



Model Identification and Control of Priority Queueing in Software Defined Networks

Enrico Reticcioli

Department of Information Engineering,
Computer Science and Mathematics

Ph.D. Program in ICT - System Engineering, Telecommunications and HW/SW Platforms
XXXIII cycle - SSD ING-INF/03

Università degli Studi dell'Aquila

Advisor: Prof. Alessandro D'Innocenzo
Co-Advisor: Prof. Fabio Graziosi

Coordinator: Prof. Vittorio Cortellessa

A thesis submitted for the degree of
Doctor of Philosophy

2021

Acknowledgements

This work was supported by the Italian Government under Cipe resolution n.135 (Dec. 21, 2012), project *INnovating City Planning through Information and Communication Technologies* (INCIPICT).

Abstract

The heterogeneity of modern network infrastructures involves different devices and protocols bringing out several issues in organizing and optimizing network resources, making their coexistence a very challenging engineering problem. In this scenario, Software Defined Network (SDN) architectures decouple control and forwarding functionalities by enabling the network devices to be remotely configurable/programmable in run-time by a controller, and the underlying infrastructure to be abstracted from the application layer and the network services, with the final aim of increasing flexibility and performance. As a direct consequence identifying an accurate model of a network and forwarding devices is crucial in order to apply advanced control techniques such as Model Predictive Control (MPC) to optimize the network performance. An enabling factor in this direction is given by recent results that appropriately combine System Identification and Machine Learning techniques to obtain predictive models using historical data retrieved from a network. This thesis presents a novel methodology to learn, starting from historical data and appropriately combining autoregressive exogenous(ARX) identification with Regression Trees (RT) and Random Forests (RF), an accurate model of the dynamical input-output behavior of a network device that can be directly and efficiently used to optimally and dynamically control the bandwidth of the queues of switch ports, within the SDN paradigm. Both the Mininet network emulator environment and a real dataset obtained from measurements of the network of an Italian internet service provider (Sonicate S.r.l.). have been used to validate the prediction accuracy of the derived predictive models. The benefits of the proposed dynamic queueing control methodology in terms of Packet Losses reduction and Bandwidth savings (i.e. improvement of the Quality of Service) has been finally demonstrated.

Contents

Abstract	4
Introduction	1
1 Background Knowledge	3
1.1 Software Defined Networks Architecture	3
1.1.1 Data Plane	3
1.1.2 Control Plane	4
1.1.3 Application Plane	5
1.2 Overview Of Machine Learning Algorithms	7
1.2.1 Supervised Learning	7
1.2.2 Unsupervised Learning	13
1.2.3 Semi-Supervised Learning	14
1.2.4 Reinforcement Learning	14
1.3 Related Works	16
2 RT- and RF-based models of SDN switched for Priority Queueing	21
2.1 Mininet network emulation environment and control problem	21
2.2 Regression Trees and Random Forest based models for MPC	24
2.2.1 RT and RF background	25
2.2.2 Switching ARX (SARX) model identification via RT	26
2.2.3 SARX model identification via RF	28
2.2.4 MPC problem formulation.	29
2.2.5 Disturbance forecast	30
2.3 Simulation results	30
2.3.1 Disturbance predictive model validation	31
2.3.2 Queues predictive model validation	33
2.3.3 Control performance	37

3 RT- and RF-based predictive models of multi-service traffic in a real Service Provider Network	43
3.1 Control performance validation over dedicated hardware network	43
3.2 Traffic predictive model validation on Italian Internet provider network	47
Conclusions	49
References	51
Publications	69
A Python Codes for Mininet Environment	73
A.1 main_controller_TOS.py	73
A.2 datapath_monitor_TOS.py	74
A.3 Controller_commands.py	82
A.4 qos_simple_switch_13.py	86
A.5 ofctl_rest.py	89
A.6 rest_conf_switch.py	104
A.7 rest_qos.py	107
A.8 Topology_qos_TRAFFIC_GEN.py	129
A.9 ditg.py	138
B Python Codes For Dedicated Hardware Network Devices	141
B.1 Traffic_Real_Hwd.py	141
B.2 Start_ITGRecv.py	147
B.3 ditg.py	147
B.4 Set_Queue.py	149
B.5 SendTime.py	150
B.6 ReadTime.ino	152

