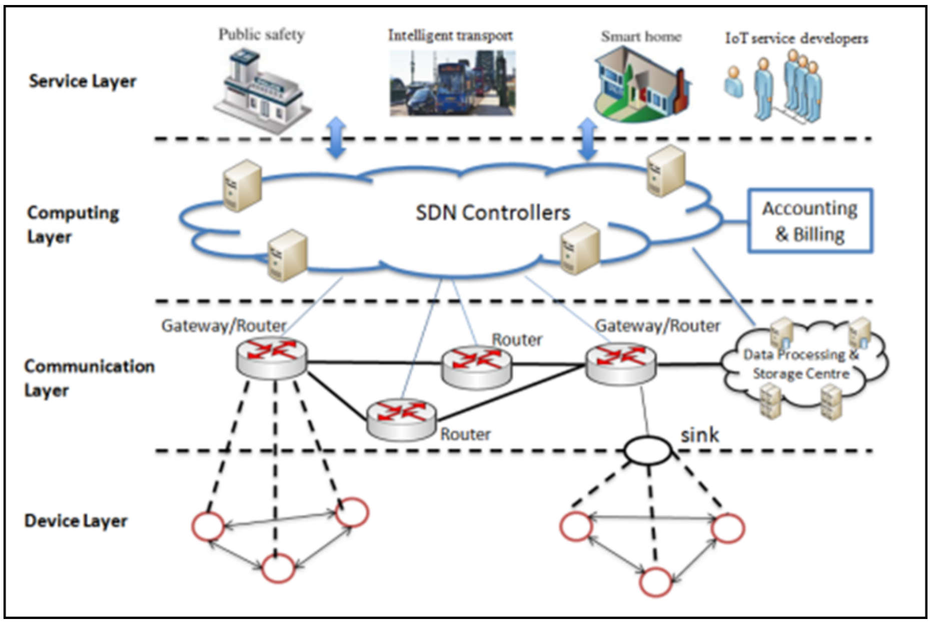


FIGURE 3.1: A layered IoT architecture based on SDN
Image source: A SDN-based Architecture for Horizontal Internet of Things Services, Yuhong Li et al.

The first layer is the device layer, which contains sensors gathering a lot of information in various formats and possibly for various IoT application domains. A few devices can also act as actuators receiving commands from the network and performing a task.

The next layer is the communication layer consisting of SDN gateways and routers which, under the control of the SDN controller, can forward data. The computing layer includes the SDN controller and the accounting and billing mechanism. They control the sending of information as per the necessities of application. By programming the SDN controllers, service developers and operators build IoT services on the service layer.

Let's look briefly at the functions of the architecture's major components.

1. **SDN Controller**: The controller not only does data forwarding but even performs processing of data. The SDN controller have the following functions:
    - Device management, such as gateway / router configuration, network resources, policies and rules for data processing in the relevant devices, etc.
    - Topology management, such as routing calculation and updating topology.
    - Security management, such as recognizing conflicts and confirming service access.
2. **Gateways/Routers**: Gateways and routers are responsible for transmitting data in the network under the guidance of SDN controllers, gateways can either store or cache local data or process data.
3. **Data Processing and Storage Center**: The data obtained in the networks from IoT devices and sinks can be stored in this module selectively as instructed by the controller.
4. **Sinks**: They are responsible for aggregating and caching the data obtained from IoT devices. They can not be programmed by SDN controller. They can perform task such as eliminating some redundant data obtained fom the sensors.
5. **Accounting and billing Center**: It keeps the record of the resources consumed in the network for e.g., network bandwidth, computing and storage resources [9].

Based on the SDIoT Architecture we have taken up a problem in the area of the Road Traffic Management System, which is aimed at to solve congestion. The architecture and details of the Road Traffic Management System will be discussed in detail in the next chapter.

Chapter 4

Road Traffic Mangement System Model

"But one thing is for sure: things are changing fast, and it appears software defined networking, in one form or another, is the way forward.". –Brad Hedlund

This chapter focuses on the model developed in order to avoid congestion. This model is based on the real - life scenario that we will discuss shortly in the upcoming section. We have made certain assumptions in this model that will emulate the behavior equivalent to networking.

4.1 Road Traffic Management System Architecture

Based on SDIoT architecture, we addressed the Road Traffic Management System (RTMS) problem, which aims to solve the congestion. In this section, we will discuss the current vehicle communication architecture, what are the limitations and how SDN can overcome these problems. The Figure 3.2 shows the traditional architecture, which is the layered architecture. We will discuss the architecture in brief and what are the drawbacks of the architecture in the next section.

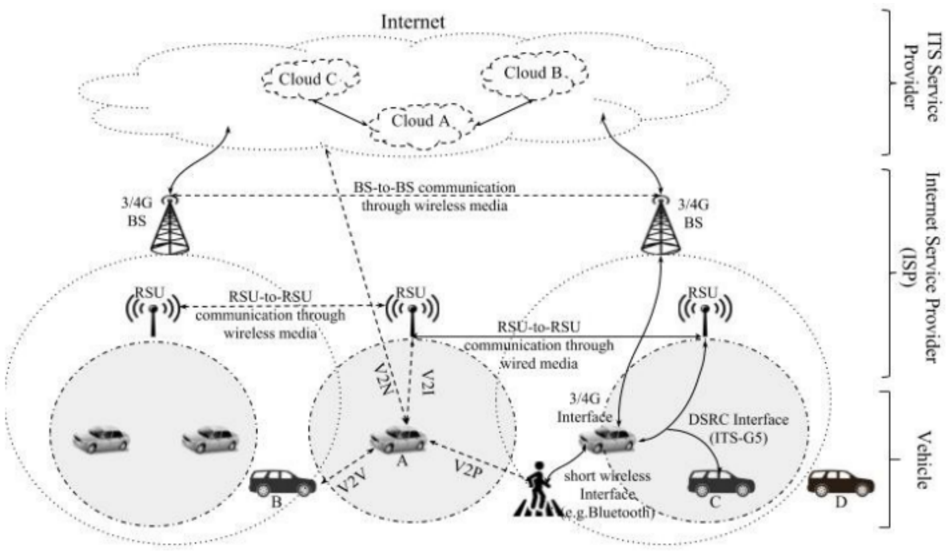

Internet
Cloud C
Cloud B
Cloud A
ITS Service Provider
3/4G BS
BS-to-BS communication through wireless media
3/4G BS
Internet Service Provider (ISP)
RSU
RSU-to-RSU communication through wireless media
RSU
RSU
RSU-to-RSU communication through wired media
V2N
V2I
3/4G Interface
DSRC Interface (ITS-G5)
A
V2V
V2P
short wireless Interface (e.g.Bluetooth)
B
C
D
Vehicle

FIGURE 4.1: Traditioinal architecture for transport system
Image source:An SDN hybrid architecture for vehicular networks:Application to Intelligent Transport System, Soufian et al.

Architecture consists of three main parts as shown in Figure 4.1. The vehicle, the network infrastructure managed by an Internet Service Provider, and the cloud platform controlled by an ITS service provider. There are different types of communication involved to name a few vehicle-to-vehicle (V2V), vehicle-to-Pedestrian (V2P), Vehicle-to-Infrastructure (V2I), Vehicle-to-Network (V2N) [10].

The density and the speed of the vehicles are the major factors affecting the quality of vehicular communication. In high density networks, vehicle must efficiently share available network resources in order to avoid congestion problems, which is a challenging task. Besides, the high speed of the vehicles complicates the maintenance of the communication between the nodes. This situation becomes more complicated when the vehicles move in opposite direction.

To overcome the limitations involved in this architecture, we have looked into the architecture where the SDN controller was able to overcome the drawbacks involved in the architecture. We will discuss the architecture with controller briefly in the next section.

4.2 Road Traffic Management System Architecture with Controller

This architecture overcomes the limitations of traditional architecture. Figure 3.3 shows the architecture with the controller.

