

# **Towards development of Intent Driven Network(IDN)**

A Thesis Submitted  
in Partial Fulfillment of the Requirements  
for the Degree of  
**Master of Technology**  
in  
**Information Security**

by  
**Kumar Shubham**  
2019IS14

Under the Guidance of  
**Dr. Shashwati Banerjea**



**COMPUTER SCIENCE AND ENGINEERING DEPARTMENT**  
**MOTILAL NEHRU NATIONAL INSTITUTE OF TECHNOLOGY ALLAHABAD**  
**PRAYAGRAJ – 211004, INDIA**  
**29<sup>th</sup> June, 2021**

# UNDERTAKING

I declare that the work presented in this thesis titled “*Towards Development of Intent Driven Network(IDN)*”, submitted to the Computer Science and Engineering Department, Motilal Nehru National Institute of Technology Allahabad, for the award of the ***Master of Technology*** degree in ***Information Security***, is my original work. I have not plagiarized or submitted the same work for the award of any other degree. In case this undertaking is found incorrect, I accept that my degree may be unconditionally withdrawn.

29<sup>th</sup> June, 2021

Prayagraj

Kumar Shubham

(Kumar Shubham)

# CERTIFICATE

Certified that the work contained in the thesis titled “*Towards development of Intent Driven Network(IDN)* ”, by *Kumar Shubham*, Registration Number *2019IS14* has been carried out under my supervision and this work has not been submitted elsewhere for a degree.

---

(Dr. Shashwati Banerjea)  
Computer Science and Engineering Dept.  
MNNIT Allahabad

29<sup>th</sup> June, 2021

# Acknowledgment

I would like to take this opportunity to express my deep sense of gratitude to my supervisor(Dr. Shashwati Banerjea) who provided me to work on the thesis topic "*Towards Development of Intent Driven Network(IDN)*" which helped me in doing lots of research and I came to know about many new things in the field of computer networks. Secondly I would also like to thank my parents, friends and rest all other who directly or indirectly helped me in completion of my thesis work.

**- Kumar Shubham**

# Biographical Sketch

## Kumar Shubham

---

Address: PG Hostel, MNNIT Allahabad, Teliarganj, Prayagraj, Uttar Pradesh, 211004

District: Prayagraj

E-Mail: shubhamcsed@gmail.com , Contact. No. +91-9717743766.

### Education

- I am currently pursuing my M.Tech in Information Security from MNNIT Allahabad under the department of Computer Science and Engineering (CSED).
- I have completed my B.Tech in Electronics and Communication (ECE) from JS-SATE, Noida.
- I have completed my 12th grade and 10th grade with ICSE board.

**Father's Name:** Mr. Arvind Kumar

**Mother's Name:** Mrs. Rita Rani

– *Dedicated to my loving family for their kind love and support.*

***“Education is the manifestation of the perfection already in man.”***

**-Swami Vivekanand**

# Abstract

With the rapid growth of technology in networking arena there are higher demands for performance. In-order to address these growing demands of consumers, Software Defined Networking (SDN) is gaining importance day by day. SDN provides network flexibility, customization and optimized way of traffic management in the network by decoupling of control and packet forwarding planes. Though SDN model provides benefits but still configuration and management of network is hectic for network administration. Therefore, we require an advanced model of SDN model in-order to solve these issues. This model is known as Intent Driven Network (IDN) model. Ideally, IDN should allow operators to express their intents in natural language without requiring knowledge of low-level configurations, eliminating the need for learning new languages. In this paper I propose an IDN model with RYU as SDN controller. Intents are specified in natural languages. In order to process and perform cleaning up of text we have made use of the capabilities of NLP. With reduced intent, a specific task is identified to be performed. The task is then send to RYU controller in JSON format to be performed in real time. We also introduce the concepts of contradiction detection in order to identify any previous deployed intents on hosts and switches. For contradiction detection different machine learning algorithms and upon training and testing of different models we observe that decision tree classifier gives the best results. To our knowledge, this is the first work that provides such interface between network users (in the form of natural language) and RYU SDN controller along with contradiction detection.



# Contents

<b>Acknowledgment</b>	<b>iv</b>
<b>Biographical Sketch</b>	<b>v</b>
<b>Abstract</b>	<b>viii</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Preliminaries</b>	<b>3</b>
<b>3 Literature Review</b>	<b>11</b>
<b>4 Proposed Work</b>	<b>12</b>
4.1 Introduction . . . . .	12
4.2 Implementation details . . . . .	13
<b>5 Experimental setup</b>	<b>34</b>
5.1 Installation . . . . .	34
5.2 Environment Setup . . . . .	35
<b>6 Analysis and Results</b>	<b>36</b>
6.1 Machine learning model results . . . . .	36
6.2 Model Interaction . . . . .	39
<b>7 Future Work and Conclusion</b>	<b>57</b>
7.1 Future Work . . . . .	57

7.2 Conclusion . . . . .	57
<b>References</b>	<b>58</b>

# List of Figures

1	Minimal topology in Mininet . . . . .	8
2	Single topology in Mininet . . . . .	9
3	Linear topology in Mininet . . . . .	10
4	Custom topology in Mininet . . . . .	10
5	RYU based IDN model architecture . . . . .	12
6	Natural Language Processing(NLP) pipeline . . . . .	14
7	Intent after passing through tozenization phase . . . . .	15
8	Intent after passing through tozenization phase . . . . .	15
9	Figure depicting case conversion phase of NLP processing pipeline . . . .	16
10	Figure depicting case conversion phase of NLP processing pipeline . . . .	16
11	Figure depicting input/output of text filtration phase . . . . .	16
12	Figure depicting input/output of text filtration phase . . . . .	17
13	Figure depicting lemmatization phase of NLP pipeline . . . . .	17
14	Figure depicting lemmatization phase of NLP pipeline . . . . .	18
15	Figure depicting working of task identification phase and bypassing of contradiction detection phase . . . . .	18
16	Figure depicting working of task identification phase and next phase as contradiction detection . . . . .	20
17	Figure depicting working of contradiction detection phase . . . . .	22
18	ROC curve for Logistic Regression model . . . . .	37
19	ROC curve for Random Forest Classifier model . . . . .	37
20	ROC curve for Naive Bayes model . . . . .	38
21	ROC curve for Decision Tree Classifier model . . . . .	38

22	Figure depicting linear topology . . . . .	39
23	Figure depicting an intent to know the number of switches present in the topology . . . . .	40
24	Figure depicting an intent to know the hardware and software description of switch. . . . .	40
25	Figure depicting an intent to install a flow rule in switches in order to block host h1. . . . .	41
26	Figure depicting pingall operation in Mininet . . . . .	41
27	Figure depicting an intent to enable host h1 to communicate with all other hosts . . . . .	41
28	Pingall operation in Mininet . . . . .	42
29	Figure depicting an intent to block all the traffic between host h2 and host h3 . . . . .	42
30	Pingall operation in Mininet . . . . .	43
31	Intent to allow the host h2 and host h3 to communicate with each other . .	43
32	Pingall operation in Mininet . . . . .	44
33	Intent to allow host h2 and host h3 to communicate with each other . . . .	44
34	Pingall operation in Mininet . . . . .	45
35	Intent to block traffic between hosts h3 and host h4 . . . . .	45
36	Pingall operation in Mininet . . . . .	46
37	Intent to enable host h3 and host h4 to communicate . . . . .	46
38	Pingall operation in Mininet . . . . .	47
39	Intent to block the traffic between host h5 and host h7 . . . . .	47
40	Pingall operation in Mininet . . . . .	48
41	Intent to get the detailed flow entries installed in switch s1 . . . . .	48
42	Intent to get the detailed flow entries installed in switch s1 . . . . .	49
43	Intent to disable switch s5 from packet forwarding . . . . .	49
44	Pingall operation in Mininet . . . . .	50
45	Intent to enable switch s5 . . . . .	50
46	Pingall operation in Mininet . . . . .	51
47	Intent to enable switch s5 . . . . .	51

48	Pingall operation in Mininet . . . . .	52
49	Intent to allow host h5 and host h7 to communicate . . . . .	52
50	Pingall operation in Mininet . . . . .	53
51	Figure depicting an intent to get all the flow entries installed in switch s1 with rule id . . . . .	53
52	Intent for removal of a flow entry . . . . .	54
53	Intent to get the flows of switch with different rule id's . . . . .	54
54	Intent to get the status of all the switch present in the topology . . . . .	55
55	Figure depicting an intent to know the port status of switch s1 . . . . .	55
56	Intent to get the aggregate flows of switch s1 . . . . .	56

# List of Tables

1	Different user's intent . . . . .	19
2	Initial block status of hosts . . . . .	23
3	Initial enable status of hosts . . . . .	23
4	Block status of hosts after modification . . . . .	24
5	Current block status of hosts . . . . .	24
6	Current enable status of hosts . . . . .	24
7	Enable status of hosts after modification . . . . .	25
8	Initial block status of hosts . . . . .	25
9	Initial enable status of hosts . . . . .	26
10	Block status after modification . . . . .	26
11	Current block status of hosts . . . . .	27
12	Current enable status of hosts . . . . .	27
13	Enable status of hosts after modification . . . . .	28
14	Evaluated results for contradiction detection . . . . .	36

# Chapter 1

## Introduction

Traditional networking [16] techniques has been followed over years. With the growing demands of users for performance it has been observed that the traditional model is not able to satisfy the needs. Configuration and management of network is also hectic as control and data plane is bounded to one physical entity. In-order to satisfy the growing demands for performance Software Defined Networking(SDN) model was developed. SDN model made it possible to decouple the control and data plane. Though SDN model [11] was able to satisfy the performance issues of users but still configuration and management of network was a matter of concern as network engineers has to still write programming codes to manage network. Ease in configuration and management of network can be provided by introducing a new model known as Intent Driven Networking(IDN) model. The IDN model [17] that I propose introduces natural language based interface between network engineers and networking environment and it is coupled with contradiction detection of flows installed in switches. The model allows the operators to specify the intents in natural language to be processed without requiring the need to write programming codes for network management and learning new programming languages. Network management is not a difficult task as the model provides flexibility for management in operator's natural language.

Configuring and managing the network by writing complex programming codes can be very hectic process instead if we can provide a natural language based interface between operators and network we observe an ease in the process of network management [7]. I

propose a RYU SDN controller based IDN model such that operators can specify their network intents in natural language and after identifying some specific keywords the model can detect a network task and then the task in JSON format can be passed on to the RYU controller to take desired actions and generate its response. One of the greatest advantages of the model is that RYU controller is python based and hence all the programming written is in python and therefore all the latest technologies based on python can be effectively applied. Two of the major aspects of my model are:

- The Model provides an interface between network operators and network management and bridges the gap between them with the help of natural language processing tools. [7]
- The Model also provides contradiction detection which detects contradictory changes made by network operators. This feature is currently limited to number of hosts and switches. [18]



# Chapter 2

## Preliminaries

In order to understand the working of the proposed model it is important to know about some basic concepts which are introduced in this section. This section first introduces the networking concepts i.e. traditional and SDN network and then introduces the software's required i.e. Mininet, VirtualBox, etc.

- Networking concepts:

- Traditional Networking:

The traditional networking [16] has been used from a long time. One of the major drawbacks of traditional networking is coupling together of control and data plane in one physical system. Packets are forwarded from one device to another with help of local knowledge of routing paths. Hence, the forwarding devices like hubs, switches, routers etc. operate in distributed manner and there is no global knowledge of forwarding paths to these devices. As global knowledge of path is not known then devices may take longer time to compute an alternate path between source and destination if link failures occurs.

Traditional networks [11] are completely static in nature and inflexible to satisfy the needs of growing demands of users. Moreover, traditional networks are operated manually which consumes more time and has more operating and deployment costs. Also, for any complex network many control planes might be present and each control plane is configured separately which is a very

hectic process. Hence, in-order to solve these flaws and issues a new model known as Software Defined Networking(SDN) was introduced.

– Software Defined Networking(SDN):

Traditional networks as discussed previously are unable satisfy increased consumer demands. They are static and inflexible and hence they suffer from various performance issues. In-order to adjust and satisfy the requirements specified by consumers, SDN model [11, 16] was evolved. SDN model can be defined as the “decoupling of control and packet forwarding planes in the network”. SDN model is gaining lot of importance day by day as it offers an effective way of traffic management by the inclusion of logically centralized SDN controller. SDN controller provides a global view of entire network thus the forwarding devices can forward the packet based on the global knowledge and hence no distributed protocols are obeyed.