List of Figures

1.1	The high-level SDN architecture proposed by ONE.	4
1.2	Example of OpenFlow-based SDN network.	6
1.3	Common machine learning algorithms.	8
1.4	A basic neural network with three layers: an input layer, a hidden layer and an output layer.	10
2.1	Mininet emulated network architecture.	22
2.2	Static queues rate with routed packets relative to DSCP.	23
2.3	NRMSE of the disturbance predictive model over a time horizon of $N = 5$	32
2.4	Comparison between the real traffic (YELLOW LINE) and the traffic prediction for the different models for Service 0.	32
2.5	NRMSE, up to $N = 5$ and for each priority class, for RT (blue), RF (red), NN with sigmoids as activation function (yellow) and NN with hyperbolic tangent as activation function (black).	34
2.6	Grid pattern emerged from dataset distribution.	35
2.7	NRMSE of the queues output predictive model over a time horizon of $N = 5$, without knowledge of the future disturbances	36
2.8	NRMSE of the queues output predictive model over a time horizon of $N = 5$, with knowledge of the 4-steps future disturbances	36
2.9	Cumulative Packet Losses without knowledge of the future disturbance.	38
2.10	Comparison between Cumulative Packet Losses with (solid lines) and without (dashed lines) knowledge of the future disturbance.	38
2.11	Bandwidth saving comparison without (a) and with (b) knowledge of the future disturbances.	40
2.12	Static controller up to the 400th, then MPC controller.	41
3.1	Network architecture with dedicated hardware devices.	44

3.2	Packets that transit inside the switch (a), comparison between packets incoming in the switch in emulation environment and in dedicated hardware (b) and sum of packets sent by the switch port with different control model (c)	46
3.3	NRMSE of the packets predictive model over a time horizon of $N = 10$	47
3.4	NRMSE of the packets predictive model over a time horizon of $N = 10$ after one data update.	48
3.5	NRMSE of the packets predictive model at prediction horizon of $N=1$ with different updated models. Every update add 288 samples (about 1 day) to the previous model training data.	49

List of Tables

2.1 Identification parameters	33
2.2 Constraints in Problem 3	37

Introduction

A communication network involves the interconnection of a large number of devices, protocols and applications, as well as application-, service- and user-specific Quality of Service (QoS) and Quality of Experience (QoE) requirements: the problem of optimizing the performance of such a complex distributed system while guaranteeing the desired QoS and QoE specifications is a very challenging engineering problem since the heterogeneity and complexity of such network infrastructures pose a number of challenges in effectively modeling, managing and optimizing network resources (e.g. see [1, 2] and references therein).

Thanks to the recently introduced Software Defined Network (SDN) paradigm [3] the control plane and the data plane are decoupled: this enables the possibility of learning (i.e. identifying) dynamical network models to be used for management and optimization purposes. Indeed, in SDN, network resources are managed by a logically centralized controller that have a global view of the network: this feature provides the capacity of monitoring and collecting, in real-time, data on the network state and configuration as well as packet and flow-granularity information [4]. More in detail, a SDN controller device can configure the forwarding state of each switch by using a standard protocol called OpenFlow (OF) [5]. Thanks to the OF *counter variables* (e.g. flow statistics, port statistics, queue statistics, etc.), the controller can retrieve information (feedback) from the network devices and store/process them for optimization purposes [6]. A SDN controller can supervise many aspects of traffic flow, as segment routing and queue management on switch ports.

Indeed, the most difficult challenge to be addressed in order to apply optimization techniques is to derive a predictive model of the queues of the switch behaviour. To the best of the author knowledge the state of the art in deriving accurate dynamical models of communication networks still lacks of methods that exploit historical network data to learn (identify) a dynamical network model that can be directly used for optimal control (e.g. of segment routing and/or queue management) and is practical from the computational complexity point of view.