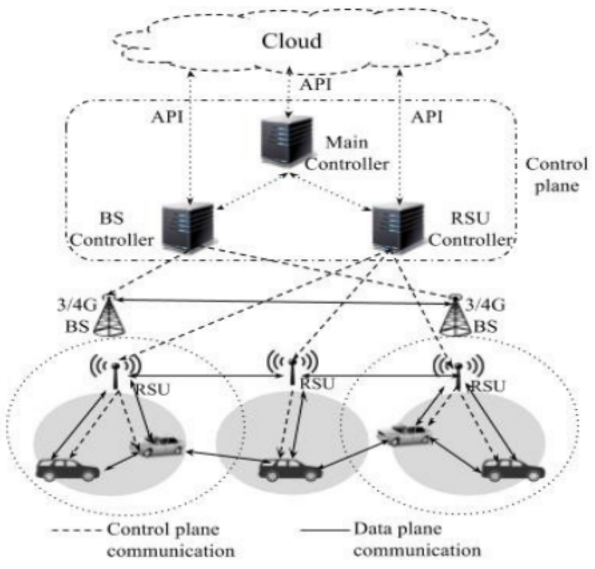


FIGURE 4.2: Architecture with controller
Image source:An SDN hybrid architecture for vehicular networks, Soufian, et al.

In this architecture multiple controllers were involved, one to manage a cellular network, another to manage the Road Side Unit (RSU) based network and a last one to coordinate between the different controller. The main controller gives the global view of architecture using the information sent by the controller of each network coupled with the data present in the cloud. This architecture gives the vehicle status information which can be position, direction and speed which can be used to predict the number of vehicles that is present in a given region at a given time [10]. The advantages this architecture could give us to list a few of them

- It gives a vision of the state of all the network involved in the architecture.
- the ability to jointly control these networks.
- knowledge of the environment in which vehicles evolve.

4.3 Real-Life Scenario

We have taken up a real-life scenario from the state of Orissa and the city of Bhubaneshwar. The source and destination are Orissa Vidhan Soudha the State Assembly and Biju Patnaik International Airport. This is a straight road with three junction. The Figure 4.1 shows the real map of the source and destination.

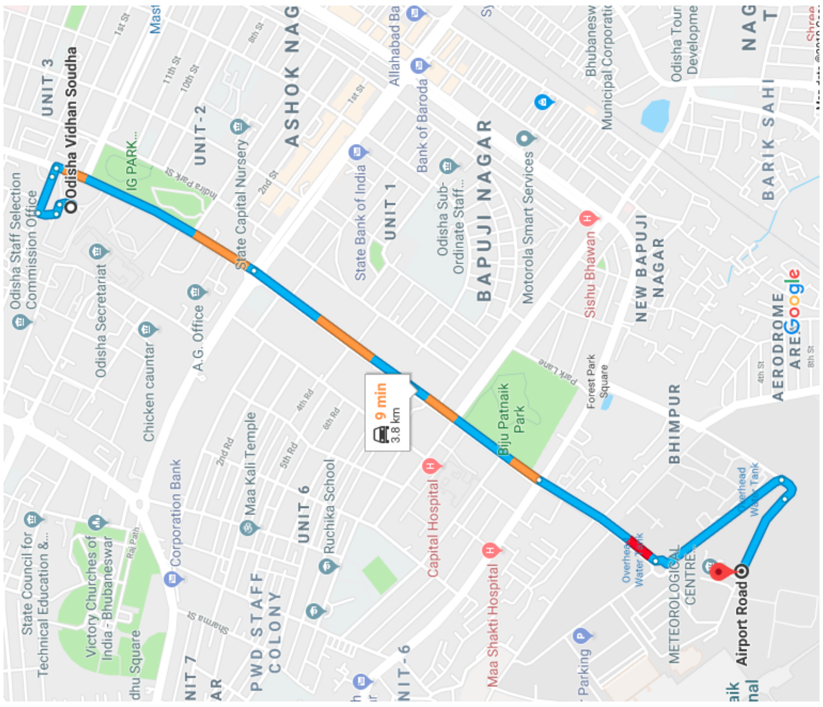


FIGURE 4.3: Real Map
Image source: Google Map

The Figure 4.2 shows the Real road scenario. This is a five lane road with three junctions.

As seen in Figure 4.2, it's a five-lane road and according to our model, we are going to keep first lane for the VIP or emergency transit and rest of the lanes for the normal transit. The first lane is specifically reserved for the VIP or the emergency transit so as to avoid the inconvenience faced by the citizen.

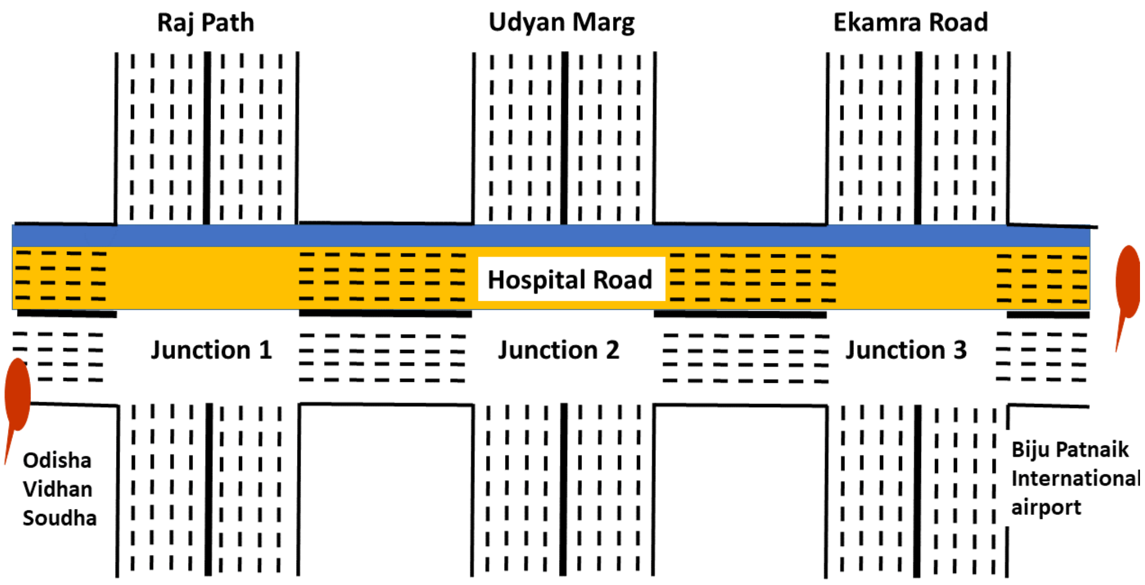


FIGURE 4.4: Networking model at layer three

4.4 Networking Model

Figure 4.5 shows the Networking Model.

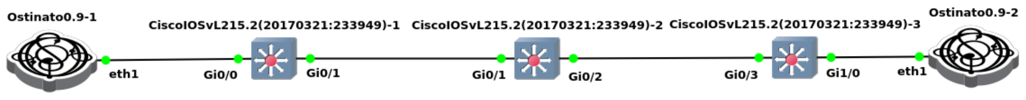

Ostinato0.9-1
CiscoIOSvL215.2(20170321:233949)-1
CiscoIOSvL215.2(20170321:233949)-2
CiscoIOSvL215.2(20170321:233949)-3
Ostinato0.9-2
eth1
Gi0/0
Gi0/1
Gi0/1
Gi0/2
Gi0/3
Gi1/0
eth1

FIGURE 4.5: Networking Model

This model is based on **Quality of Services** features at layer two. As mentioned earlier we are emulating real-life road scenario as networking which is as follows:

- Generators are represented as locality.
- Packets are represented as Vehicles.
- Switches are reprsented as Junctions.
- Lanes are represented as Queues.

The features which we have used in our model are Services, Policiers and Queues to solve the problem of traffic congestion. Just to give a brief overview of the features in brief which are as follows:

- **Ethernet Virtual Connection**: An EVC is a conceptual service pipe within the service provider network. A bridge domain is a local broadcast domain that is VLAN-ID-agnostic. An Ethernet flow point (EFP) service instance is a logical interface that connects a bridge domain to a physical port or to an EtherChannel group in a switch.
- **Policers**: Networks police traffic by limiting the input or output transmission rate of a class of traffic based on user-defined criteria. Policing traffic allows you to control the maximum rate of traffic sent or received on an interface and to partition a network into multiple priority levels or class of service (CoS).
- **Queue**: A queue, in computer networking, is a collection of data packets collectively waiting to be transmitted by a network device using a per-defined structure methodology.