SDN [10] offers faster means for deployment of applications. Enterprises can effectively use SDN model for deployment of their applications without any kind of issues. SDN model is also cost effective and reduces the overall deployment and operating costs as compared to traditional networks. SDN controller is logically centralized and hence is no issues of single point failure. So, these are some of the important factors for the growth of Software Defined Network over traditional networks. SDN model comprises of three major planes i.e. Data, Control and Application plane. The architecture of SDN model is shown below:

\* Data plane:

Data plane is responsible for forwarding packets so that the packet finally reaches its destination. When packets arrives at forwarding device it can take particular action based on pre-installed ow rules. If flow rule for handling specific packet is not present then forwarding devices communicates with centralized SDN controller in-order to take specific actions for handling packets.

\* Control plane:

Control plane consists of centralized SDN controller which is responsible

for providing core services and global view of the entire network. It's the responsibility of SDN controller to manage the traffic in optimized way such that response time is as low as possible. Some of the services provided by control plane are:

- Topology services:

These services are responsible for determining the way by which forwarding devices are connected. Some of the pre-defined topology are minimal, single, linear and tree.

- Inventory services:

These services are responsible for tracking all SDN enabled devices.

- Statistics services:

These services provides counter information from forwarding devices. Counter information determines how times table hit occurs of ow tables maintained by forwarding devices.

- Host tracking services:

These services have the ability to track IP and MAC addresses of host located in a network. In-order to perform their services they might also use Address Resolution Protocol(ARP) for conversion of IP to MAC addresses.

- \* Application plane:

Application plane is responsible for running several applications as required by organisation. Organisations can produce their own applications by defining their network policies.

#### Southbound interface:

The interaction between forwarding devices and SDN controller takes place via. Southbound interface. Southbound interface allows any component to interact with components at lower-level hence the name south is given. Protocols present in southbound interface are:

- \* OpenFlow responsible for packet handling.

- \* OVSDB(Open Virtual Switch Database responsible for managing Open virtual switch implementations.
- \* NETCONF responsible for network configurations.
- \* SNMP(Simple Network Management Protocol) responsible for network management and network monitoring.

Northbound interface:

North Bound Interface in Software Defined Network (SDN) can be defined as an interface which provides communication between SDN controller and network applications running at application plane. SDN applications are ultimately responsible for directing the packets to desired destination with the help of North-Bound Interface. Certain organisations can design specific network applications using high level programming languages such as Java, Python, PHP, C sharp etc. These organisation specific network applications can communicate with SDN controller with the help of North-Bound Application Programming Interfaces(APIs). In-case of any abnormal events such as link failures SDN controller can communicate with applications via. NBI to latter instruct the data plane forwarding devices via. South-Bound Interface using OpenFlow protocol to handle the packets effectively. Northbound interface allows any component to interact with components at higher-level hence the name north is given.

- Software used:

- VirtualBox:

VirtualBox is a virtualization product and it is widely used due to its high performance. It can be easily downloadable as it is available as open source software. It can run on Windows, Linux and many other platforms and it can support various guest operating systems such as Windows, Linux, Solaris, etc. It provides the flexibility to load multiple guest operating systems under a single host operating system. The host operating system, the guest operating system and applications can interact with each other. With certain configurations different virtual machines could be made to interact with each other.

– Mininet:

Mininet [1] can be defined as a network emulator which helps in creating a virtual network of switches, hosts, controllers and links. It provides the flexibility of network analysis with the help of Mininet CLI commands which can be useful in learning, debugging, research and development, etc. It also helps in the setup of user defined custom topologies. The predefined topologies provided by Mininet are minimal topology, single topology, linear topology and tree topology. Mininet is fast as it takes only few seconds to setup custom or predefined topology. Using the Mininet `ovs-ofctl` command it is very easy to install new flow rules, delete ow rules and view ow rules of open flow switches. It is open source and it can easily run on virtual machines.

Mininet Command-Line Interface(CLI) commands can be very helpful for analysing any given network with pre-defined or custom topologies. Some of the Mininet CLI commands are listed below:

- \* `mininet¿help`: Displays Mininet CLI commands and how they can be used.
- \* `mininet¿nodes`: Displays all the available nodes present in the given topology.
- \* `mininet¿dump`: Displays information such as I.P., MAC addresses, etc. for all available nodes.
- \* `mininet¿h1 ping -c1 h2`: It can be used to test the connectivity or reachability between two hosts for one count as `c1` is mentioned.
- \* `mininet¿pingall`: It can be used to test the connectivity among all the present hosts.
- \* `mininet¿iperf`: It can be used to determine the bandwidth present in the channel.
- \* `mininet¿exit`: It is used to exit from the Mininet command line interface.
- \* `sudo mn -c`: It is used to perform cleanup process. It can be helpful if due to some circumstances Mininet crashes.

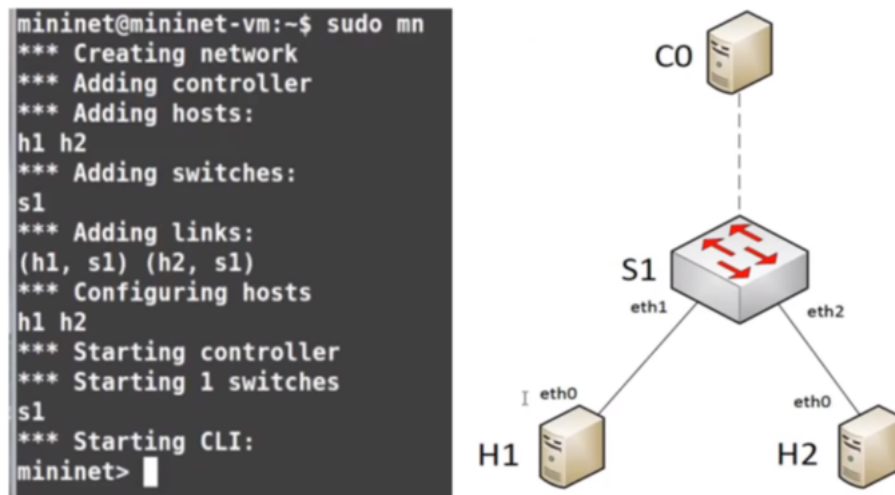


Figure 1: Minimal topology in Mininet

\* `sudo mn`: It is used to establish minimal topology network with 1 switch and 2 hosts.

\* `sudo mn -topo=single, n`: It can be used to establish single topology which contains 1 switch and n hosts. Single topology is shown in below figure:

\* `sudo mn -topo=linear, n`: It can be used to establish linear topology which contains n switches and n hosts. Linear topology is shown in below figure:

The above mininet commands are examples of pre-defined topology. An example to define any custom topology is shown in below figure:

The command to establish this topology and connect with remote controller is:

```

sudo mn -custom mininet/custom/topo-4sw-6host.py -topo mytopo -controller=remote
-mac

```

Here, `topo-4sw-6host.py` is the name of the file where the topology is

```

mininet@mininet-vm:~$ sudo mn --topo=single,4
*** 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
*** Starting 1 switches
s1
*** Starting CLI:
mininet>

```

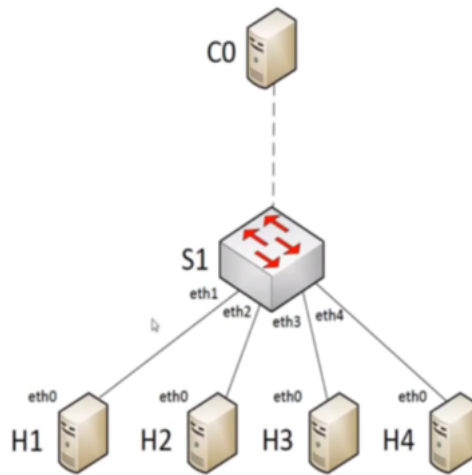


Figure 2: Single topology in Mininet

defined and programmed.

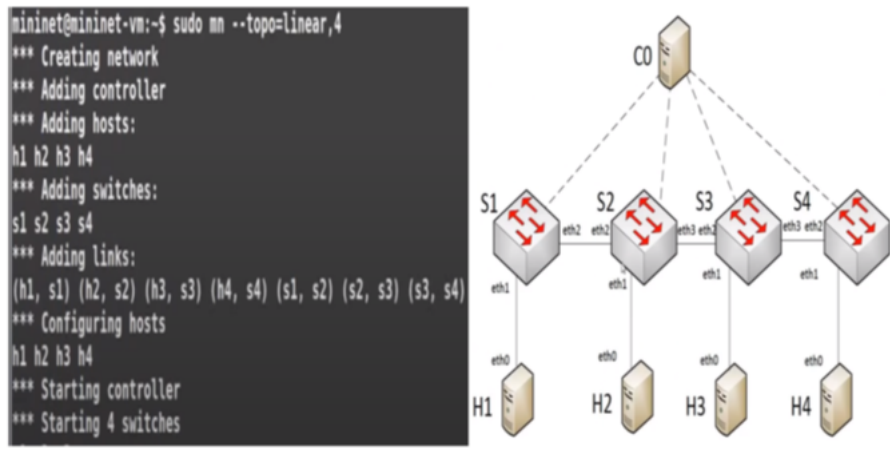


Figure 3: Linear topology in Mininet

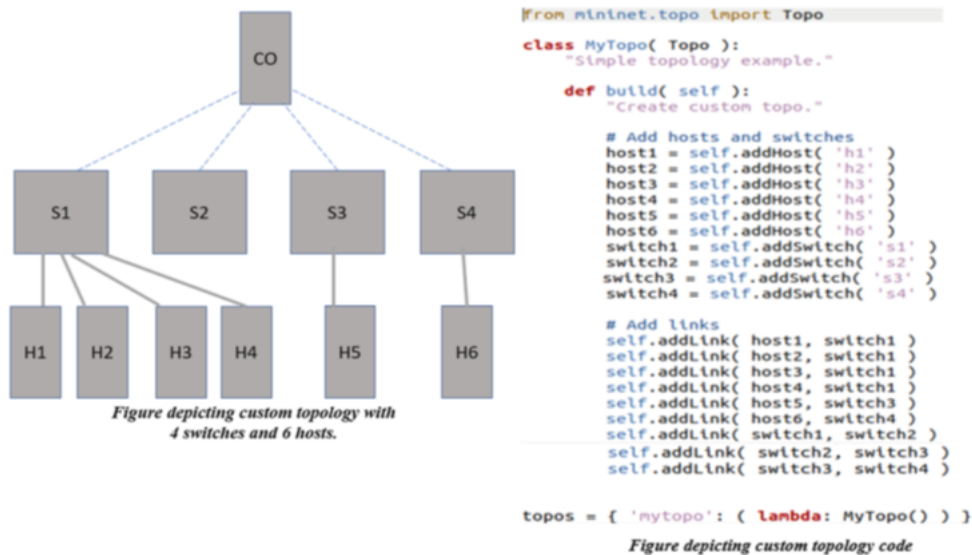


Figure 4: Custom topology in Mininet



## Chapter 3

### Literature Review

Intent Driven Networking(IDN) [8, 15, 17] is a network which is intelligent and can achieve a network state which is specified by the intent of the operators. Operators intent refers to the ability to express freely with highest level of abstraction which is parsed and finally converted into configuration settings. It's an advance version of Software Defined Networking(SDN) where no manual configurations are performed. In-order to process the intents specified by the operators Natural Language Processing(NLP) tools can be used and then parsing could be performed easily [7]. The current work on IDN [4, 13, 21, 22] suggests that the IDN model could be designed either using any Software Defined Networking(SDN) controller or without using them. Some examples of SDN controllers are POX [2, 14], Floodlight, RYU [3, 9], ONOS, Open daylight, etc. The ultimate aim of IDN model [5, 6, 12, 19, 20] is to process any intent specified by the operator. Organisations such as Internet Engineering Task Force (IETF) has conducted many researches on IDN and working on automated network management.

# Chapter 4

## Proposed Work

### 4.1 Introduction

The complete architecture of the proposed model is shown in below figure. As shown in the figure, user specifies the intent initially. This intent is then passed through NLP processing pipeline. The major steps involved in the pipeline are tokenization, case conversion, text filtration and lemmatization.