The goal of this research is exploiting control theory combined with Machine Learning techniques to compute an accurate model for predicting queues state inside a switch ports,

and use this model to optimally control queues scheduling via bandwidth allocation. In particular, a novel methodology to learn an accurate model of the dynamical input-output behavior of a switch device starting from historical data, that combines ARX identification with Regression Trees (RT) and Random Forests (RF) algorithms [7, 8, 9], has been presented as the main contribution [10]. At first a comparison between the prediction accuracy of the proposed technique with respect to Neural Network (NN) models has been shown. Then in a network emulation environment the proposed novel identification technique (differently from NNs, that provide nonlinear predictive models that are impractical for optimization from the computational complexity viewpoint) has been directly and efficiently used to control the bandwidth of the queues of switch ports with the final aim of reducing packet losses, and thus improving QoS, taking into account the priority of different services. We validate our approach both on real traffic data and on an emulative setup.

The manuscript is organized as follows: a background knowledge about SDN and Machine learning has been introduced in Section 1.1 and in Section 1.2 respectively and in Section 1.3 a survey on related works is provided; in Section 2.1 the network emulation environment has been illustrated; in Section 2.2 the model identification technique and its embedding in a MPC problem formulation solvable via Quadratic Programming (QP) has been described; in Section 2.3 the prediction accuracy and control performance validation using the proposed emulation environment has been provided.

Chapter 1

Background Knowledge

This chapter briefly presents the main characteristics of SDN and the main ML algorithms of interest. For a more in-depth knowledge on ML applied to the SDN we refer to [11].

1.1 Software Defined Networks Architecture

The Open Networking Foundation (ONF) [12] is a nonprofit consortium dedicated to the development and standardization of SDN. The SDN paradigm has been defined by ONF as follows: “In the SDN architecture, the control plane and data plane are decoupled, network intelligence and state are logically centralized, and the underlying network infrastructure is abstracted from the applications” [13]. A SDN architecture is composed by three main planes, including data plane, control plane and application plane. The architectural components of each plane and their interactions are shown in Figure 1.1. In the following, we give a brief description of these planes and their interactions.

1.1.1 Data Plane

The data plane, or infrastructure plane, is the lowest layer in SDN architecture. This plane is composed by physical switches, virtual switches and others forwarding devices. Virtual switches are software-based switches, which can run on common operating systems. Open vSwitch [14], Indigo [15] and Pantou [16] are three implementations of virtual switches. Physical switches are hardware-based switches. They can be implemented on open network hardware (e.g., NetFPGA [17]) or implemented on networking hardware vendors’ merchant switches. Many networking hardware vendors such as HP, NEC, Huawei, Juniper and Cisco, have supported SDN protocols. Virtual switches support complete features of SDN protocols, while physical switches lack the flexibility and feature completeness. However, physical switches have a higher flow forwarding rate than virtual switches. SwitchBlade