The Model is divided into two sections, based on operation at the ingress side and egress side. In this model, we are considering unidirectional traffic. All vehicles in the model will have a VLAN ID and priority. We are going to create two services (S1: VLAN ID 100 and S2: VLAN ID 200) between the ones and egress port. Policers will be applied on S1 services allocating 25% and 75% of resource to S1 and S2 respectively. Consider a VIP vehicle transit (VLAN 100), it will go through S1 and any public vehicle (VLAN 200) will go through S2. Since policer is applied on S2, the smooth transition of VIP vehicle is ensured. At the egress port, the traffic will egress out through a particular queue based on the priority.

4.5 Model Implementation

We have tried to implement this model on the various platforms as follows:

- **SDN controller ONOS**
- **SDN controller ODL (Opendaylight)**
- **IOSvL2 switch in GNS3**
- **CSR1000V in GNS3**

- **Cisco 2950 Catalyst Switch (Hardware)**
- **Cisco 2800 Router (Hardware)**

There were several limitations associated with these platforms. Because the hardware devices were very old, they could not support the QoS features. We also had to face the license issue with the necessary features. Applications for the required features in the controller were not getting activated. These limitations made us move to Layer three which is the Network Layer. In this layer, we used only one feature, the Access Control List called ACL, which will be discussed in detail in the upcoming section.

4.6 Networking Model at Layer three

Figure shows the Real -life model.

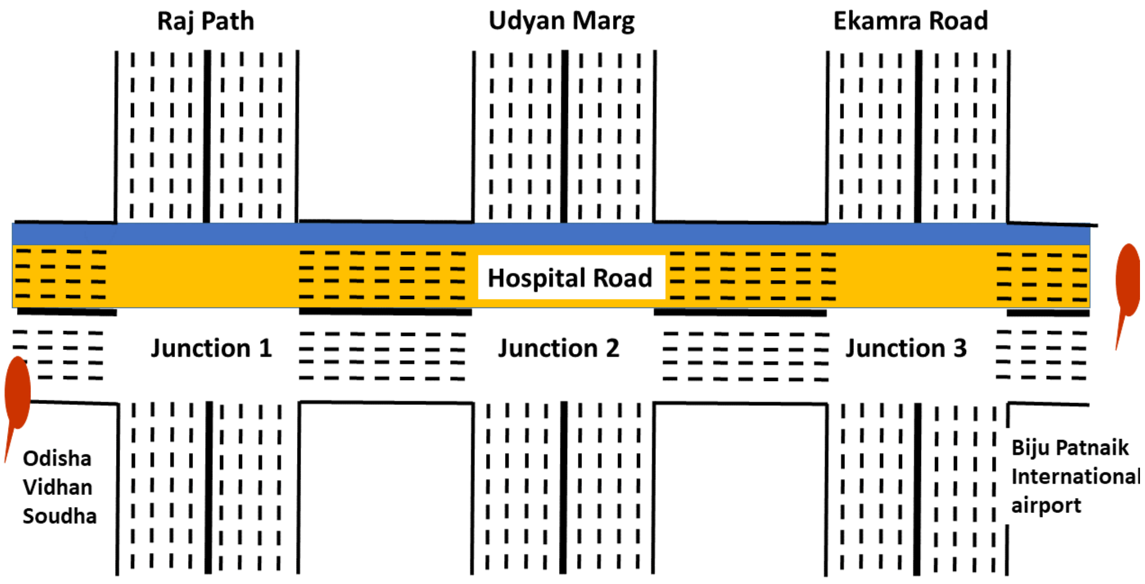


FIGURE 4.6: Networking model at layer three

In this model. we overlay a layer three network on top of it, where

- Packets are reprsented as Vehicles.
- Junctions are represented as Switches.

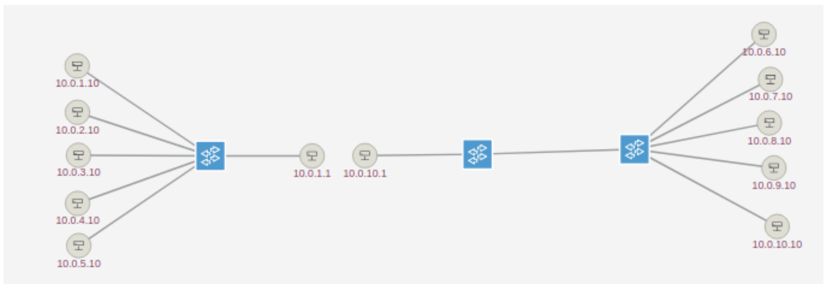


FIGURE 4.7: Networking model at layer three

- Interfaces to the Switches are lanes.
- Generators are the localities.
- Access Control List (ACL) is used to block or allow the traffic which represents red and green light in real-road scenario.

This is a model for driverless cars where the traffic flows in different arterial roads and traffic flows smoothly in different lanes. As mentioned earlier, the first lane is specifically reserved for VIP or emergency transit and the other four lanes are reserved for normal transit. In this model, traffic from VIP or emergency transit represents one subnet IP and traffic from other lanes represents another subnet IP. Assuming the link bandwidth between the two switches 100 Mbps, the resources are allocated to each lane in the following manner. 25 Mbps to the first lane, second lane and third lane. 20 Mbps to the fourth lane. And 5 Mbps to the fifth lane. In this model traffic is filled lane wise which means untill and unless the traffic is completely filled in the first lane the next lane is not utilized. As and when the resources are used in the corresponding lane, the traffic is sent to the next lane. We will be continuously poling at the fifth lane. If the fifth interface receives more than 5 Mbps, the ACL will be triggered and traffic will be blocked and if it is less than 5 Mbps ACL is removed. In this way, we will ensure that the congestion is avoided and the smooth flow of traffic is ensured.

4.7 Access Control List (ACL)

Access Control List are filter that enables you to control which routing updates or packets are permitted or denied in or out of a network. They are specifically used by network adminstrator to filter traffic and to provide extra security for the network. It contains a list of conditions that categorize packets and help you determine when to allow or deny network traffic. They are applied on the interface basis to packets leaving or entering an interface.

Classification for filtering traffic can be based on source and destination IP address, source port and destination port and the protocol of the packet. When a packet arrives, the device extracts certain information from the packet header and makes decision according to the filter rules as to whether the packet can pass through or be dropped.

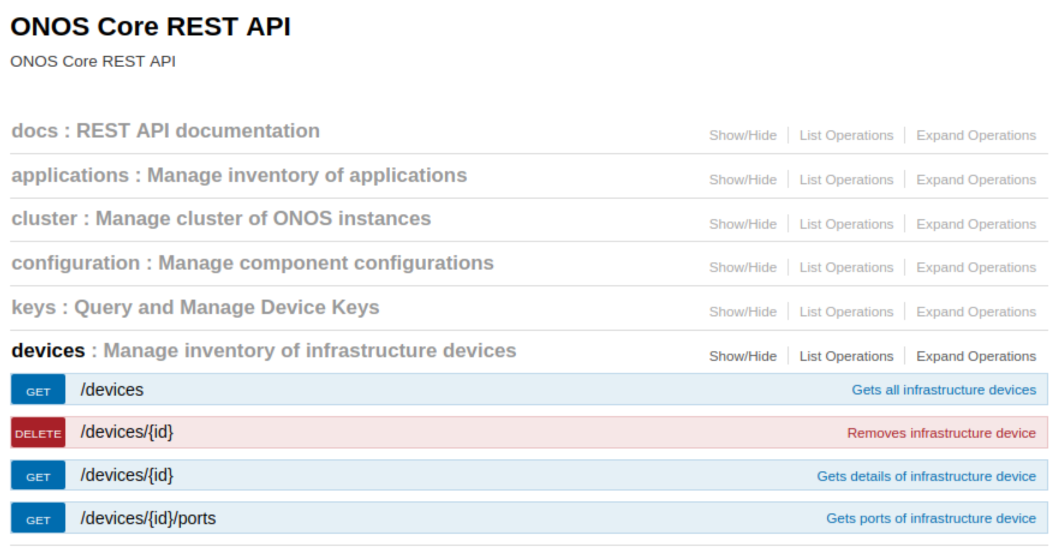
Types of Access Control List

- **Standard access lists**: Allow you to evaluate only on the source IP address of a packet. Standard ACL's are not as powerful as extended access list, but they are less CPU intensive for the device.
- **Extended Access List**: Allow you to evaluate the source and destination IP address, the layer 3 protocol, source and destination port and other parameters. They are more complex to configure and require more CPU time than the standard ACL's.