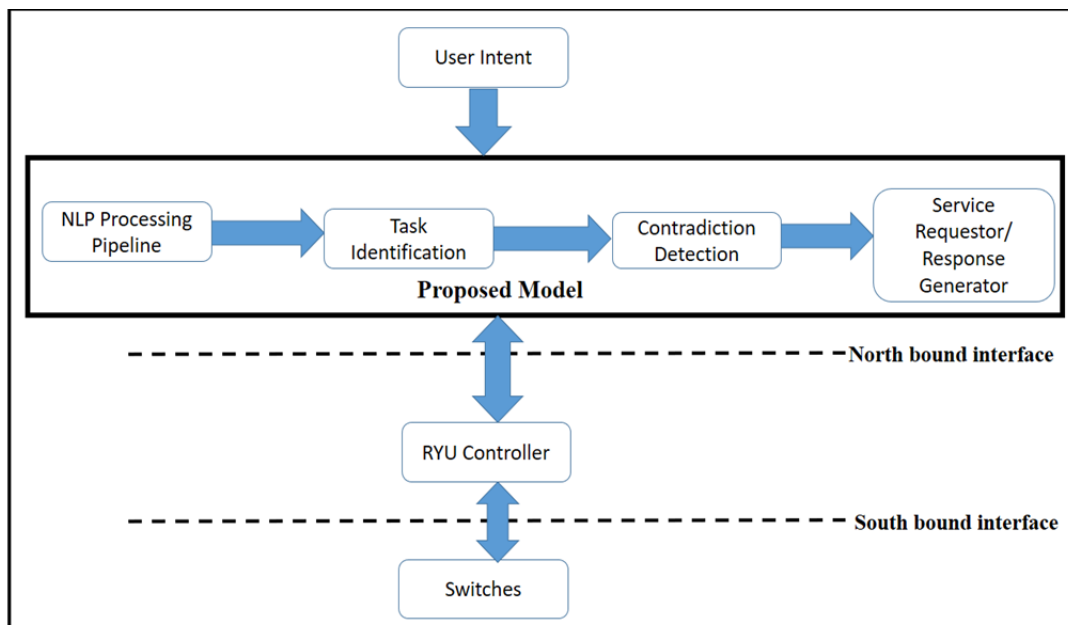


Figure 5: RYU based IDN model architecture

After the intent passes through the above mentioned steps it is now the time to parse the intent and finally convert it into a form that could be understood by the RYU controller. The major steps involved in parsing are task identification phase and JSON format composition phase which are explained briefly in implementation details section. The output from the NLP processing pipeline goes to the task identification phase. In this phase, network task is identified with help of certain keywords which are defined manually. With the help of these keywords, 45 different intents can be processed easily. After successful identification of task, we now move on to the next phase i.e. contradiction detection. Contradiction detection phase is a conditional phase and it is not involved in every intent processing. The condition here is the involvement of certain keywords such as [block, host], [enable, host], etc. If these keywords are involved, only then the contradiction detection phase will start else this phase is bypassed and we move directly onto the next phase i.e. JSON format composition. In the JSON format composition phase appropriate message is constructed in JSON format w.r.t. the intent specified by network operators. The constructed JSON format is then send to the RYU controller as a request and the response of the request is sent to operator on the terminal window.

## 4.2 Implementation details

- NLP PROCESSING PIPELINE AND USE OF NLTK PACKAGE:

We have made a python application that reads the intents of operators and then passes them over the NLP processing pipeline. For accomplishing the NLP part of our model NLTK package has been used in this application. NLP helps to bridge the gap between humans and machines by providing an efficient and convenient way of interaction in natural language.

Any intent specified by operators has to pass through the pipeline as shown in figure 2. The need for NLP processing pipeline is to remove any unwanted elements from the intent so that further processing of intent is easier. The steps involved in pipeline are following:

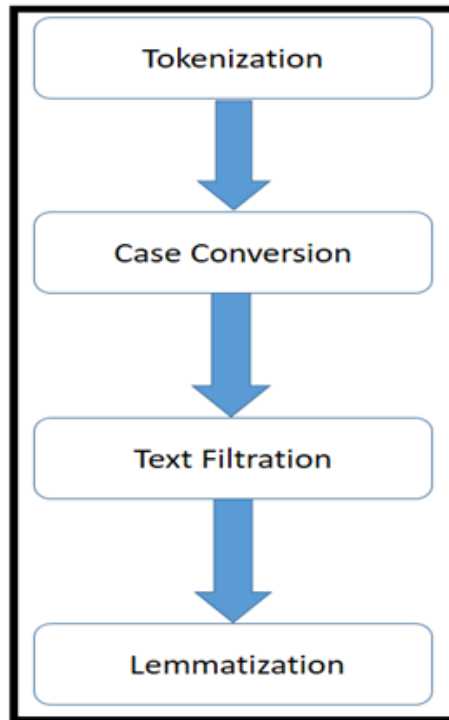


Figure 6: Natural Language Processing(NLP) pipeline

– Tokenization:

Tokenization can be defined as the process of splitting a sentence, paragraph or an entire text data into smaller segments such as words. Each words, numbers or punctuation marks are called tokens. Tokens are created with the help of word boundaries. Word boundaries is the boundary between the ending point of word and starting point of next word. The generation of different tokens is an essential step for the process of Lemmatization which is performed at a later point.

In the proposed model tokenization helps to separate out the words of the intent specified. The intent is the overall sentence that is to be processed. For an intent specified as “Get the number of Switches present in the Topology?”. After the intent passes through the tokenization phase of NLP processing pipeline the output as shown in below figure is [get, the, number, of, Switches, present, in, the, Topology,].



Figure 7: Intent after passing through tozenization phase

For another intent specified as “Please Block a Host h1”. When this intent passes through tokenization then the output as shown in below figure is [Please, Block, a, Host, h1].

There can be different methods that can be used to perform tokenization but



Figure 8: Intent after passing through tozenization phase

in the proposed model we have used the method of Natural Language Toolkit (NLTK) package to perform tokenization. The method named `word.tokenize()` in the package is used to perform word tokenization which separates the string at each space and produces the output as list of strings. In case if sentence tokenization is required which means separating the string at each full stop(.) then the method `sent.tokenize()` can be used.

- Case conversion:

In this process each token that is generated previously is converted to its lower case so that the processing becomes completely case insensitive. Depending upon the requirement the tokens can also be converted to upper case. This process is effective in eliminating any errors occurring due to case mismatch. The input and the output is shown in below figure:

- Text Filtration: This process is responsible for removing punctuation marks, special characters and stop words from tokens. There are fourteen punctuation

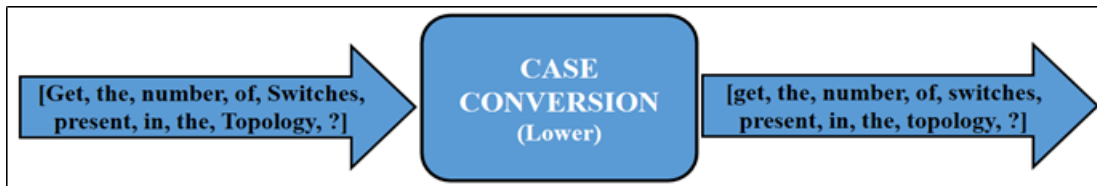


Figure 9: Figure depicting case conversion phase of NLP processing pipeline

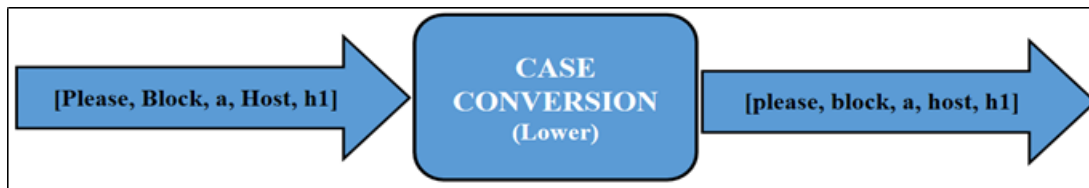


Figure 10: Figure depicting case conversion phase of NLP processing pipeline

marks in English such as full stop(.), question mark(?), comma (,), exclamation mark(!), colon (:), Semicolon (;), etc. Some of the special characters includes hash(#), dollar sign(\$), ampersand(), percent(%), etc. Stop words can be defined as most commonly occurring words such as a, an, the, in, etc. These words do not add any special meaning to the sentence.

The elements such as punctuation marks, special characters and stop words are not needed to be processed as per the requirements and hence are removed. This is the most important step in the NLP processing pipeline and it is mainly responsible to reduce the overall size of the intent so that it could be processed much faster at later stages.

Two examples of input and output from this phase is shown below:



Figure 11: Figure depicting input/output of text filtration phase



Figure 12: Figure depicting input/output of text filtration phase

– Lemmatization:

The output of lemmatization is known as ‘lemma’ which is a root word. Unlike stemming where the root word can be valid or invalid in lemmatization the root word is always valid as it refers to the context of the sentence. Python NLTK package provides WordNet Lemmatizer which is very useful in generating the root word of a given word.

In the proposed model use of lemmatization is responsible to convert all the tokens or words of the intent to its root form so that further there is no difficulty in the identification of network task. After completion of the above steps the tokens are again re-joined together and we observe an obvious reduction in the size of the intent as compared to its size originally. The reduced intent can be easily identified for specific network task details to which is discussed in the next section i.e. task identification phase.

Input and output of lemmatization phase of NLP processing pipeline is shown below:



Figure 13: Figure depicting lemmatization phase of NLP pipeline

• **TASK IDENTIFICATION PHASE:**

After the output of the NLP pipeline it is now the time to identify the network task. As mentioned earlier, in order of identify a network tasks many different sets



Figure 14: Figure depicting lemmatization phase of NLP pipeline

of keywords have been defined manually. As the intents are expressed naturally therefore there are many different ways of expressing the same intent. In-order to solve this every set of keywords is defined with synonyms. Below table shows list of different intents that could be processed by the proposed model:

The behaviour of task identification phase can be seen in below figure:

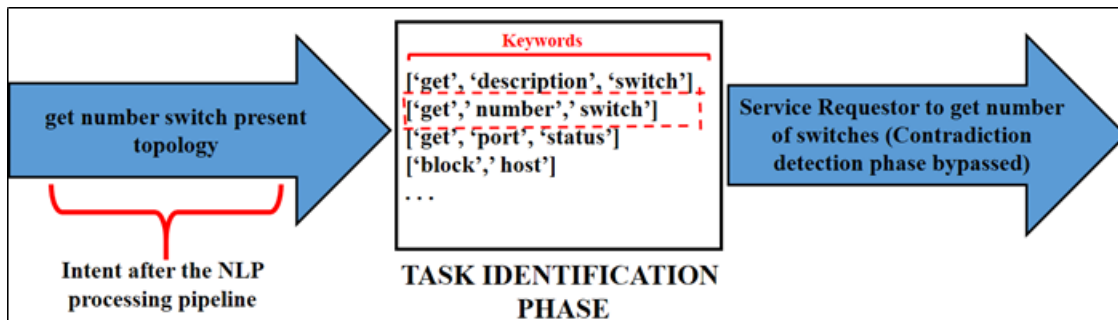


Figure 15: Figure depicting working of task identification phase and bypassing of contradiction detection phase

The above figure shows the working of task identification phase. The initial intent specified as “Get the number of switches present in the topology?” after passing through NLP processing pipeline becomes “get number switch present topology”. The output from NLP processing pipeline acts as an input to the task identification phase. The task identification phase as mentioned above consists of manually defined sets of keyword which is eventually responsible for the recognition of task to be performed. In this case the keywords responsible for task recognition are [‘get’, ‘number’, ‘switch’] as shown by dashed red colour in the above figure. As there



S.NO.	Network Intents	Keywords	REST API method
1.	Get the number of switches present in the topology?	get, number, switch	GET
2.	Get the description of switch s1?	get, description, switch	GET
3.	Get detailed flows installed in switch s1?	get, detailed, flow, switch	GET
4.	Get all the flows installed in switch s1 with rule id?	get, flow, switch, rule	GET
5.	Please Block a Host h1	block, host	POST
6.	What are the number of switches present in the topology?	what, number, switch7	GET
7.	Deny traffic between host h2 and host h3 block,	deny,host	POST
8.	Enable host h1	enable, host	POST
9.	Get the details of switch s1?	get, details, switch	GET
10.	Allow the traffic between host h2 and host h3	allow, host	POST
11.	Block switch s1	block, switch	PUT
12.	Activate switch s1	activate, switch	PUT
13.	How many switches are present in the topology?	how, many, switch.	GET
14.	Get the overall status of all the switches?	overall, status, switch	GET
15.	Delete flows from switch	delete	DELETE
16.	Get port status of switch s1	get, port, status, switch	GET
17.	Get aggregate flows of switch s1	get, aggregate, flow, switch	GET
18.	Show the description of switch s3	show, description, switch	GET
19.	Show the details of switch s4	show, detail, switch	GET
20.	Display the details of switch s5	display, detail, switch	GET
21.	Display the description of switch s6	display, description, switch	GET
22.	What are the details of switch s7?	what, details, switch	GET
23.	Show the detailed flow entries of switch s2	show, detailed, flow, switch	GET
24.	Display the detailed flows of switch s4	display, detailed, flow, switch	GET
25.	Deny the traffic between host h2 and host h5	deny, host	POST
26.	Disable host h3	disable, host	POST
27.	Display the aggregate flows of switch s4	display, aggregate, flow, switch	GET
28.	Show the flow entries of switch s3 with rule id	show, flow, switch, rule	GET
29.	Deactivate the traffic between host h6 and host h7	deactivate, host	POST
30.	Allow the communication between host h2 and host h5	allow, host	POST
31.	Activate host h3 to communicate	activate, host	POST
32.	Forward the traffic between host h6 and host h7	forward, host	POST
33.	Get the overall status of switches present in the topology	get, overall, status, switch	POST
34.	Disable switch s4	disable, switch	PUT
35.	Delete a flow rule from switch	delete	DELETE
36.	Display the flows of switch s2 with rule id	display, flow, switch, rule	GET
37.	Remove a flow rule from switch	remove	DELETE
38.	What is the port status of switch s2?	what, port, status, switch	GET
39.	What are the aggregate flows of switch s2	what, aggregate, flow, switch	GET
40.	Show the port status of switch s3	show, port, status, switch	GET
41.	Show the aggregate flows of switch s3	show, aggregate, flow, switch	GET
42.	Display the port status of switch s4	display, port, status, switch	GET

Table 1: Different user's intent

is no keywords which are concerned with blocking or enabling of hosts therefore the contradiction detection phase is bypassed and the task identification phase calls for service requestor to handle future operations concerned with getting number of switches present in the topology.