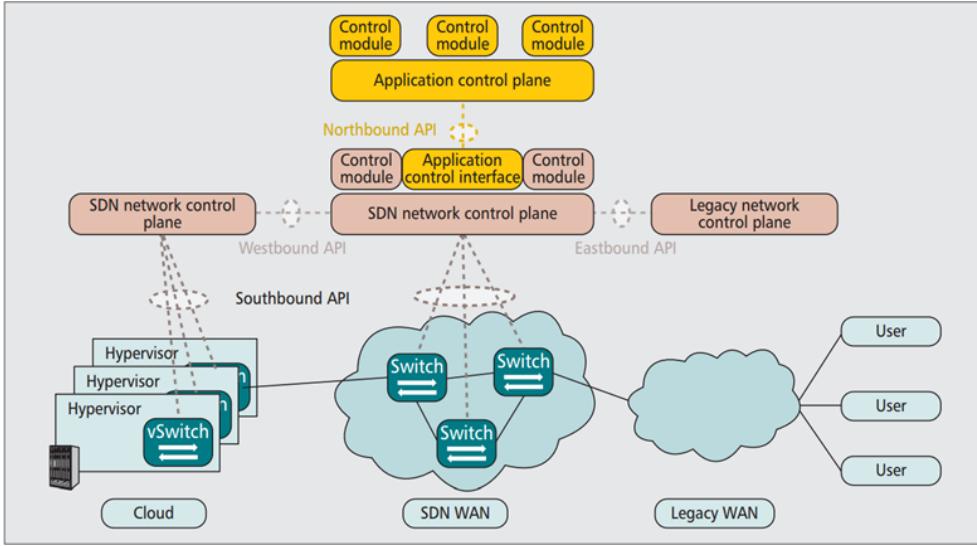


Figure 1.1: The high-level SDN architecture proposed by ONF.

[18] and ServerSwitch [19] are two NetFPGA-based physical switches. These switches in data plane are responsible for forwarding, dropping and modifying packets based on instructions received from the Control Plane (CP) through Southbound Interfaces (SBIs).

1.1.2 Control Plane

The control plane is the “brain” of SDN systems, which can define network resources, dynamically choose forwarding rules and make network administration flexible and agile. The controller is responsible of many relevant tasks like:

- the communication between forwarding devices and applications;
- it exposes and abstracts network state information of the data plane to the application plane;
- it translates the requirements from applications into custom policies and distributes them to forwarding devices;
- provides essential functionalities that most of network applications need, such as shortest path routing, network topology storage, device configuration and state information notifications etc.

There are many controller architectures, such as Ryu [20], OpenDayLight, [21] NOX [22], POX [23], Floodlight [24] and Beacon [25]. Three communication interfaces allow the controllers to interact: southbound, northbound (NBI) and eastbound/westbound interfaces.

The SBIs are defined between the control plane and the data plane. They allow forwarding devices to exchange network state information and control policies with the CP and provide functions such as statistics reports, forwarding operations, programmatic control of all device-capability advertisements and event notifications. OpenFlow [5] promoted by ONF is the first and the most popular open standard SBI. There exist other less popular proposals such as OVSDB [26], Protocol-Oblivious Forwarding (POF) [27] and OpenState [28]. With NBIs, automation, innovation and management of SDN networks has been facilitated thanks to the fact that applications can exploit the abstract network views provided by the CP. The ONF is trying to define the standard NBIs and a common information model. The eastbound/westbound interfaces are used in the multi-controller SDN networks. Due to the vast amount of data flows in such networks and the limited processing capacity of one controller, large-scale networks are always partitioned into several domains where each domain has its own controller. The eastbound/westbound interfaces are responsible for the communication among multiple controllers. This communication is necessary to exchange information in order to provide a global network view to the upper-layer applications. Onix [29] and HyperFlow [30] are two distributed control architectures. Because their eastbound-/westbound interfaces are private, they cannot communicate with each other. To enable the communication between different types of SDN controllers, SDNi [31], East-West Bridge [32] and Communication Interface for Distributed Control plane (CIDC) [33] have been proposed as eastbound/westbound interfaces to exchange network information. However, the eastbound/westbound interfaces have not yet been standardized.

1.1.3 Application Plane

The highest layer in the SDN architecture is the application plane. These applications can provide new services and perform business management, optimization and can obtain the required network state information through controllers' NBIs. Based on the received information and other requirements, the applications can apply some control logic to change network behaviors. The SDN-based applications have attracted a lot of attention from academia. Mendiola et al. [34] have discussed the impact of SDN on Traffic Engineering (TE) and surveyed the SDN-based TE solutions. Security in SDN has been surveyed in [35, 36, 37, 38, 39, 40]. Especially, Yan et al. [39] have researched on Distributed Denial of Service (DDoS) attacks in SDN-based cloud computing systems, and discussed future research challenges. Fault management in SDN has been surveyed in [41], which gives an identification and classification of the main fault management issues, and does valuable surveys and discussions about efforts that address those issues. Guck et al. [42] have studied the centralized QoS routing mechanisms in SDN, and introduced a novel Four-Dimensional

(4D) evaluation framework. SDN has been deployed in many networks, such as transport networks [43], optical networks [44], wireless networks [45, 46], Internet of Things (IoT) [47], edge computing [48], Wide Area Networks (WAN) [49], cloud computing [50], Network Function Virtualization (NFV) [51, 52].