4.8 RESTful API

Representational State Transfer (REST) API also known as RESTful API which is a protocol for communication between the control plane and the application plane. It's just a documented method of interacting with someone else's service. ONOS controller provides REST API to communicate with the application plane. ONOS controller has a very large number of REST API as shown in Figure 4.8. ONOS uses Swagger to auto-generate REST API documentation [11]. This can be viewed from any running ONOS instance at the following URL: http:// ONOS IP:8181/onos/v1/docs/. The important part in using REST API is all the endpoint needs to be prefixed with `http://ONOSIP:8181/onos/v1/docs/`. Just to given an example:

`http://127.0.0.1:8181/onos/v1/devices`



ONOS Core REST API

ONOS Core REST API

docs : REST API documentation — Show/Hide | List Operations | Expand Operations

applications : Manage inventory of applications — Show/Hide | List Operations | Expand Operations

cluster : Manage cluster of ONOS instances — Show/Hide | List Operations | Expand Operations

configuration : Manage component configurations — Show/Hide | List Operations | Expand Operations

keys : Query and Manage Device Keys — Show/Hide | List Operations | Expand Operations

devices : Manage inventory of infrastructure devices — Show/Hide | List Operations | Expand Operations

GET	/devices	Gets all infrastructure devices
DELETE	/devices/{id}	Removes infrastructure device
GET	/devices/{id}	Gets details of infrastructure device
GET	/devices/{id}/ports	Gets ports of infrastructure device

FIGURE 4.8: ONOS REST API
Image source: Screenshot from PC

If ONOS is running on your local machine. Where `127.0.0.1:8181` is IP of the controller.

In the RTMS model we have made use of ACL API which we will discuss in the next chapter.

Chapter 5

Results

"If theres some unique problem you happen to have that traditional networking architectures cant resolve for you due to technical, practical, or financial limitations, then OpenFlow might just provide a liberating answer.". –Ethan Banks

5.1 Preliminary Results

In this chapter, we will focus on how the ACL API is used to allow or block the traffic which represents the red and green light scenario in a real-life model. The ACL API is provided by the SDN controller ONOS.

Consider a topology as shown in Figure 5.1 with one switch representing the single junction and three hosts representing the different lanes. The respective hosts IPs are as follows:

HOST 1 : 10.0.0.1
HOST 2 : 10.0.0.2
HOST 3 : 10.0.0.3

Consider a scenario in which no traffic is blocked, this means that all hosts can send traffic to each other. Figure 5.2 shows ping takes place among all the hosts.

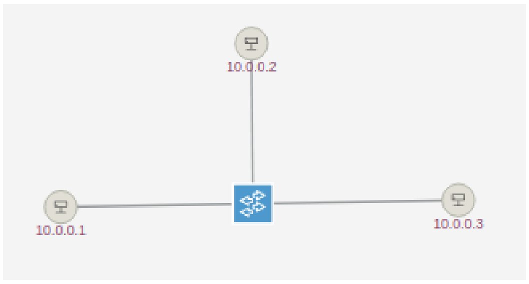
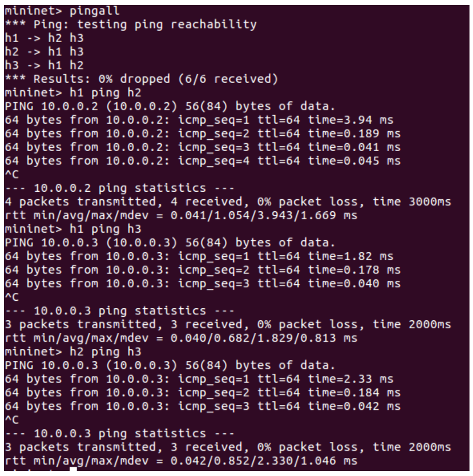


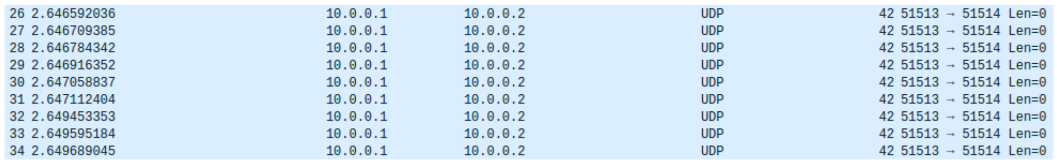
FIGURE 5.1: Topology



```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=3.94 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.189 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.041 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.045 ms
^C
--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3000ms
rtt min/avg/max/mdev = 0.041/1.054/3.943/1.669 ms
mininet> h1 ping h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=1.82 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.178 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.040 ms
^C
--- 10.0.0.3 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2000ms
rtt min/avg/max/mdev = 0.040/0.682/1.829/0.813 ms
mininet> h2 ping h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=2.33 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.184 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.042 ms
^C
--- 10.0.0.3 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2000ms
rtt min/avg/max/mdev = 0.042/0.852/2.330/1.046 ms
```

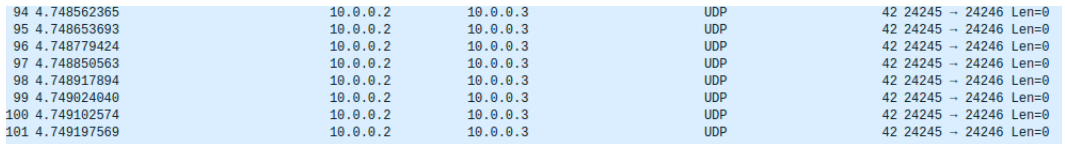
FIGURE 5.2: Ping among all the hosts

Packets were also sent from H1 to H2 and from H2 to H3. Figure 5.3 shows packet captured from wireshark.



```
26 2.646592036    10.0.0.1    10.0.0.2    UDP    42 51513 → 51514 Len=0
27 2.646709385    10.0.0.1    10.0.0.2    UDP    42 51513 → 51514 Len=0
28 2.646784342    10.0.0.1    10.0.0.2    UDP    42 51513 → 51514 Len=0
29 2.646916352    10.0.0.1    10.0.0.2    UDP    42 51513 → 51514 Len=0
30 2.647058837    10.0.0.1    10.0.0.2    UDP    42 51513 → 51514 Len=0
31 2.647112404    10.0.0.1    10.0.0.2    UDP    42 51513 → 51514 Len=0
32 2.649453353    10.0.0.1    10.0.0.2    UDP    42 51513 → 51514 Len=0
33 2.649595184    10.0.0.1    10.0.0.2    UDP    42 51513 → 51514 Len=0
34 2.649689045    10.0.0.1    10.0.0.2    UDP    42 51513 → 51514 Len=0
```

FIGURE 5.3: Packet sent from H1 to H2

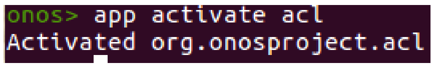


```
 94 4.748562365    10.0.0.2    10.0.0.3    UDP    42 24245 → 24246 Len=0
 95 4.748653693    10.0.0.2    10.0.0.3    UDP    42 24245 → 24246 Len=0
 96 4.748779424    10.0.0.2    10.0.0.3    UDP    42 24245 → 24246 Len=0
 97 4.748850563    10.0.0.2    10.0.0.3    UDP    42 24245 → 24246 Len=0
 98 4.748917894    10.0.0.2    10.0.0.3    UDP    42 24245 → 24246 Len=0
 99 4.749024040    10.0.0.2    10.0.0.3    UDP    42 24245 → 24246 Len=0
100 4.749102574    10.0.0.2    10.0.0.3    UDP    42 24245 → 24246 Len=0
101 4.749197569    10.0.0.2    10.0.0.3    UDP    42 24245 → 24246 Len=0
```

FIGURE 5.4: Packet sent from H2 to H3

Now let's apply the ACL and see how traffic is blocked. Following are the steps involved in order to apply the ACL.

Step 1: Activate the ACL application a.k.a ACL app, from the ONOS CLI.