Consider another example of task identification phase shown in below figure when the next stage i.e. contradiction detection phase is not bypassed.

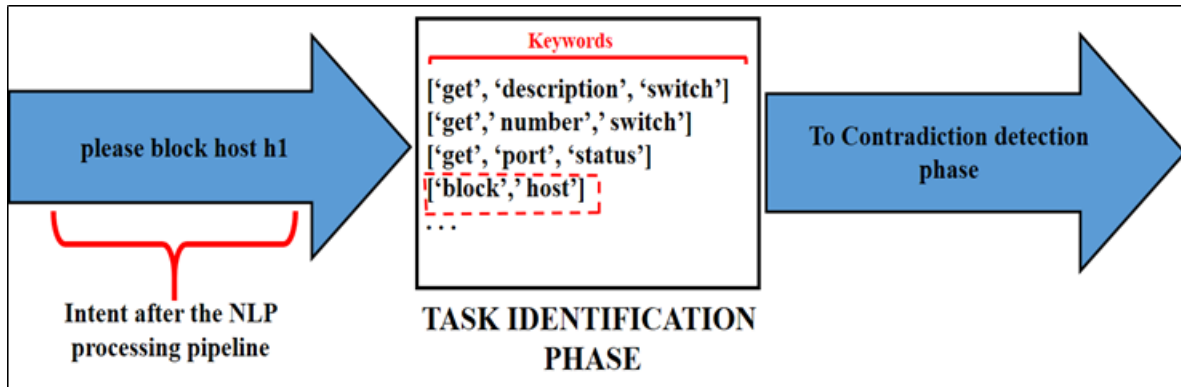


Figure 16: Figure depicting working of task identification phase and next phase as contradiction detection

With the reference to above figure the initial intent undergoing the NLP processing pipeline is “Please Block a Host h1”. After passing through the pipeline the intent becomes “please block host h1” which is given as input to the task identification phase as shown in figure above. Task identification phase with help of keywords [‘block’, ‘host’] identifies that blocking of certain host has to be performed and calls for contradiction detection to handle further process.

#### • CONTRADICTION DETECTION TECHNIQUE AND RESULTS:

As mentioned earlier contradiction detection technique will allow the network operators to detect contradictions with respect to previously deployed intents. In order to accomplish this task machine learning algorithm have been used and for detecting contradictions mirroring is used.

To train and test our contradiction detection model we require a dataset. No such dataset is available and therefore we have composed a dataset with combination

of 0's and 1's. In the dataset 0's indicates either no previous intent has been deployed for host/switch or host/switch is not enabled/blocked and 1's indicates that the host/switch is either enabled or blocked. In the dataset there are a total of 21 columns in which the first ten columns is the block status of host and next ten columns is the enable status of host finally the last column is the result whether contradiction has been detected or not. We have worked with four different machine learning models i.e. Logistic regression, Naïve Bayes, Random Forest Classifier and Decision Tree Classifier.

In order to train and test the models we have used the standard 75%-25% train-test dataset split. In order to predict contradictions, we are composing a list with 21 indexes in real time. This list is used with predict methods of different models.

In-order to track the current status of hosts we have defined four arrays. Two arrays are one dimensional which are responsible to track block and enable status of one host and the other two arrays are two dimensional which are responsible to track block and enable status of two hosts. After a network task has been identified for blocking or enabling of host it first moves to the contradiction detection phase.

The important four steps involved in the contradiction detection phase are following:

- As a first step number of host involved is checked which could be either one or two.
- In the second step depending upon the request to block/enable host respected array is modified to hold the current status of host. If there is only one host involved then all the modifications will be done on one dimensional array whereas if two hosts are involved then two dimensional array is involved.
- In third step list is composed with 20 indexes with first 10 indexes are block status of host and next 10 indexes are enable status of host. The composed list is used to predict contradiction detection.
- In the fourth step the composed list is sent to the machine learning model for contradiction detection.

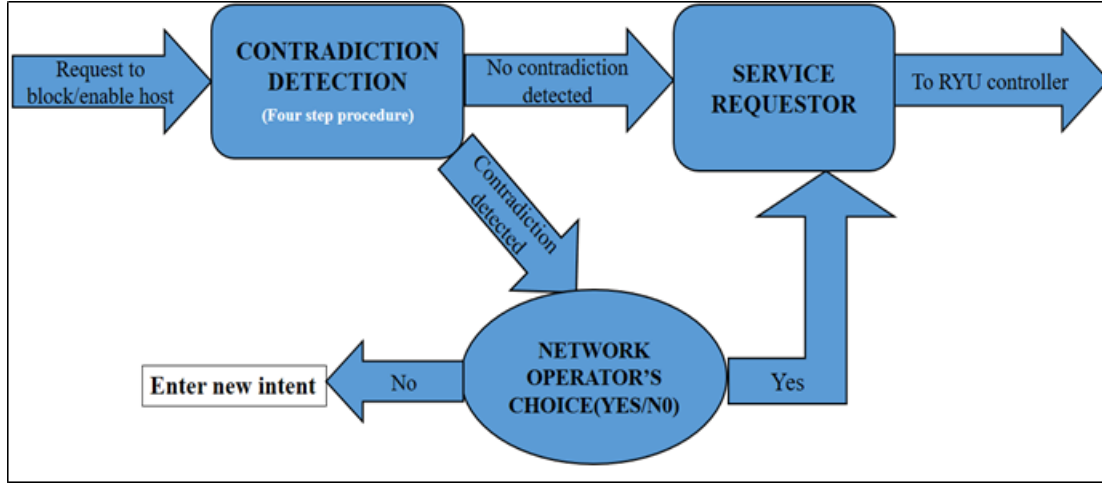


Figure 17: Figure depicting working of contradiction phase

After sending the list for contradiction detection there can be two possible outcomes. First outcome is that no contradiction could be detected and automatically service requestor is called which composes a data in JSON format and sends to RYU controller for flow installation. If contradiction is detected then the operator is prompted with a notification that contradiction has been detected and does he wants to proceed further and install the flow or not. If the operator chooses to proceed then service requestor is called which composes a data in JSON format and sends to RYU controller for flow installation process. If the operator chooses not to proceed further then service requestor phase is bypassed and the modified arrays are reverted back to hold the current status of hosts.

Consider three examples of list composition in real time as below:

- First example as list containing elements such as: 0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0 in this case while composing the list in real time contradiction will be detected as 1st index and 11th index both are 1.
- Consider second example as: 1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 in this no contradiction will be detected and our model assumes that there is a certain request to block host h1 as 1st index is 1.

- Consider third example as: 1,1,1,1,0,0,0,0,0,1,1,1,0,0,0,0,0,0 in this case again no contradiction will be detected as this is the case of error. This is the case of error because our model is trained only to predict contradictions with one or at maximum two hosts therefore if contradiction is detected for more than two hosts then it becomes an error and our model predicts that no contradiction has been detected. As we are including cases of errors as well in the dataset hence its size very complex.

Consider four examples of intents as block host h1, enable host h1, block host h2 and host h3 and finally enable host h2 and host h3. These intents after passing through NLP processing pipeline and task identification phase will finally pass through the contradiction detection phase. The detailed description of every step is explained with four different intents as shown below:

– **INTENT: Deny host h1**

- \* Only one host is involved here hence one dimensional arrays which keeps track of block and enable status of hosts are used.
- \* This step involves modification of one dimensional arrays. Initially the status of block and enable arrays are:

h1	h2	h3	h4	h5	h6	h7	h8	h9	h10
0	0	0	0	0	0	0	0	0	0

Table 2: Initial block status of hosts

h1	h2	h3	h4	h5	h6	h7	h8	h9	h10
0	0	0	0	0	0	0	0	0	0

Table 3: Initial enable status of hosts

After the request to block host h1. Block array at index H1 is modified to 1 as shown below:

h1	h2	h3	h4	h5	h6	h7	h8	h9	h10
1	0	0	0	0	0	0	0	0	0

Table 4: Block status of hosts after modification

- \* Now the list with 20 indexes is composed taking all of the block array and enable array. The contents of the list are: 1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.
- \* The composed list of 20 indexes is sent to the machine learning model for contradiction detection.

In this case no contradiction will be detected and service requestor is called automatically to compose the data in JSON format and send to RYU controller for flow installation process.

– **INTENT: Allow host h1**

- \* Here again only one host is involved and hence one dimensional arrays will be used.
- \* Initial block and enable status of host from arrays are:

h1	h2	h3	h4	h5	h6	h7	h8	h9	h10
1	0	0	0	0	0	0	0	0	0

Table 5: Current block status of hosts

h1	h2	h3	h4	h5	h6	h7	h8	h9	h10
0	0	0	0	0	0	0	0	0	0

Table 6: Current enable status of hosts

After the request to enable host h1. Enable array indexed at H1 is modified as:

h1	h2	h3	h4	h5	h6	h7	h8	h9	h10
1	0	0	0	0	0	0	0	0	0

Table 7: Enable status of hosts after modification

- \* Now the list with 20 indexes are composed as: 1,0,0,0,0,0,0,0,0,0, 1,0,0,0,0,0,0,0,0,0
- \* This list is then sent to the machine learning model for contradiction detection.

We observe that contradiction has been detected in this case and operator is given a choice that does he wants to proceed and install the flow or roll back from the current state.

– **INTENT: Deactivate the traffic between host h2 and host h3.**

- \* Here two hosts are involved and hence two dimensional arrays are used.
- \* Initially block and enable status of hosts are:

	h1	h2	h3	h4	h5	h6	h7	h8	h9	h10
h1	0	0	0	0	0	0	0	0	0	0
h2	0	0	0	0	0	0	0	0	0	0
h3	0	0	0	0	0	0	0	0	0	0
h4	0	0	0	0	0	0	0	0	0	0
h5	0	0	0	0	0	0	0	0	0	0
h6	0	0	0	0	0	0	0	0	0	0
h7	0	0	0	0	0	0	0	0	0	0
h8	0	0	0	0	0	0	0	0	0	0
h9	0	0	0	0	0	0	0	0	0	0
h10	0	0	0	0	0	0	0	0	0	0

Table 8: Initial block status of hosts

After the request to block host h2 and host h3. Block array at indexes (2,3) and (3,2) will be modified to 1 and enable array will remain unchanged.

- \* Now two list with 20 indexes are composed for strict verification of contradiction detection. Overall contradiction will be detected only if the





\* In the specified intent two hosts are involved and therefore both the 2D arrays will be involved.