For more details on SDN, please refer to [53, 54, 55, 56, 57, 58, 59, 60].

To understand the SDN architecture, it is important to recall its basic operation. Figure 1.2 shows the working procedure of the OpenFlow-based SDN network [6]. Each OpenFlow switch has a flow table and uses the OpenFlow protocol to communicate with the SDN controller. The messages transmitted between the OpenFlow-based switches and the software-based controller are standardized by the OpenFlow protocol [25]. The OpenFlow controller can manage the traffic forwarding by modifying flow entries in switches flow tables. The flow table in the OpenFlow switch is comprised of flow entries to determine the processing actions of different packets on the data plane. When an OpenFlow switch receives a packet on the data plane, the packet header fields will be extracted and matched

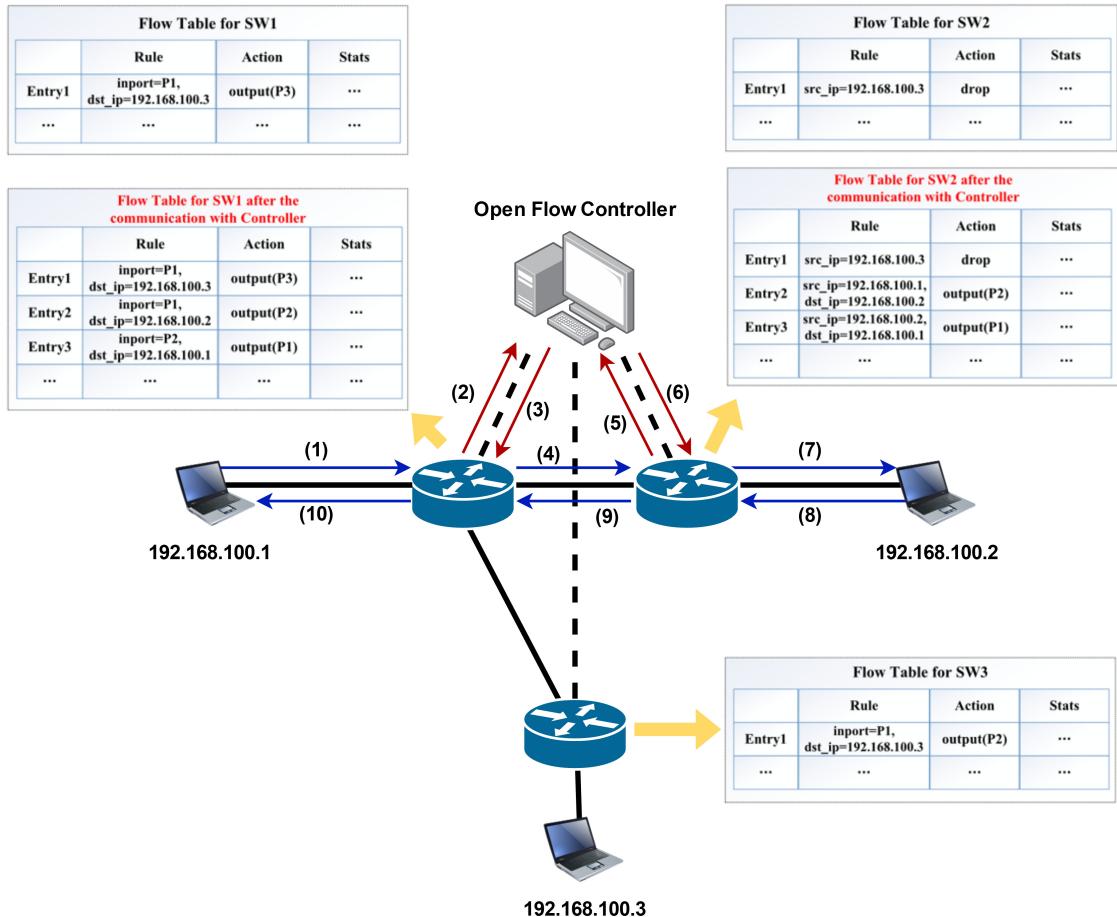


Figure 1.2: Example of OpenFlow-based SDN network.