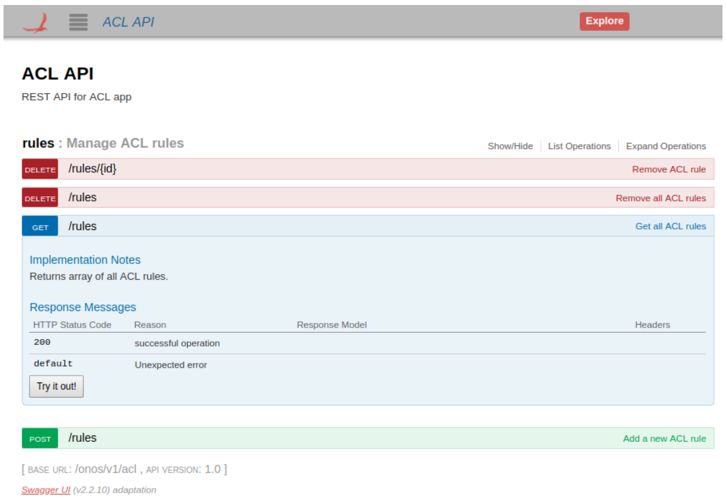
```
onos> app activate acl
Activated org.onosproject.acl
```

FIGURE 5.5: ACL app activated

Step 2: Get the REST API via

http://172.17.0.2:8181/onos/v1/docs/#!/rules/get_rules

Where 172.17.0.2:8181 is IP of the controller. Figure 5.6 shows the ACL API.



ACL API

REST API for ACL app

rules : Manage ACL rules

DELETE /rules/{id} Remove ACL rule

DELETE /rules Remove all ACL rules

GET /rules Get all ACL rules

Implementation Notes

Returns array of all ACL rules.

Response Messages

HTTP Status Code	Reason	Response Model	Headers
200	successful operation		
default	Unexpected error		

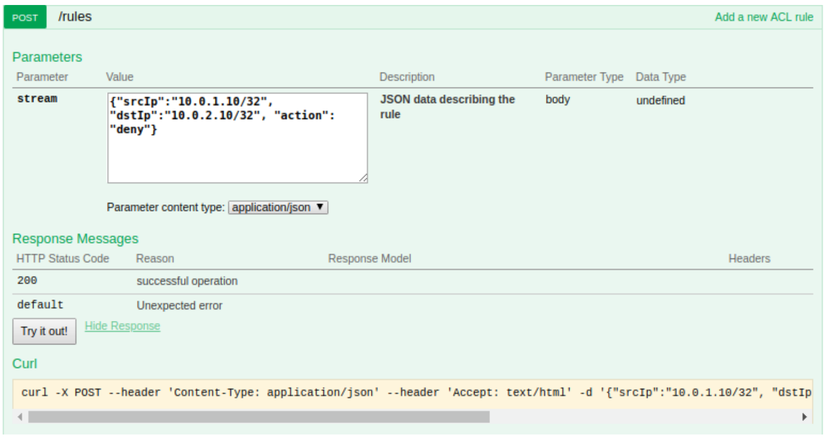
POST /rules Add a new ACL rule

[BASE URL: /onos/v1/acl , API VERSION: 1.0]

Swagger UI (v2.2.10) adaptation

FIGURE 5.6: ACL API

Step 3: POST the rule, that says traffic will be blocked from H1 to H2.



POST /rules Add a new ACL rule

Parameters

Parameter	Value	Description	Parameter Type	Data Type
stream	{"srcIp":"10.0.1.10/32", "dstIp":"10.0.2.10/32", "action": "deny"}	JSON data describing the rule	body	undefined

Parameter content type: application/json

Response Messages

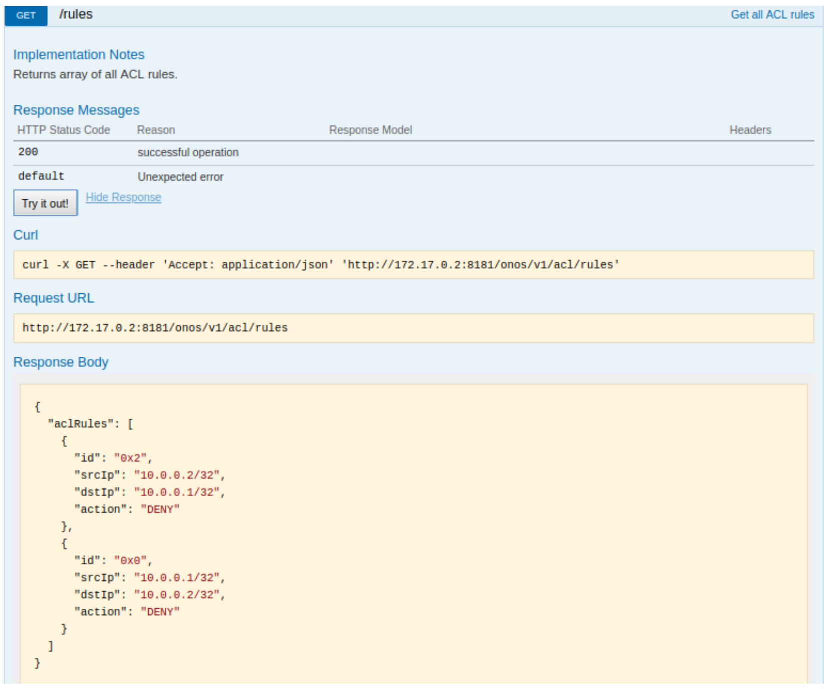
HTTP Status Code	Reason	Response Model	Headers
200	successful operation		
default	Unexpected error		

Curl

```
curl -X POST --header 'Content-Type: application/json' --header 'Accept: text/html' -d '{"srcIp":"10.0.1.10/32", "dstIp
```

FIGURE 5.7: POST rule

Step 4: GET the rule. Rules will be shown in the response body denying traffic from H1 to H2 and H2 to H1 and these rules will be installed on the switch.



Returns array of all ACL rules.

HTTP Status Code	Reason	Response Model	Headers
200	successful operation		
default	Unexpected error		

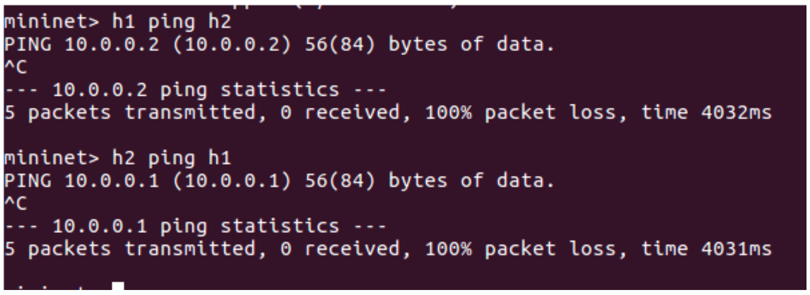
```
curl -X GET --header 'Accept: application/json' 'http://172.17.0.2:8181/onos/v1/acl/rules'
```

```
http://172.17.0.2:8181/onos/v1/acl/rules
```

```
{
  "aclRules": [
    {
      "id": "0x2",
      "srcIp": "10.0.0.2/32",
      "dstIp": "10.0.0.1/32",
      "action": "DENY"
    },
    {
      "id": "0x0",
      "srcIp": "10.0.0.1/32",
      "dstIp": "10.0.0.2/32",
      "action": "DENY"
    }
  ]
}
```

FIGURE 5.8: GET rules

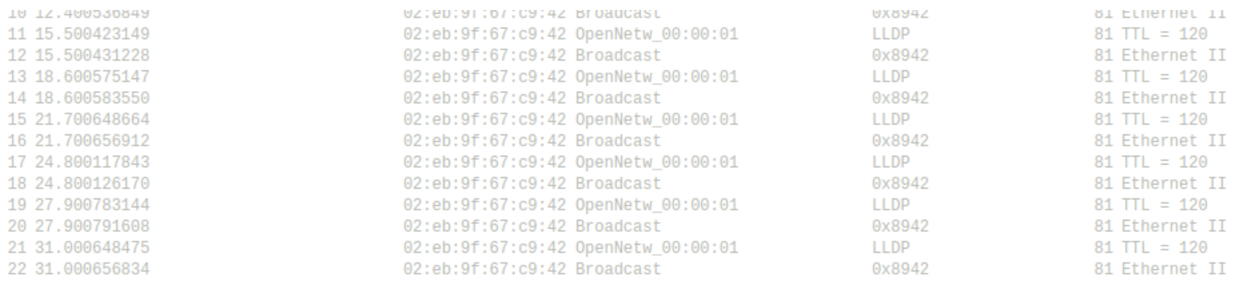
Step 5: Now, if we ping between H1 and H2 and H2 and H1 it won't be successful as shown in Figure 5.9.



```
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
^C
--- 10.0.0.2 ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 4032ms

mininet> h2 ping h1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
^C
--- 10.0.0.1 ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 4031ms
```