\* The status of both 2D arrays as of now are shown below:

	h1	h2	h3	h4	h5	h6	h7	h8	h9	h10
h1	0	0	0	0	0	0	0	0	0	0
h2	0	0	1	0	0	0	0	0	0	0
h3	0	1	0	0	0	0	0	0	0	0
h4	0	0	0	0	0	0	0	0	0	0
h5	0	0	0	0	0	0	0	0	0	0
h6	0	0	0	0	0	0	0	0	0	0
h7	0	0	0	0	0	0	0	0	0	0
h8	0	0	0	0	0	0	0	0	0	0
h9	0	0	0	0	0	0	0	0	0	0
h10	0	0	0	0	0	0	0	0	0	0

Table 11: Current block status of hosts

	h1	h2	h3	h4	h5	h6	h7	h8	h9	h10
h1	0	0	0	0	0	0	0	0	0	0
h2	0	0	0	0	0	0	0	0	0	0
h3	0	0	0	0	0	0	0	0	0	0
h4	0	0	0	0	0	0	0	0	0	0
h5	0	0	0	0	0	0	0	0	0	0
h6	0	0	0	0	0	0	0	0	0	0
h7	0	0	0	0	0	0	0	0	0	0
h8	0	0	0	0	0	0	0	0	0	0
h9	0	0	0	0	0	0	0	0	0	0
h10	0	0	0	0	0	0	0	0	0	0

Table 12: Current enable status of hosts

After the request to enable host h2 and host h3 the enable array will be modified at index positions (2,3) and (3,2) to 1.

\* The two lists after composition are:

List1: 0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0.

List2: 0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0.

The composed list is now ready to be sent to machine learning model.

	h1	h2	h3	h4	h5	h6	h7	h8	h9	h10
h1	0	0	0	0	0	0	0	0	0	0
h2	0	0	1	0	0	0	0	0	0	0
h3	0	1	0	0	0	0	0	0	0	0
h4	0	0	0	0	0	0	0	0	0	0
h5	0	0	0	0	0	0	0	0	0	0
h6	0	0	0	0	0	0	0	0	0	0
h7	0	0	0	0	0	0	0	0	0	0
h8	0	0	0	0	0	0	0	0	0	0
h9	0	0	0	0	0	0	0	0	0	0
h10	0	0	0	0	0	0	0	0	0	0

Table 13: Enable status of hosts after modification

- \* When both these lists are sent to the machine learning model for contradiction detection one by one then contradiction is detected in both the lists and hence there is an overall contradiction detection. Contradiction is detected as there was a previous intent specifying blocking of host h2 and host h3 and then request to enable these hosts are specified.

After the contradiction is detected the operator as usual is given two choices whether he wants to proceed with installation of new flow or rollback and take no actions.

As an initial step we are able to detect contradictions among 10 hosts and 10 switches.

Machine Learning models used:

Machine learning refers to the field of study that is concerned with certain algorithms which are trained and later tested. There can be many different ways of performing contradiction detection but we have incorporated the concepts of machine learning in-order to provide better accuracy and faster responses.

For performing contradiction detection four machine learning models are used and they are Logistic Regression, Naïve Bayes, Random Forest Classifier and Decision Tree Classifier. These algorithms are used as binary classification. Binary classification is a type of classification which have two states. One state can be called as normal state and other state can be called as abnormal state.

In the proposed model if contradiction is detected then it is an abnormal state and it is assigned class label 1 whereas if contradiction is not detected then it is normal state and it is assigned class label 0. Different machine learning models that has been used in the proposed model is discussed below and results can be viewed later under the section analysis and results.

- Logistic Regression:

Logistic Regression is a machine learning algorithm that can be used to solve problems based on classification which can have outcomes such as two states, three states and three or more states. There are three types of logistic regression model i.e. binary/binomial, multinomial and ordinal. In binary, the dependent variable can have only two states. In multinomial, the dependent variable can have three states and in ordinal the dependent variable can have three or more states. Therefore, it is a good choice to make use of logistic regression for contradiction detection.

- Naïve Bayes:

Naïve Bayes is a machine learning algorithm which is also a classification technique and it is based on Bayes theorem. As usual the python library, Scikit learn is used to build the Naïve Bayes model. The three Naïve Bayes models provided by the Scikit learn library are Gaussian Naïve Bayes, Multinomial Naïve Bayes and Bernoulli Naïve Bayes. Out of these three models we are using Gaussian Naïve Bayes model for contradiction detection as it is the simplest form and easier to implement.

- Random Forest Classifier:

Random forest is a machine learning algorithm that can be used to solve problems based on classification and regression. It is mainly used to solve classification problems. Random forest creates many decision trees and gets the predictions from each of the decision tree and selects the best solution by voting. It works good even if there is large range of data items.

- Decision Tree Classifier:

They can also be used for classification and regression problems. They also work well with large range of data items. In the proposed model the results of the decision tree classifier is best as compared with rest of the machine learning models.

- **SERVICE REQUESTOR AND RESPONSE GENERATOR:**

After the completion of above phases, we are finally assured what task has to be performed by the RYU controller. This is the phase where parsing takes place and the intent specified in natural language is finally converted to a form that is understood by the RYU controller. For this phase corresponding to each network task a service handler function is defined that will be called and it will be responsible for creating a request in desired format and then sending it to the controller as a request for the service. After sending the request for a service, controller generates responses as per the requirements specified in the intent. The response of the controller is unordered and difficult to read. Therefore, our function modifies the response of the controller and then produces the final output on the screen which is ordered and readable.

All the GET and PUT requests that is sent to the RYU controller is directly sent to a specified URL without requiring the need to compose the data in JSON format. The requests that are send as POST and DELETE are sent to RYU on a specified URL along-with the data in JSON format. There are two types of POST request that could be sent to RYU controller and these are request to block host and request to enable host. There is one DELETE request that could be sent to controller for deletion of flow rules installed in switches. Consider an example of blocking two hosts h2 and h3, for this the data in JSON format that will be sent is:

```
{  
  "priority": 101,  
  "nw_proto": ICMP,  
  "nw_src": 10.0.0.2,  
  "nw_dst": 10.0.0.3,  
  "actions": DENY
```

```
}
```

The URL on which this data will be sent is: <http://localhost:8080/firewall/rules/all>

Consider another example of enabling two hosts h1 and h4, for this the data in JSON format that will be sent is:

```
{  
  "priority": 102,  
  "nw_proto": ICMP,  
  "nw_src": 10.0.0.1,  
  "nw_dst": 10.0.0.4,  
  "actions": ALLOW  
}
```

Along with this another data in JSON format that will be sent is:

```
{  
  "priority": 102,  
  "nw_proto": ICMP,  
  "nw_src": 10.0.0.4,  
  "nw_dst": 10.0.0.1,  
  "actions": ALLOW  
}
```

The URL on which these both data in JSON format will be sent is same as above i.e. <http://localhost:8080/firewall/rules/all>

In-order to get the I.P. address of hosts we have made a mapping of host to I.P. address and all the I.P. addresses are fetched from this mapping. Considering the delete operation as third example where the data will be sent as JSON format. Suppose there is request to delete flow rule with id 3 from switch s1. For this the data

in JSON format will be:

```
{  
  "rule_id": 3  
}
```

The delete request will be sent on URL:

<http://localhost:8080/firewall/rules/0000000000000001>.

In this URL the switch id must always be in 16-digit hexadecimal format.

The different GET requests and their respected URL are given below:

- Getting the number of switches present in the topology:  
<http://localhost:8080/stats/switches>
- Getting the hardware and software description of switch:  
<http://localhost:8080/stats/desc/swid>  
swid stands for switch id.
- Getting detailed flows of switch:  
<http://localhost:8080/stats/flow/swid>
- Getting flows installed in switches with rule id's:  
<http://localhost:8080/firewall/rules/SWID>  
Here, the SWID also stands switch id but it must be in 16-digit hexadecimal format.
- Getting the status of all switches:  
<http://localhost:8080/firewall/module/status>
- Getting the port status of switch:  
<http://localhost:8080/stats/port/swid>
- Getting aggregate flows of switch:  
<http://localhost:8080/stats/aggregateflow/swid>

After sending these requests to RYU controller we get response from the controller

which is processed and arranged in ordered manner and then provided to the operator.

# Chapter 5

## Experimental setup

### 5.1 Installation

In order to start working with proposed model we need to complete the installation process. We need to install the VirtualBox which supports the creation and management of guest virtual machines running Windows, Linux, etc. We then need to download the Ubuntu Operating System and install it on the VirtualBox. After completing the installation of Ubuntu O.S. in VirtualBox, Mininet [1] and RYU controller [3] needs to be installed. RYU controller is a python based controller and requires Ubuntu O.S. for its installation. We can then run some commands of Mininet and RYU controller in-order to verify that the installation was successful. In order to deal with the Natural Language Processing(NLP) part of my model I have installed NLTK package. Intent specified by network admin. are passed through NLP processing pipeline and finally we get a reduced form of intent or a subset of intent that has been specified. This process is called cleaning up of text. In order to accomplish the use of machine learning algorithms, scikit-learn machine learning library has been installed. For reading of the dataset pandas library has been used.



## 5.2 Environment Setup

After completing the overall installation process, in order to start working with the model and specify intents certain steps must be followed to setup proper environment. Setting up proper environment will mean providing knowledge to our model such as network topology. First step in setting up the environment is the need to start the RYU controller so that any request either in form of GET/POST/PUT/DELETE are accepted by the controller via. the REST API's and hence network can be managed effectively. Second step is the need to start the Mininet with any topology and establishing connection between Mininet and RYU controller via. command `--controller=remote`, which signals Mininet to connect to a controller which is running remotely. After performing the steps mentioned, we are finally ready to specify the intents to be processed by RYU controller.

# Chapter 6

## Analysis and Results

### 6.1 Machine learning model results

S.No.	Model Name	Accuracy Score	Precision Score	F1 Score	Recall Score
1.	Logistic Regression	0.809	0.650	0.556	0.553
2.	Naïve Bayes	0.809	0.650	0.556	0.553
3.	Random Forest Classifier	0.977	0.983	0.960	0.941
4.	Decision Tree Classifier	0.999	0.999	0.998	0.998

Table 14: Evaluated results for contradiction detection

The above table shows the evaluation results of different machine learning models. The machine learning models are judged on the basis of accuracy score, precision score, f1 score and recall score. On analysing these results we come the fact that decision tree classifier is the best model for contradiction detection.

After analysing the machine learning models with four parameters mentioned above these models are tested with ROC curve.

The ROC curve for different models are shown below:

- Logistic Regression:

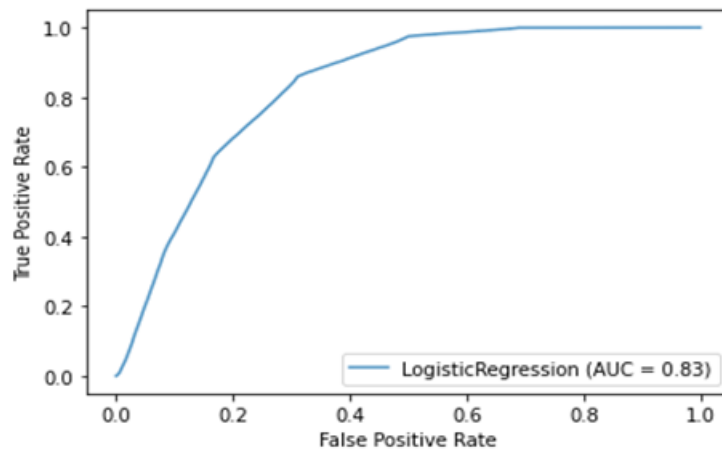


Figure 18: ROC curve for Logistic Regression model

- Random Forest Classifier:

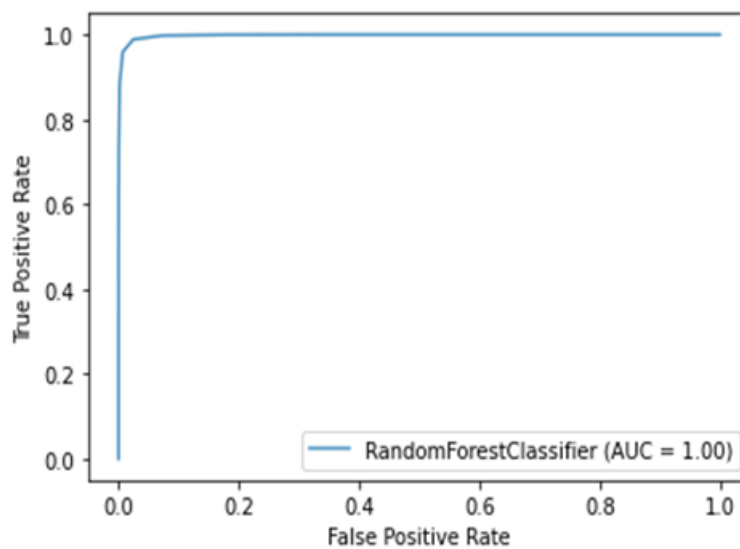


Figure 19: ROC curve for Random Forest Classifier model

- Naive Bayes:

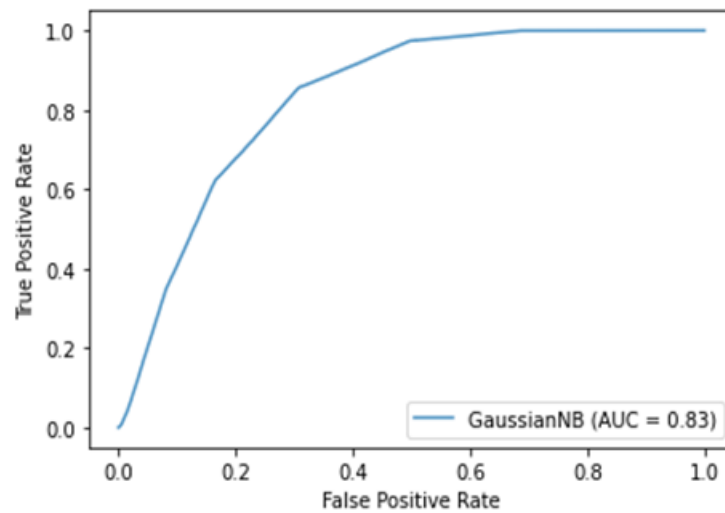


Figure 20: ROC curve for Naive Bayes model

- Decision Tree Classifier:

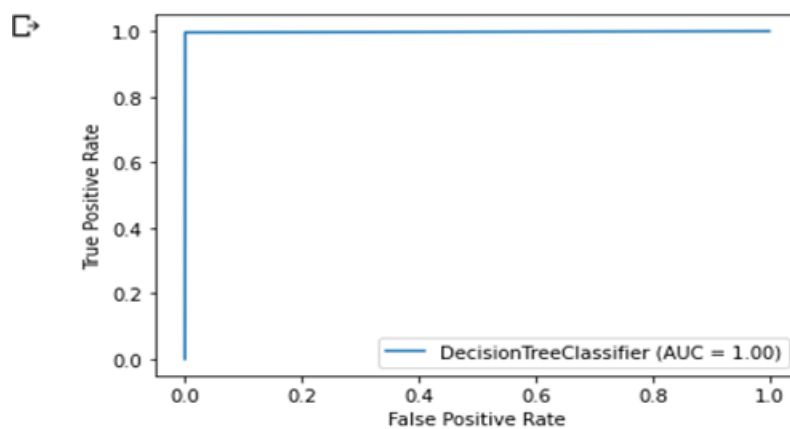


Figure 21: ROC curve for Decision Tree Classifier model

On observing the above ROC curve for different models we found that decision tree classifier has the best area under curve. The area under the curve for logistic regression

and Naïve Bayes machine learning models are less and it should not be preferred for contradiction detection.

## 6.2 Model Interaction

This section deals with interaction with proposed model. The response of different intents can be easily seen in this section. As an example linear topology with 7 switches and 7 hosts as shown in the figure below is taken for viewing the response after processing of user's intents.

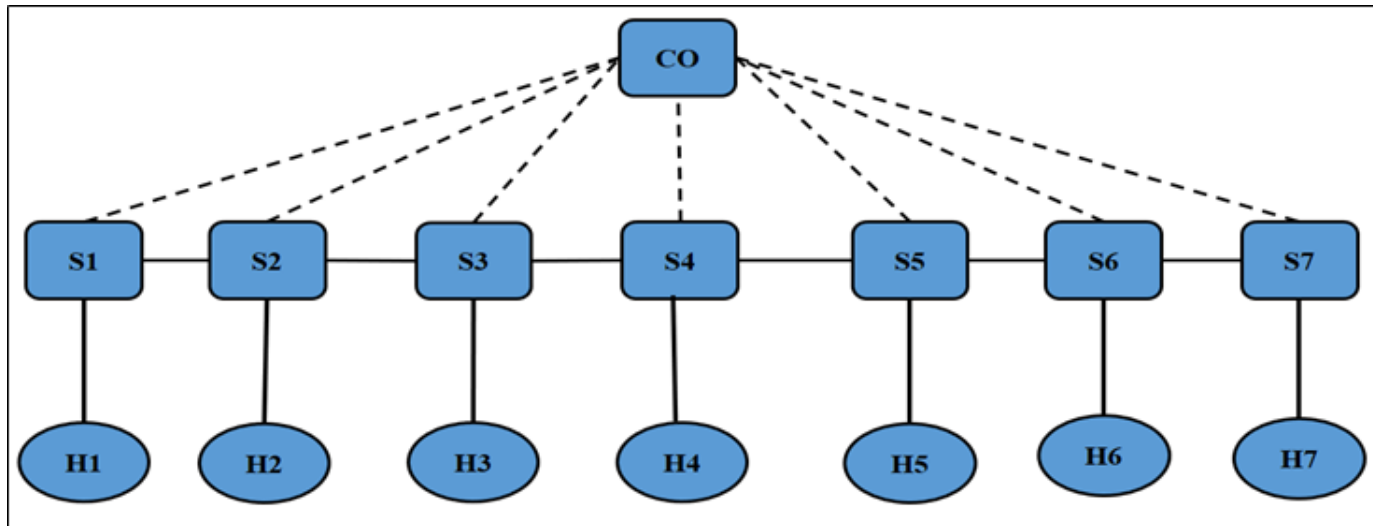


Figure 22: Figure depicting linear topology

- Intent to know the number of switches present in the topology.

In this response we are getting to know about the response code, switch id's and number of switches. The response code is 200 which is an indication that the intent has been successfully processed. There are 7 switches present in the topology therefore the switch id's as indicated is from 1 to 7.

- Intent to know the hardware and software description of switch.

```

Please enter new intent or enter exit
How many switches are present in the topology?

Getting you the list of all the switch IDs which are connected to the controller
RESPONSE CODE      : 200
Switch IDs present are: [7, 6, 4, 1, 5, 2, 3]
Number of switches are: 7

```

Figure 23: Figure depicting an intent to know the number of switches present in the topology

```

Please enter new intent or enter exit
What are the details of switch s7?

Getting you the description of switch with ID:7
[RESPONSE CODE] 200

Manufacturer description ::Nicira, Inc.
Hardware description      ::Open vSwitch
Software description      ::2.13.1
Serial number             ::None
Datapath description      ::s7

```

Figure 24: Figure depicting an intent to know the hardware and software description of switch.

In this response we are getting to know about the response code, manufacturer description, hardware description, software description, serial number and data path description. The response code of 200 indicates that the intent has been processed by the RYU controller.

- Intent to install a flow rule in switches in order to block host h1.

In this response we come to know about the response code after installation of flow entry in switches. Here the response code is 200 after installation of flow rule in switches. The result of pingall from Mininet emulator is shown below after

```

Hello! Please enter your intent
Deny host h1

[RESPONSE CODE] 200

```

Figure 25: Figure depicting an intent to install a flow rule in switches in order to block host h1.

successful processing of above intent:

```

mininet> pingall
*** Ping: testing ping reachability
h1 -> X X X X X X
h2 -> X h3 h4 h5 h6 h7
h3 -> X h2 h4 h5 h6 h7
h4 -> X h2 h3 h5 h6 h7
h5 -> X h2 h3 h4 h6 h7
h6 -> X h2 h3 h4 h5 h7
h7 -> X h2 h3 h4 h5 h6
*** Results: 28% dropped (30/42 received)

```

Figure 26: Figure depicting pingall operation in Mininet

The above Mininet result shows the host h1 is completely blocked and it cannot communicate with any other host.

- Intent to enable host h1 to communicate with all other hosts.

```

Please enter new intent or enter exit
Activate host h1

Contradiction detected,Do you want to proceed? Yes/No:
Yes

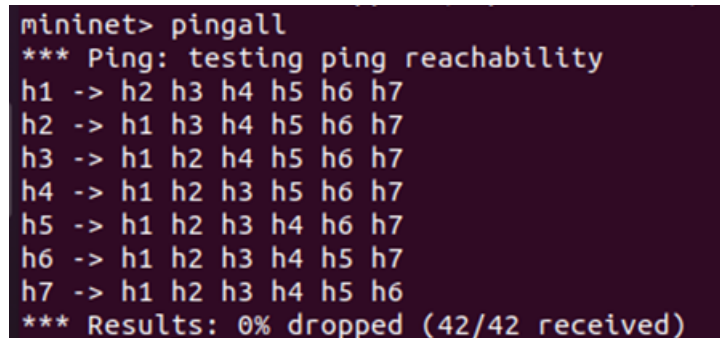
[RESPONSE CODE] 200

```

Figure 27: Figure depicting an intent to enable host h1 to communicate with all other hosts

In the response we get to know about the response code and a message displaying contradiction has been detected. It is the network operator's choice whether he

wants to proceed and install the flow rule by choosing yes or to roll back from current situation by choosing no. The pingall result from Mininet emulator is shown in below figure after the network operator chooses 'yes' when contradiction was detected.

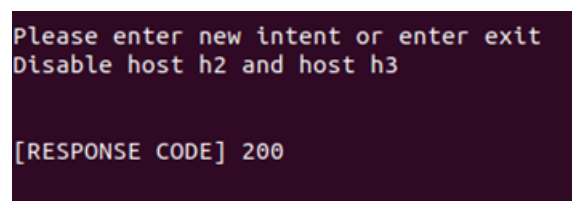


```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7
h2 -> h1 h3 h4 h5 h6 h7
h3 -> h1 h2 h4 h5 h6 h7
h4 -> h1 h2 h3 h5 h6 h7
h5 -> h1 h2 h3 h4 h6 h7
h6 -> h1 h2 h3 h4 h5 h7
h7 -> h1 h2 h3 h4 h5 h6
*** Results: 0% dropped (42/42 received)
```

Figure 28: Pingall operation in Mininet

The above Mininet result shows that host h1 can now communicate with all the other hosts present in the network as per the requirement specified by the network operator through the intent.

- Intent to block all the traffic between host h2 and host h3.



```
Please enter new intent or enter exit
Disable host h2 and host h3

[RESPONSE CODE] 200
```

Figure 29: Figure depicting an intent to block all the traffic between host h2 and host h3

Here in this response we observe that no contradiction has been detected as there is no previous intent installed which specifies enabling the traffic between the host h2 and host h3. We get a response code of 200 which indicates successful installation of flow entry. The figure below shows the output of pingall command from Mininet emulator after the flow rule has been installed.



```

mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7
h2 -> h1 X h4 h5 h6 h7
h3 -> h1 X h4 h5 h6 h7
h4 -> h1 h2 h3 h5 h6 h7
h5 -> h1 h2 h3 h4 h6 h7
h6 -> h1 h2 h3 h4 h5 h7
h7 -> h1 h2 h3 h4 h5 h6
*** Results: 4% dropped (40/42 received)

```

Figure 30: Pingall operation in Mininet

From the above result we observe that the host h2 and host h3 are not able to communicate with each other.

- Intent to allow the host h2 and host h3 to communicate with each other.

```

Please enter new intent or enter exit
Forward all the traffic between host h2 and host h3

Contradiction detected,Do you want to proceed? Yes/No:
No

```

Figure 31: Intent to allow the host h2 and host h3 to communicate with each other

After sending the request to install a flow entry in order to allow host h2 and host h3 to communicate with each other we observe that contradiction has been detected. After contradiction has been detected operator chooses 'no' and decides to roll back from current situation. The pingall result from Mininet emulator is shown below:

```

mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7
h2 -> h1 X h4 h5 h6 h7
h3 -> h1 X h4 h5 h6 h7
h4 -> h1 h2 h3 h5 h6 h7
h5 -> h1 h2 h3 h4 h6 h7
h6 -> h1 h2 h3 h4 h5 h7
h7 -> h1 h2 h3 h4 h5 h6
*** Results: 4% dropped (40/42 received)

```

Figure 32: Pingall operation in Mininet

From the above result of Mininet we observe the same result as previous because the network operator chooses not to install the flow entry in switches for allowing the host h2 and host h3 to communicate with each. Hence, the traffic remains blocked between these two hosts.