against flow entries. If a matching entry is found, the switch will process the packet locally according to the actions in matched flow entry. Otherwise, the switch will forward an OpenFlow PacketIn message to the controller (arrows 2 and 5). The packet header (or the whole packet, optionally) is included in the OpenFlow PacketIn message. Then, the controller will send OpenFlow FlowMod messages to manage the switch's flow table by adding flow entries (arrows 3 and 6), which can be used to process subsequent packets of the flow. For example, by adding two flow entries (i.e., Entry2 and Entry3) at SW1 and SW2, the communications between 192.168.100.1 and 192.168.100.2 are allowed. However, packets from 192.168.100.3 to 192.168.100.2 are denied at SW2 due to security policies.

1.2 Overview Of Machine Learning Algorithms

Machine learning is evolved from a collection of powerful techniques in AI areas. These methods start from training data to learn useful structural patterns and models. A machine learning approach consists of two main phases: the training phase and the decision making phase. In the training phase, after a data mining period that creates a training dataset, machine learning methods are applied to learn a system model. In the decision making phase, the trained model is used to estimate the output corresponding to each new input. Machine learning algorithms can be distinguished into four main categories: supervised, unsupervised, semi-supervised and reinforcement learning. Each algorithm in Figure 1.3 is briefly explained with some examples. For a more insightful discussion on machine learning theory, please refer to [61, 62, 63].

1.2.1 Supervised Learning

Supervised learning is a kind of labelling learning technique. Supervised learning algorithms are given a labeled training dataset (i.e., inputs and known outputs) to build the system model representing the learned relation between the input and output. After training, when a new input is fed into the system, the trained model can be used to get the expected output [64, 65]. In the following, an exhaustive representation of supervised learning algorithms is provided:

1) *k-Nearest Neighbor* (k-NN): In k-NN the classification of a data sample is determined based on the k nearest neighbors of that unclassified sample. The process of the k-NN algorithm is very simple: if the most of the k nearest neighbors belong to a certain class, the unclassified sample will be classified into that class. The higher the value of k is, the less effect the noise will have on the classification. Since the