FIGURE 5.9: Ping couldn't take place

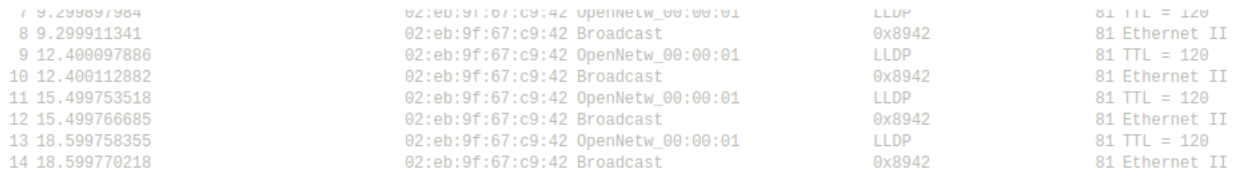
Step 6: Packets were also not received as the rule was applied, which was confirmed using a Wireshark. Figure 5.10 shows no packets were received when sent from H1 to H2.



```
10 12.400536849    02:eb:9f:67:c9:42 Broadcast           0x8942    81 Ethernet II
11 15.500423149    02:eb:9f:67:c9:42 OpenNetw_00:00:01   LLDP      81 TTL = 120
12 15.500431228    02:eb:9f:67:c9:42 Broadcast           0x8942    81 Ethernet II
13 18.600575147    02:eb:9f:67:c9:42 OpenNetw_00:00:01   LLDP      81 TTL = 120
14 18.600583550    02:eb:9f:67:c9:42 Broadcast           0x8942    81 Ethernet II
15 21.700648664    02:eb:9f:67:c9:42 OpenNetw_00:00:01   LLDP      81 TTL = 120
16 21.700656912    02:eb:9f:67:c9:42 Broadcast           0x8942    81 Ethernet II
17 24.800117843    02:eb:9f:67:c9:42 OpenNetw_00:00:01   LLDP      81 TTL = 120
18 24.800126170    02:eb:9f:67:c9:42 Broadcast           0x8942    81 Ethernet II
19 27.900783144    02:eb:9f:67:c9:42 OpenNetw_00:00:01   LLDP      81 TTL = 120
20 27.900791608    02:eb:9f:67:c9:42 Broadcast           0x8942    81 Ethernet II
21 31.000648475    02:eb:9f:67:c9:42 OpenNetw_00:00:01   LLDP      81 TTL = 120
22 31.000656834    02:eb:9f:67:c9:42 Broadcast           0x8942    81 Ethernet II
```

Figure 5.10: Wireshark indicating non-recipient of packets

And also packets were not received when sent from H2 to H1.



```
 7 9.299897984     02:eb:9f:67:c9:42 OpenNetw_00:00:01   LLDP      81 TTL = 120
 8 9.299911341     02:eb:9f:67:c9:42 Broadcast           0x8942    81 Ethernet II
 9 12.400097886    02:eb:9f:67:c9:42 OpenNetw_00:00:01   LLDP      81 TTL = 120
10 12.400112882    02:eb:9f:67:c9:42 Broadcast           0x8942    81 Ethernet II
11 15.499753518    02:eb:9f:67:c9:42 OpenNetw_00:00:01   LLDP      81 TTL = 120
12 15.499766685    02:eb:9f:67:c9:42 Broadcast           0x8942    81 Ethernet II
13 18.599758355    02:eb:9f:67:c9:42 OpenNetw_00:00:01   LLDP      81 TTL = 120
14 18.599770218    02:eb:9f:67:c9:42 Broadcast           0x8942    81 Ethernet II
```

Figure 5.11: Wireshark indicating non-recipient of packets

5.2 Model Results

Figure 5.12 shows the Networking model, as mentioned earlier where interfaces to the switch represents lanes and switches represent the junctions and the packets represent the vehicles.

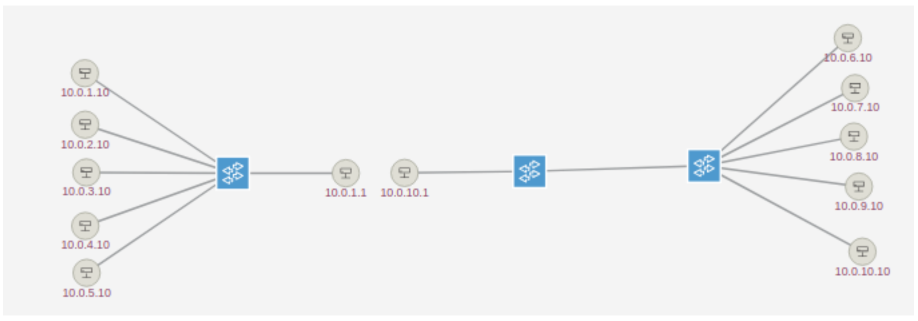
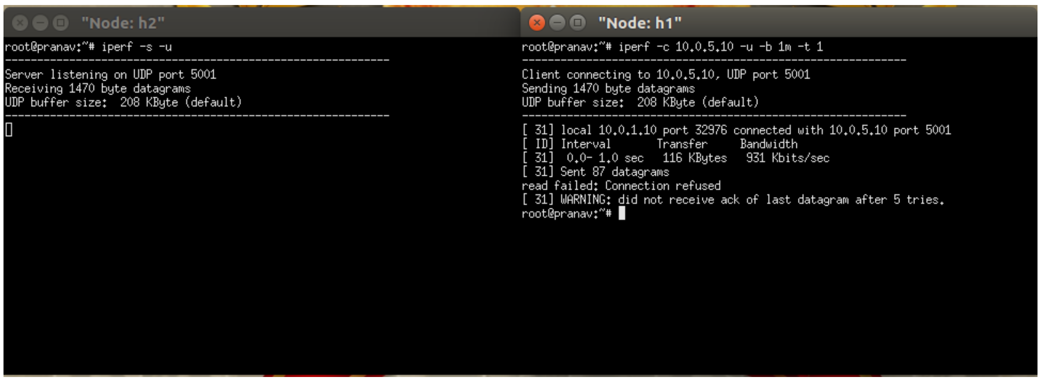


Figure 5.12: Networking model

Now let's say we want to block traffic on a first lane. The packets were sent by the packet generator called ***Iperf***, its a terminal based generator which sends packets based on seconds. UDP packets were sent from H1 to H5 using a packet generator. Let's look at how packets are sent from the generator.
Open xterm for H1 and H5 as shown in Figure 5.13.


"Node: h2"

```
root@pranav:~# iperf -s -u
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
```

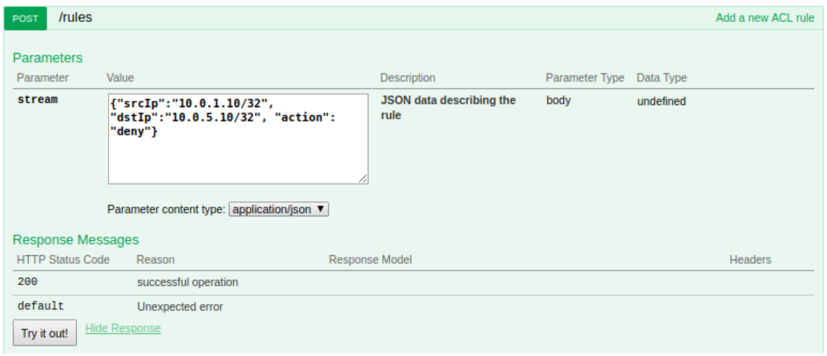
"Node: h1"

```
root@pranav:~# iperf -c 10.0.5.10 -u -b 1m -t 1
------------------------------------------------------------
Client connecting to 10.0.5.10, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[ 31] local 10.0.1.10 port 32976 connected with 10.0.5.10 port 5001
[ ID] Interval       Transfer     Bandwidth
[ 31]  0.0- 1.0 sec   116 KBytes   931 Kbits/sec
[ 31] Sent 87 datagrams
read failed: Connection refused
[ 31] WARNING: did not receive ack of last datagram after 5 tries.
root@pranav:~#
```

FIGURE 5.13: Packet Generator

The typical way that iperf is used is to first start one iperf process running in server mode as the traffic receiver, and then start another iperf process running in client mode on another host as the traffic sender. In order to send a single UDP stream from 10.0.1.10 i.e, H1 to 10.0.5.10 i.e, H5 , we would run iperf in server mode on H5 and iperf in client mode on H1.