- Intent to allow host h2 and host h3 to communicate with each other.

```

Please enter new intent or enter exit
Allow communication between host h2 and host h3

Contradiction detected,Do you want to proceed? Yes/No:
Yes
[RESPONSE CODE] 200

```

Figure 33: Intent to allow host h2 and host h3 to communicate with each other

From the above response we observe that contradiction has been detected and network operator chooses 'yes' which means flow entry will be installed for allowing the host h2 and host h3 to communicate. We also get a response code of 200 which indicates that flow rule has been installed successfully. Below is the result from the Mininet emulator when reachability of hosts are checked after installation of flow rule:

```

mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7
h2 -> h1 h3 h4 h5 h6 h7
h3 -> h1 h2 h4 h5 h6 h7
h4 -> h1 h2 h3 h5 h6 h7
h5 -> h1 h2 h3 h4 h6 h7
h6 -> h1 h2 h3 h4 h5 h7
h7 -> h1 h2 h3 h4 h5 h6
*** Results: 0% dropped (42/42 received)

```

Figure 34: Pingall operation in Mininet

From the above result we observe that all hosts can communicate with each other.

- Intent to block traffic between hosts h3 and host h4.

After the intent is processed we observe that no contradiction has been detected

```

Please enter new intent or enter exit
Deactivate all the traffic between host h3 and host h4

[RESPONSE CODE] 200

```

Figure 35: Intent to block traffic between hosts h3 and host h4

here and a flow rule to block the traffic between hosts h3 and host h4 has been installed automatically. We get a response code of 200 which indicates successful installation of flow entry. Below is the figure from Mininet emulator depicting the reachability of all the hosts:

```

mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7
h2 -> h1 h3 h4 h5 h6 h7
h3 -> h1 h2 X h5 h6 h7
h4 -> h1 h2 X h5 h6 h7
h5 -> h1 h2 h3 h4 h6 h7
h6 -> h1 h2 h3 h4 h5 h7
h7 -> h1 h2 h3 h4 h5 h6
*** Results: 4% dropped (40/42 received)

```

Figure 36: Pingall operation in Mininet

From the above result we observe that the host h3 and host h4 are blocked from communicating each other as specified in the intent.

- Intent to enable host h3 and host h4 to communicate.

```

Please enter new intent or enter exit
Enable host h3 and host h4

Contradiction detected,Do you want to proceed? Yes/No:
Yes
[RESPONSE CODE] 200

```

Figure 37: Intent to enable host h3 and host h4 to communicate

In the above figure we observe that contradiction has been detected and operator chooses 'yes' and decides to install the flow entry. The below figure is the result from Mininet emulator after testing the reachability of hosts:

```

mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7
h2 -> h1 h3 h4 h5 h6 h7
h3 -> h1 h2 h4 h5 h6 h7
h4 -> h1 h2 h3 h5 h6 h7
h5 -> h1 h2 h3 h4 h6 h7
h6 -> h1 h2 h3 h4 h5 h7
h7 -> h1 h2 h3 h4 h5 h6
*** Results: 0% dropped (42/42 received)
mininet>

```

Figure 38: Pingall operation in Mininet

From the above figure we observe that all the hosts are reachable and communicates with each other.

- Intent to block the traffic between host h5 and host h7.

```

Please enter new intent or enter exit
Block host h5 and host h7

[RESPONSE CODE] 200

```

Figure 39: Intent to block the traffic between host h5 and host h7

The above figure shows that after processing of intent it is observed that no contradiction has been detected as there is no previous intent specifying to enable host h5 and host h7. A flow entry to block the traffic between host h5 and host h7 is installed in switches and we get a response code of 200 which indicates successful installation of flow rule. The result of reachability of hosts from Mininet emulator is shown below:

```

mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7
h2 -> h1 h3 h4 h5 h6 h7
h3 -> h1 h2 h4 h5 h6 h7
h4 -> h1 h2 h3 h5 h6 h7
h5 -> h1 h2 h3 h4 h6 X
h6 -> h1 h2 h3 h4 h5 h7
h7 -> h1 h2 h3 h4 X h6
*** Results: 4% dropped (40/42 received)
mininet>

```

Figure 40: Pingall operation in Mininet

From the above figure we observe that all the hosts are reachable and can communicate with each other.

- Intent to get the detailed flow entries installed in switch s1. (Flow entry must contain idle timeout, hard timeout, duration, packet count, etc.)

```

Please enter new intent or enter exit
Show detailed flows of switch s1

Getting you the flow entry of switch with ID:1
[RESPONSE CODE] 200

Priority      :65534
Cookie       :0
Idle Timeout  :0
Hard Timeout  :0
Actions      :['OUTPUT:NORMAL']
Match        :{'dl_type': 2054}
Byte Count    :9198
Duration Sec. :10941
Duration nSec.:473000000
Packet Count  :219
Table ID     :0

Priority      :204
Cookie       :5
Idle Timeout  :0
Hard Timeout  :0
Actions      :['OUTPUT:CONTROLLER']
Match        :{'dl_type': 2048, 'nw_src': '10.0.0.2', 'nw_dst': '10.0.0.3', 'nw_proto': 1}
Byte Count    :0
Duration Sec. :9318
Duration nSec.:92000000
Packet Count  :0
Table ID     :0

```

Figure 41: Intent to get the detailed flow entries installed in switch s1

```

Priority      :205
Cookie       :6
Idle Timeout  :0
Hard Timeout  :0
Actions      :['OUTPUT:NORMAL']
Match        :{'dl_type': 2048, 'nw_src': '10.0.0.2', 'nw_dst': '10.0.0.3', 'nw_proto': 1}
Byte Count   :0
Duration Sec. :7205
Duration nSec.:140000000
Packet Count  :0
Table ID     :0

Priority      :205
Cookie       :7
Idle Timeout  :0
Hard Timeout  :0
Actions      :['OUTPUT:NORMAL']
Match        :{'dl_type': 2048, 'nw_src': '10.0.0.3', 'nw_dst': '10.0.0.2', 'nw_proto': 1}
Byte Count   :0
Duration Sec. :7205
Duration nSec.:131000000
Packet Count  :0
Table ID     :0

Priority      :206
Cookie       :8
Idle Timeout  :0
Hard Timeout  :0
Actions      :['OUTPUT:CONTROLLER']
Match        :{'dl_type': 2048, 'nw_src': '10.0.0.3', 'nw_dst': '10.0.0.4', 'nw_proto': 1}
Byte Count   :0
Duration Sec. :6344
Duration nSec.:424000000
Packet Count  :0

```

Figure 42: Intent to get the detailed flow entries installed in switch s1

The above figure shows only a part of flow entries installed in switch s1. We observe many important parameters in the flow entry like hard timeout, idle timeout, nw\_src, nw\_dst, nw\_proto, dl\_type, actions, etc. These important parameters are necessary while analysing switches.

- Intent to disable switch s5 from packet forwarding.

```

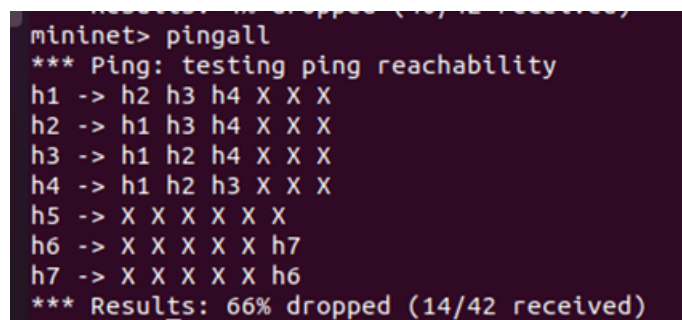
Please enter new intent or enter exit
Disable switch s5

RESPONSE CODE 200

```

Figure 43: Intent to disable switch s5 from packet forwarding

The above figure shows an intent which is specified to disable switch s5 from communication. After processing of this intent we observe that no contradiction has been detected and we get a response code of 200 which indicates that the request to block switch s5 has been sent successfully to the RYU controller.



```

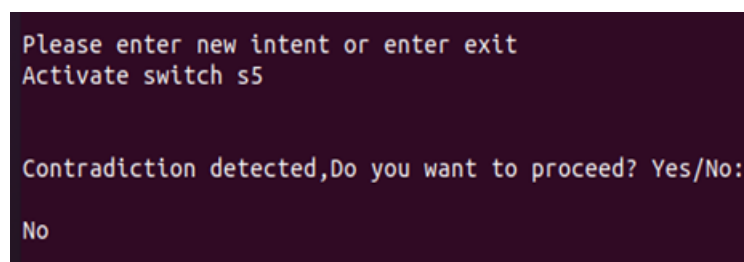
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 X X X
h2 -> h1 h3 h4 X X X
h3 -> h1 h2 h4 X X X
h4 -> h1 h2 h3 X X X
h5 -> X X X X X X
h6 -> X X X X X h7
h7 -> X X X X X h6
*** Results: 66% dropped (14/42 received)

```

Figure 44: Pingall operation in Mininet

The above figure depicts the result of reachability from the Mininet emulator. When the switch s5 is blocked in the given linear topology then the hosts from h1 to h4 can communicate with each other and host h6 and h7 can communicate with each other. The host communicates with each other as if there is a split at host h5 and no host can communicate in either side.

- Intent to enable switch s5.



```

Please enter new intent or enter exit
Activate switch s5

Contradiction detected,Do you want to proceed? Yes/No:
No

```

Figure 45: Intent to enable switch s5

The above figure shows that while allowing switch s5 to communicate, contradiction has been detected and as usual network operator is given a choice to choose to



which 'no' was chosen.

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 X X X
h2 -> h1 h3 h4 X X X
h3 -> h1 h2 h4 X X X
h4 -> h1 h2 h3 X X X
h5 -> X X X X X X
h6 -> X X X X X h7
h7 -> X X X X X h6
*** Results: 66% dropped (14/42 received)
```

Figure 46: Pingall operation in Mininet

The figure above is the result of pingall operation from Mininet emulator. We observe here that no change in result takes place as previous because the operator chooses 'no' as choice.

- Intent to enable switch s5 and choosing 'Yes' as contradiction detection choice.

```
Please enter new intent or enter exit
Enable switch s5

Contradiction detected,Do you want to proceed? Yes/No:
Yes

RESPONSE CODE 200
```

Figure 47: Intent to enable switch s5

We observe from the above figure that contradiction is detected in enabling the switch s5 to communicate and the network operator chooses 'yes' as choice to which we get a response code of 200.

```

mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7
h2 -> h1 h3 h4 h5 h6 h7
h3 -> h1 h2 h4 h5 h6 h7
h4 -> h1 h2 h3 h5 h6 h7
h5 -> h1 h2 h3 h4 h6 X
h6 -> h1 h2 h3 h4 h5 h7
h7 -> h1 h2 h3 h4 X h6
*** Results: 4% dropped (40/42 received)

```

Figure 48: Pingall operation in Mininet

From the above figure we observe that all the hosts are communicating in normal way as usual before disabling of switch s5. The host h5 and host h7 are not able to communicate as no flow entry has been installed to allow the communication of these two hosts.

- Intent to allow host h5 and host h7 to communicate.

```

Please enter new intent or enter exit
Enable host h5 and host h7

Contradiction detected,Do you want to proceed? Yes/No:
Yes
[RESPONSE CODE] 200

```

Figure 49: Intent to allow host h5 and host h7 to communicate

From the figure above we observe that when contradiction detection was predicted by the machine learning model while allowing the host h5 and host h7 to communicate with each other the network operator chooses 'yes' as choice and continues forward to install the corresponding flow entry.

```

mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7
h2 -> h1 h3 h4 h5 h6 h7
h3 -> h1 h2 h4 h5 h6 h7
h4 -> h1 h2 h3 h5 h6 h7
h5 -> h1 h2 h3 h4 h6 h7
h6 -> h1 h2 h3 h4 h5 h7
h7 -> h1 h2 h3 h4 h5 h6
*** Results: 0% dropped (42/42 received)
mininet>

```

Figure 50: Pingall operation in Mininet

The above figure shows that all the hosts are now reachable and can communicate with each other.

- Intent to get all the flow entries installed in switch s1 with rule id.