The procedure for blocking traffic has already been mentioned in the preliminary results. After activating the app, GET the rule and then post the rule that will be installed on the switch as shown in Figure 5.16


POST /rules Add a new ACL rule

Parameters

Parameter	Value	Description	Parameter Type	Data Type
stream	{"srcIp":"10.0.1.10/32", "dstIp":"10.0.5.10/32", "action": "deny"}	JSON data describing the rule	body	undefined

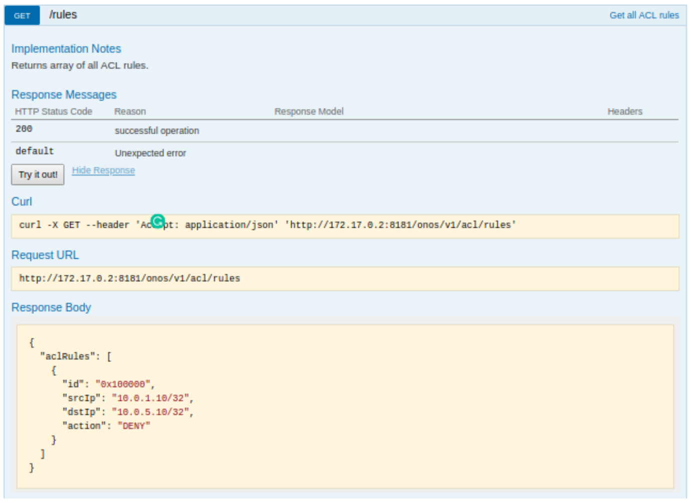
Parameter content type: application/json

Response Messages

HTTP Status Code	Reason	Response Model	Headers
200	successful operation		
default	Unexpected error		

FIGURE 5.14: POST RULES

This is how we can block traffic on a particular lane that represents the real - life scenario.



Returns array of all ACL rules.

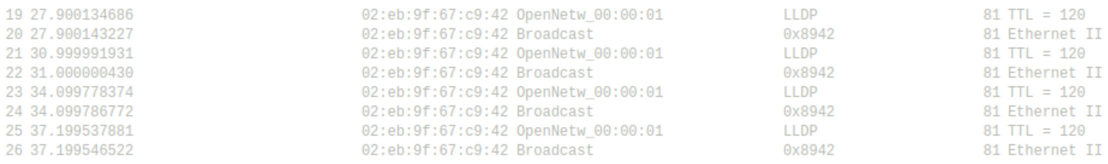
HTTP Status Code	Reason	Response Model	Headers
200	successful operation		
default	Unexpected error		

```
curl -X GET --header 'Ac[illegible]pt: application/json' 'http://172.17.0.2:8181/onos/v1/acl/rules'
```

```
http://172.17.0.2:8181/onos/v1/acl/rules
```

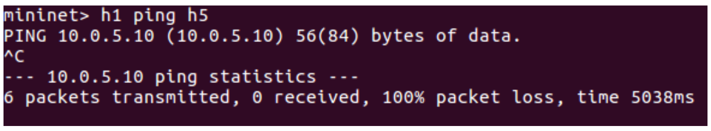
```
{
  "aclRules": [
    {
      "id": "0x100000",
      "srcIp": "10.0.1.10/32",
      "dstIp": "10.0.5.10/32",
      "action": "DENY"
    }
  ]
}
```

FIGURE 5.15: GET RULES



```
19 27.900134686                              02:eb:9f:67:c9:42 OpenNetw_00:00:01           LLDP                   81 TTL = 120
20 27.900143227                              02:eb:9f:67:c9:42 Broadcast                   0x8942                 81 Ethernet II
21 30.999991931                              02:eb:9f:67:c9:42 OpenNetw_00:00:01           LLDP                   81 TTL = 120
22 31.000000430                              02:eb:9f:67:c9:42 Broadcast                   0x8942                 81 Ethernet II
23 34.099778374                              02:eb:9f:67:c9:42 OpenNetw_00:00:01           LLDP                   81 TTL = 120
24 34.099786772                              02:eb:9f:67:c9:42 Broadcast                   0x8942                 81 Ethernet II
25 37.199537881                              02:eb:9f:67:c9:42 OpenNetw_00:00:01           LLDP                   81 TTL = 120
26 37.199546522                              02:eb:9f:67:c9:42 Broadcast                   0x8942                 81 Ethernet II
```

FIGURE 5.16: Wireshark indicating non recepient of packets



```
mininet> h1 ping h5
PING 10.0.5.10 (10.0.5.10) 56(84) bytes of data.
^C
--- 10.0.5.10 ping statistics ---
6 packets transmitted, 0 received, 100% packet loss, time 5038ms
```

FIGURE 5.17: Ping couldn't take place

5.3 Application Development for a Road Traffic Model

As we have seen before, we manually apply POST and GET rules via the REST API to allow or block traffic. In this section, we develop an application for the real - life road scenario where the controller controls traffic instead of the traffic police. Figure 5.18 shows the flow chart for the application development.

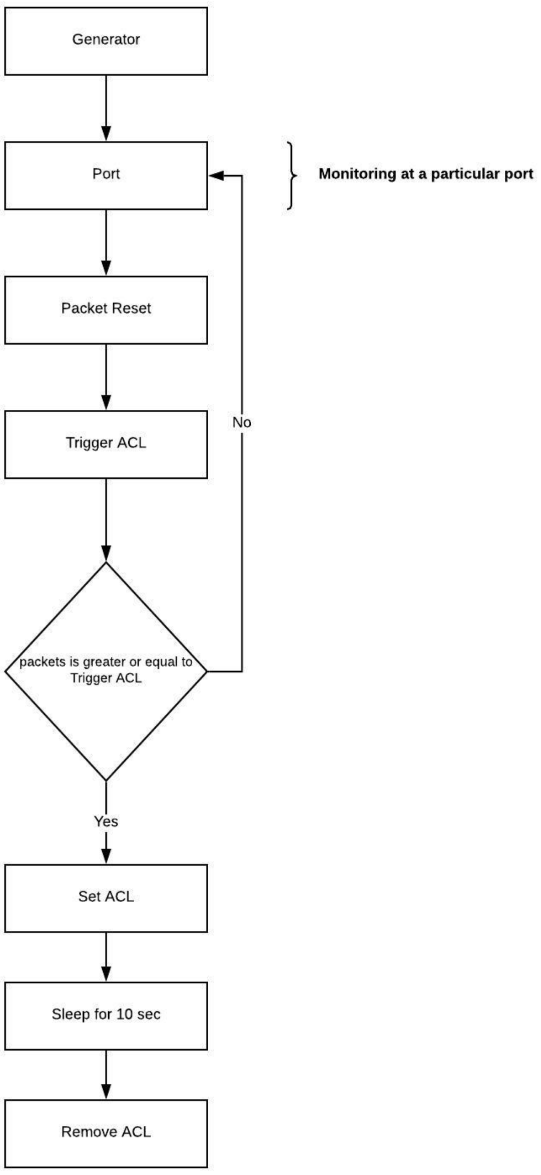
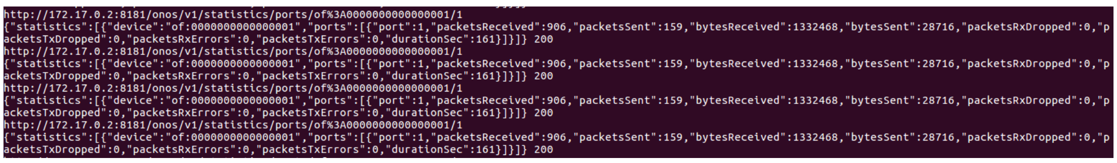


FIGURE 5.18: Flow rule for application

As the packets are begin sent from the generator, we will be monitoring the packet count at the particular port. Here we are setting two variables, PACKET_RESET and PACKET_TO_TRIGGER_ACL. When PACKET_TO_TRIGGER_ACL - PACKET_RESET is satisfied, ACL will be triggered and traffic will be blocked. To put it in a simpler way, when packet count is greater or equal to PACKET_TO_TRIGGER_ACL, traffic will be blocked. And if the condition is not met, it goes back to port monitoring. Once the condition is met, it means that the ACL is set after which it is waiting for 10 seconds and after that traffic is allowed again.