```

Please enter new intent or enter exit
Get the flows of switch s1 with rule id

[RESPONSE CODE] 200

{'rule_id': 5, 'priority': 204, 'dl_type': 'IPv4', 'nw_src': '10.0.0.2', 'nw_dst': '10.0.0.3', 'nw_proto': 'ICMP', 'actions': 'DENY'}
{'rule_id': 6, 'priority': 205, 'dl_type': 'IPv4', 'nw_src': '10.0.0.2', 'nw_dst': '10.0.0.3', 'nw_proto': 'ICMP', 'actions': 'ALLOW'}
{'rule_id': 7, 'priority': 205, 'dl_type': 'IPv4', 'nw_src': '10.0.0.3', 'nw_dst': '10.0.0.2', 'nw_proto': 'ICMP', 'actions': 'ALLOW'}
{'rule_id': 8, 'priority': 206, 'dl_type': 'IPv4', 'nw_src': '10.0.0.3', 'nw_dst': '10.0.0.4', 'nw_proto': 'ICMP', 'actions': 'DENY'}
{'rule_id': 9, 'priority': 207, 'dl_type': 'IPv4', 'nw_src': '10.0.0.3', 'nw_dst': '10.0.0.4', 'nw_proto': 'ICMP', 'actions': 'ALLOW'}
{'rule_id': 10, 'priority': 207, 'dl_type': 'IPv4', 'nw_src': '10.0.0.4', 'nw_dst': '10.0.0.3', 'nw_proto': 'ICMP', 'actions': 'ALLOW'}
{'rule_id': 11, 'priority': 208, 'dl_type': 'IPv4', 'nw_src': '10.0.0.5', 'nw_dst': '10.0.0.7', 'nw_proto': 'ICMP', 'actions': 'DENY'}
{'rule_id': 12, 'priority': 209, 'dl_type': 'IPv4', 'nw_src': '10.0.0.5', 'nw_dst': '10.0.0.7', 'nw_proto': 'ICMP', 'actions': 'ALLOW'}
{'rule_id': 13, 'priority': 209, 'dl_type': 'IPv4', 'nw_src': '10.0.0.7', 'nw_dst': '10.0.0.5', 'nw_proto': 'ICMP', 'actions': 'ALLOW'}
{'rule_id': 2, 'priority': 202, 'dl_type': 'IPv4', 'nw_src': '10.0.0.1', 'nw_proto': 'ICMP', 'actions': 'DENY'}
{'rule_id': 3, 'priority': 203, 'dl_type': 'IPv4', 'nw_src': '10.0.0.1', 'nw_proto': 'ICMP', 'actions': 'ALLOW'}
{'rule_id': 4, 'priority': 203, 'dl_type': 'IPv4', 'nw_dst': '10.0.0.1', 'nw_proto': 'ICMP', 'actions': 'ALLOW'}
{'rule_id': 1, 'priority': 201, 'dl_type': 'IPv4', 'nw_proto': 'ICMP', 'actions': 'ALLOW'}

```

Figure 51: Figure depicting an intent to get all the flow entries installed in switch s1 with rule id

The above figure shows all flow entries present in switch s1 with rule id. The different parameters we observe in flow entry are rule\_id, priority, dl\_type, nw\_src, nw\_dst, nw\_proto and actions. Rule id is necessary when any flow entry from

switch has to be deleted. This intent is required for analysing different switches and removing latter if any flow entry is not required.

- Intent to remove a flow entry from switch s1 with rule id 13.

```
Please enter new intent or enter exit
Remove a flow rule from switch

NOTE: Please do not delete any flow entries which are currently in use by RYU controller
RULE ID: 13
SWITCH ID: 1
RESPONSE CODE 200
```

Figure 52: Intent for removal of a flow entry

In the above figure we observe that a request to delete a flow entry in switch s1 with rule id 13 has been sent. After the successful removal of flow entry, we get a response code of 200.

- Intent to view the flows of switch s1 with rule id.

```
Please enter new intent or enter exit
Show flows of switch s1 with rule id

[RESPONSE CODE] 200

{'rule_id': 5, 'priority': 204, 'dl_type': 'IPv4', 'nw_src': '10.0.0.2', 'nw_dst': '10.0.0.3', 'nw_proto': 'ICMP', 'actions': 'DENY'}
{'rule_id': 6, 'priority': 205, 'dl_type': 'IPv4', 'nw_src': '10.0.0.2', 'nw_dst': '10.0.0.3', 'nw_proto': 'ICMP', 'actions': 'ALLOW'}
{'rule_id': 7, 'priority': 205, 'dl_type': 'IPv4', 'nw_src': '10.0.0.3', 'nw_dst': '10.0.0.2', 'nw_proto': 'ICMP', 'actions': 'ALLOW'}
{'rule_id': 8, 'priority': 206, 'dl_type': 'IPv4', 'nw_src': '10.0.0.3', 'nw_dst': '10.0.0.4', 'nw_proto': 'ICMP', 'actions': 'DENY'}
{'rule_id': 9, 'priority': 207, 'dl_type': 'IPv4', 'nw_src': '10.0.0.3', 'nw_dst': '10.0.0.4', 'nw_proto': 'ICMP', 'actions': 'ALLOW'}
{'rule_id': 10, 'priority': 207, 'dl_type': 'IPv4', 'nw_src': '10.0.0.4', 'nw_dst': '10.0.0.3', 'nw_proto': 'ICMP', 'actions': 'ALLOW'}
{'rule_id': 11, 'priority': 208, 'dl_type': 'IPv4', 'nw_src': '10.0.0.5', 'nw_dst': '10.0.0.7', 'nw_proto': 'ICMP', 'actions': 'DENY'}
{'rule_id': 12, 'priority': 209, 'dl_type': 'IPv4', 'nw_src': '10.0.0.5', 'nw_dst': '10.0.0.7', 'nw_proto': 'ICMP', 'actions': 'ALLOW'}
{'rule_id': 2, 'priority': 202, 'dl_type': 'IPv4', 'nw_src': '10.0.0.1', 'nw_proto': 'ICMP', 'actions': 'DENY'}
{'rule_id': 3, 'priority': 203, 'dl_type': 'IPv4', 'nw_src': '10.0.0.1', 'nw_proto': 'ICMP', 'actions': 'ALLOW'}
{'rule_id': 4, 'priority': 203, 'dl_type': 'IPv4', 'nw_dst': '10.0.0.1', 'nw_proto': 'ICMP', 'actions': 'ALLOW'}
{'rule_id': 1, 'priority': 201, 'dl_type': 'IPv4', 'nw_proto': 'ICMP', 'actions': 'ALLOW'}
```

Figure 53: Intent to get the flows of switch with different rule id's

From the above figure we observe that the flow entry with rule id 13 has been removed from the switch s1 and rest all the flows are present.

- Intent to get the status of all the switch present in the topology.

```

Please enter new intent or enter exit
Get the status of all the switches present in the topology

Switch ID :7
Status :enable
Switch ID :1
Status :enable
Switch ID :3
Status :enable
Switch ID :4
Status :enable
Switch ID :2
Status :enable
Switch ID :5
Status :enable
Switch ID :6
Status :enable

```

Figure 54: Intent to get the status of all the switch present in the topology

From the above figure we observe that there are total of 7 switches present in the topology and currently all the switches are connected to the RYU controller as enable is displayed in all the switches.

- Intent to know the port status of switch s1.

```

Get the port status of switch s1

Port number: 1
Received Packets: 290
Transmitted Packets: 359
Received Bytes: 19840
Transmitted Bytes: 24909
Packets dropped(receiver side): 0
Packets dropped(transmitter side): 0
Received packets errors: 0
Transmitted packets errors: 0
Collisions: 0
Duration(sec): 18829

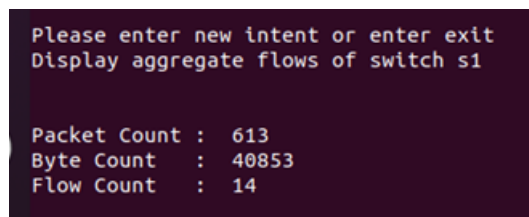
Port number: 2
Received Packets: 358
Transmitted Packets: 302
Received Bytes: 24819
Transmitted Bytes: 21795
Packets dropped(receiver side): 0
Packets dropped(transmitter side): 0
Received packets errors: 0
Transmitted packets errors: 0
Collisions: 0
Duration(sec): 18829

```

Figure 55: Figure depicting an intent to know the port status of switch s1

From the figure above we observe the port status of switch s1. All the necessary information regarding port has been fetched. The important parameters for every port number as shown above are number of received packets and transmitted packets, number of received and transmitted bytes, number of packets dropped while receiving and transmitting packets, number of errors while receiving and transmitting packets, number of collisions and finally the duration in seconds.

- Intent to get the aggregate flows of switch s1.

A terminal window with a dark background and light-colored text. The text displays a prompt and the aggregate flow statistics for switch s1.

```
Please enter new intent or enter exit
Display aggregate flows of switch s1

Packet Count : 613
Byte Count   : 40853
Flow Count   : 14
```

Figure 56: Intent to get the aggregate flows of switch s1

From the above figure we observe the aggregate flows of switch s1 which includes packet count, byte count and flow count.

# **Chapter 7**

## **Future Work and Conclusion**

### **7.1 Future Work**

In the future the researches based on this field should be accomplishing more network related task and providing the network operators with more functionalities. In the contradiction detection part focus must be laid to decrease the dataset size with increase in accuracy and number of hosts which is currently limited to ten.

### **7.2 Conclusion**

The work we have proposed is an interface between user and network which could be completely operated by intents in natural languages. For accomplishing the task for contradiction detection we have composed a dataset and then trained and tested four different machine learning models i.e. Logistic Regression, Naïve Bayes, Random Forest Classifier and Decision Tree Classifier. Upon observation of results we found out that Decision Tree Classifier gives best results as compared with other machine learning models. To the best of our knowledge this is the first work that provides such an interface by using NLP features and RYU controller additionally it also provides contradiction detection on hosts and switches to detect previously deployed intents.

# References

- [1] Mininet - an instant virtual network on your laptop or other pc. [online] available: <http://mininet.org/>.
- [2] Pox documentation available <https://noxrepo.github.io/pox-doc/html/>.
- [3] Ryu controller documentation available: <https://ryu.readthedocs.io/en/latest/index.html>.
- [4] Intent: Don't tell me what to do! accessed available at <https://www.sdxcentral.com/articles/contributed>. 2015.
- [5] Cisco intent-based networking available: <https://www.cisco.com/c/en/us/solutions/intent-based-networking.html>. 2018.
- [6] Huawei : Idn maximize your business value available on <https://developer.huawei.com/ict/en/site-idn>. 2018.
- [7] Azzam Alsudais and Eric Keller. Hey network, can you understand me? *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS): SWFAN 17*, 2017.
- [8] C Bureau. Network operators to spend over 1.8 billion dollars in 2016. 2016.
- [9] Prof. Dr. Ing. Georg Carle. Implementation and performance analysis of firewall on openvswitch. 2015.
- [10] Flavio Esposito, Jiayi Wang, Chiara Contolii, Gianluca Davoli, Walter Cerroni, and Franco Callegati. A behavior-driven approach to intent specification for software defined infrastructure management. *In IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, 2018.



- [11] Nick Feamster, Jennifer Rexford, and Ellen Zegura. The road to sdn: An intellectual history of programmable networks. *ACM SIGCOMM Computer Communication*, 44(2), 2014.
- [12] Gartner. Innovation insight: Intent-based networking systems available: <https://www.gartner.com/doc/3599617/innovation-insight-intent-based-networkingsystems>. 2017.
- [13] V. Heorhiadi, S. Chandrasekaran, M. K. Reiter, and V. Sekar. Intent driven composition of resource-management sdn applications. pages 86–97, 2018.
- [14] Sukhveer Kaur, Japinder Singh, and Navtej Singh Ghumman. Network programmability using pox controller. 2014.
- [15] Mariam Kiran, Eric Pouyoul, Anu Mercian, Brian Tierney, Chin Guok, and Inder Monga. Enabling intent to configure scientific networks for high performance demands. 2017.
- [16] Diego Kreutz, Fernando M. V. Ramos, Paulo Esteves, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. Software-defined networking: A comprehensive survey. *In Proceedings of the IEEE*, 103(1), 2015.
- [17] Lei Pang, Chungang Yang, Danyang Chen, Yanbo Song, and Mohsen Guizani. A survey on intent-driven networks. 2020.
- [18] Arthur S. Jacobs, Ricardo J. Pitscher, Rafael H. Ribeiro, Ronaldo A. Ferreira, Lisandro Z. Granville, and Sanjay G. Rao. Deploying natural language intents with lumi. *ACM*, 2019.
- [19] B. T. Saha, D. Tandur, L. Haab, and L. Podleski. Intent-based networks: An industrial perspective. pages 35–40, 2018.
- [20] D. Schulz. Intent-based automation networks: Toward a common reference model for the self-orchestration of industrial intranets. pages 4657– 4664, 2016.

- [21] Robert Soule, Shrutarshi Basu, Parisa Jalili Marandi, Fernando Pedone, Robert Kleinberg, Emin Gun Sirer, and Nate Foster. Merlin: A language for managing network resources. *IEEE/ACM TRANSACTIONS ON NETWORKING*, 26(5), 2018.
- [22] H. Zhang, Y. Wang, X. Qi, W. Xu, T. Peng, and S. Liu. Demo abstract: An intent solver for enabling intent-based sdn. pages 968–969, 2017.