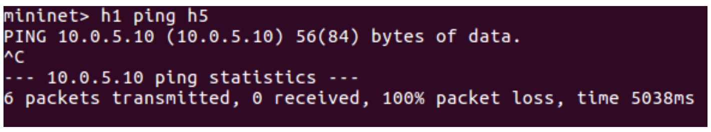
To relate this to our model, we will monitor the number of vehicles entering the junction and as soon as congestion is detected, the traffic will be blocked representing the red light scenario and after waiting for 10 seconds, the traffic will be allowed representing the green light scenario. Figure 5.19 displays the monitoring of packet count from the terminal at a particular port. And as soon as the congestion is detected the process stops. The confirmation of our application was done by ping, once the traffic was blocked and the process stopped, we checked by ping whether the traffic was blocked or not and we observed that the ping could not take place which ensured that our application worked perfectly well.



```
http://172.17.0.2:8181/onos/v1/statistics/ports/of%3A0000000000000001/1
{"statistics":[{"device":"of:0000000000000001","ports":[{"port":1,"packetsReceived":906,"packetsSent":159,"bytesReceived":1332468,"bytesSent":28716,"packetsRxDropped":0,"p
acketsTxDropped":0,"packetsRxErrors":0,"packetsTxErrors":0,"durationSec":161}]}]} 200
http://172.17.0.2:8181/onos/v1/statistics/ports/of%3A0000000000000001/1
{"statistics":[{"device":"of:0000000000000001","ports":[{"port":1,"packetsReceived":906,"packetsSent":159,"bytesReceived":1332468,"bytesSent":28716,"packetsRxDropped":0,"p
acketsTxDropped":0,"packetsRxErrors":0,"packetsTxErrors":0,"durationSec":161}]}]} 200
http://172.17.0.2:8181/onos/v1/statistics/ports/of%3A0000000000000001/1
{"statistics":[{"device":"of:0000000000000001","ports":[{"port":1,"packetsReceived":906,"packetsSent":159,"bytesReceived":1332468,"bytesSent":28716,"packetsRxDropped":0,"p
acketsTxDropped":0,"packetsRxErrors":0,"packetsTxErrors":0,"durationSec":161}]}]} 200
http://172.17.0.2:8181/onos/v1/statistics/ports/of%3A0000000000000001/1
{"statistics":[{"device":"of:0000000000000001","ports":[{"port":1,"packetsReceived":906,"packetsSent":159,"bytesReceived":1332468,"bytesSent":28716,"packetsRxDropped":0,"p
acketsTxDropped":0,"packetsRxErrors":0,"packetsTxErrors":0,"durationSec":161}]}]} 200
```

FIGURE 5.19: Packet count from terminal

Figure 5.20 shows that after the application was run, the ping could not take place.



```
mininet> h1 ping h5
PING 10.0.5.10 (10.0.5.10) 56(84) bytes of data.
^C
--- 10.0.5.10 ping statistics ---
6 packets transmitted, 0 received, 100% packet loss, time 5038ms
```

FIGURE 5.20: Ping couldn't take place

5.4 Conclusion

- In this work, we started using the REST ACL API to manually block or allow traffic.
- Then we started building an application for real - life road scenario to block or allow traffic over an SDN controller.
- The application is still being developed where traffic is still to be unblocked after 10 seconds of waiting, which, due to certain override condition, could not get implemented.

5.5 Future Outlook

- As part of future work, we would like to reroute traffic in the event of any lane resources not being used.
- We would like to scale our work to a greater number of junctions, meaning a greater number of switches and perform simulation over a longer period of time and observe the behavior of the real - life road scenario.

Bibliography

[1] Diego Kreutz, Fernando MV Ramos, Paulo Verissimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, 2015.

[2] Software-defined networking (sdn): Layers and architecture terminology. URL https://tools.ietf.org/html/rfc7426.

[3] Onf, open networking foundation, openflow switch specification, version 1.3.2 (wire protocol 0x04), april 25, 2013, onf ts-009.

[4] Open networking operating system (onos). URL https://onosproject.org/.

[5] Sdn series part seven: Onos, mar 2015, by sridhar rao. URL https://thenewstack.io/open-source-sdn-controllers-part-vii-onos/.

[6] Mininet overview. URL http://mininet.org/overview/.

[7] Nikos Bizanis and Fernando A Kuipers. Sdn and virtualization solutions for the internet of things: A survey. *IEEE Access*, 4:5591–5606, 2016.

[8] Sdn based networks for internet of things: How onos helps. URL https://hsc.com/Blog/SDN-based-Networks-for-Internet-of-things-How-ONOS-helps.

[9] Yuhong Li, Xiang Su, Jukka Riekki, Theo Kanter, and Rahim Rahmani. A sdn-based architecture for horizontal internet of things services. In *2016 IEEE International Conference on Communications (ICC)*, pages 1–7. IEEE, 2016.

[10] Soufian Toufga, Philippe Owezarski, Slim Abdellatif, and Thierry Villemur. An sdn hybrid architecture for vehicular networks: Application to intelligent transport system. *arXiv preprint arXiv:1712.05307*, 2017.

[11] Onos rest api. URL https://wiki.onosproject.org/display/ONOS/REST.

[12] Software-Defined Networks and OpenFlow- The Internet Protocol Journal, Volume 16, No.1, by William Stallings. https://www.cisco.com/c/en/us/about/press/internet-protocol-journal/back-issues/table-contents-59/161-sdn.htmlreference3

[13] Qin, Zhijing, Grit Denker, Carlo Giannelli, Paolo Bellavista, and Nalini Venkatasubramanian. *"A software defined networking architecture for the internet-of-things."* IEEE network operations and management symposium (NOMS), pp. 1-9. IEEE, 2014.

[14] Rego, Albert, Laura Garcia, Sandra Sendra, and Jaime Lloret. *"Software defined networks for traffic management in emergency situations."* Fifth International Conference on Software Defined Systems (SDS), pp. 45-51. IEEE, 2018.

[15] Berde, Pankaj, Matteo Gerola, Jonathan Hart, Yuta Higuchi, Masayoshi Kobayashi, Toshio Koide, Bob Lantz et al. *"ONOS: towards an open, distributed SDN OS."* In Proceedings of the third workshop on Hot topics in software defined networking, pp. 1-6. ACM, 2014.

[16] Tayyaba, Sahrish Khan, Munam Ali Shah, Naila Sher Afzal Khan, Yousra Asim, Wajeeha Naeem, and Muhammad Kamran. *"Software-defined networks (SDNs) and Internet of Things (IoTs): A qualitative prediction for 2020.* network 7, no. 11 (2016).

[17] Ogrodowczyk, ukasz, Bartosz Belter, and Marc LeClerc. *"IoT ecosystem over programmable SDN infrastructure for smart city applications."* In 2016 Fifth European Workshop on Software-Defined Networks (EWSDN), pp. 49-51. IEEE, 2016.

[18] Huang, X., Yu, R., Kang, J., Xia, Z. and Zhang, Y., 2018. *Software defined networking for energy harvesting internet of things.* IEEE Internet of Things Journal, 5(3), pp.1389-1399.

[19] Bizanis, Nikos, and Fernando A. Kuipers. *"SDN and virtualization solutions for the Internet of Things: A survey."* IEEE Access 4 (2016): 5591-5606.

[20] Jararweh, Yaser, Mahmoud Al-Ayyoub, Elhadj Benkhelifa, Mladen Vouk, and Andy Rindos. *"SDIoT: a software defined based internet of things framework."* Journal of Ambient Intelligence and Humanized Computing 6, no. 4 (2015): 453-461.

[21] Nadeau, Thomas D, and Ken Gray. Software Defined Netwrok. 1st ed., O'Reilly, 2013.