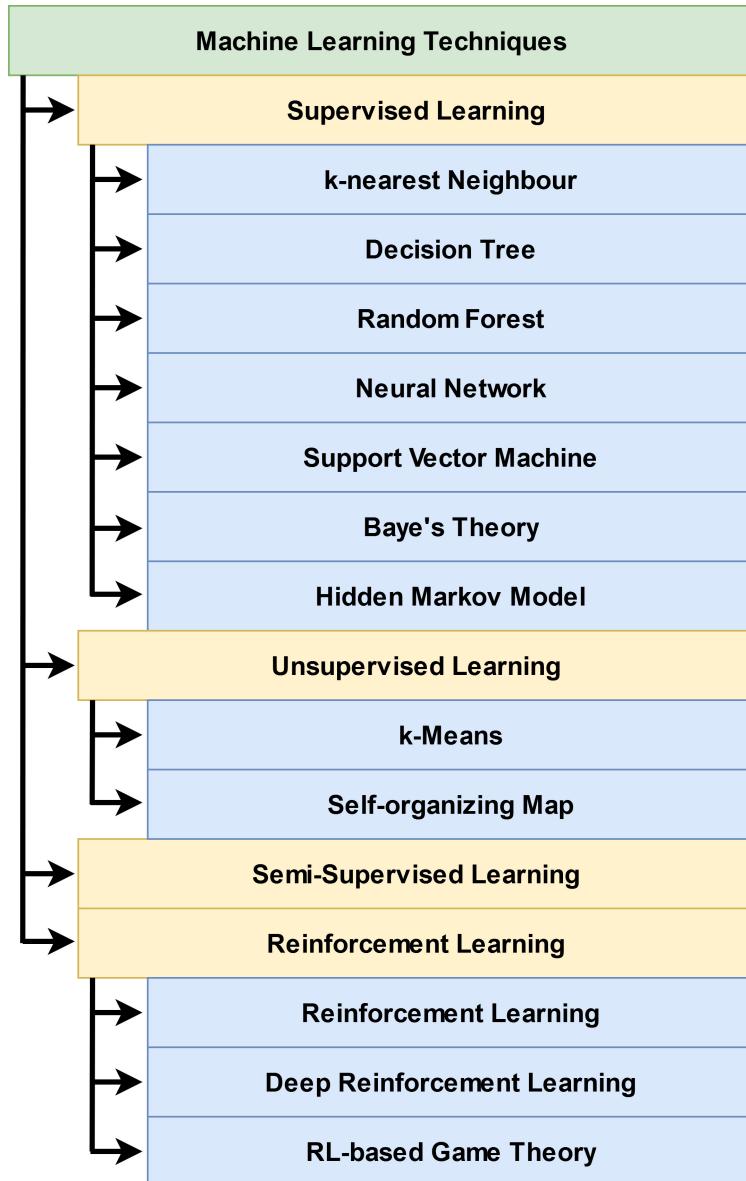


Figure 1.3: Common machine learning algorithms.

distance is the main metric of the k-NN algorithm, several functions can be applied to define the distance between the unlabeled sample and its neighbors, such as Cheby-shev, City-block, Euclidean and Euclidean squared [66].

2) *Regression Tree*: The RT performs classification through a learning tree. In the tree, each node represents a data feature, all branches represent the conjunctions of features that lead to classifications, and each leaf node is a class label. The unlabeled sample can be classified by comparing its feature values with the nodes of the RT [67]. The RT has many advantages, such as intuitive knowledge expression, simple

implementation and high classification accuracy. ID3 [68], C4.5 [69] and CART [70] are three widely-used decision tree algorithms. The biggest difference among them is the splitting criteria which are used to build decision trees.

3) Random Forest: A RF [71] consists of many RT. To mitigate over-fitting of RT methods and improve accuracy, the random forest method construct each RT by randomly choosing a subset of the features space . The steps to classify a new data sample by using random forest methods are:

- a) Put the data sample in each tree in the forest;
- (b) Each tree gives a classification result (vote);
- (c) The data sample will be classified into the class which has more votes.

4) Neural Network (NN): A neural network is a computing system composed by a large number of simple processing units, which operate in parallel to learn experiential knowledge from historical data [72]. Each neuron performs highly complex, nonlinear and parallel computations. In a NN, its nodes are the equivalent components of the neurons in the human brain. These nodes use activation functions to perform nonlinear computations. The most frequently used activation functions are the sigmoid and the hyperbolic tangent functions. Simulating the way neurons are connected in the human brain, the nodes in a NN are connected to each other by variable link weights. A NN has many layers. The first layer is the input layer, the last layer is the output layer and layers between them are the hidden layers. The output of each layer is the input of the next layer and the output of the last layer is the result. By changing the number of hidden layers and the number of nodes in each layer, complex models can be trained to improve the performance of NNs. NNs are widely used in many applications, such as pattern recognition. In figure 1.4 the most basic NN with three layers has been shown. An input has m features (i.e., X_1, X_2, \dots, X_m) and the input can be assigned to n possible classes (i.e., Y_1, Y_2, \dots, Y_n). Also, W_{ij}^l denotes the variable link weight between the i^{th} neuron of layer l and the j^{th} neuron of layer $l + 1$, and ak^l denotes the activation function of the k^{th} neuron in layer l . There are many types of neural networks, which can be divided in supervised or unsupervised main group [73]. In the following, we will give a brief description of supervised neural networks.

- a) Random NN:* The random NN can be represented as an interconnected network of neurons that exchange spiking signals. The main difference between

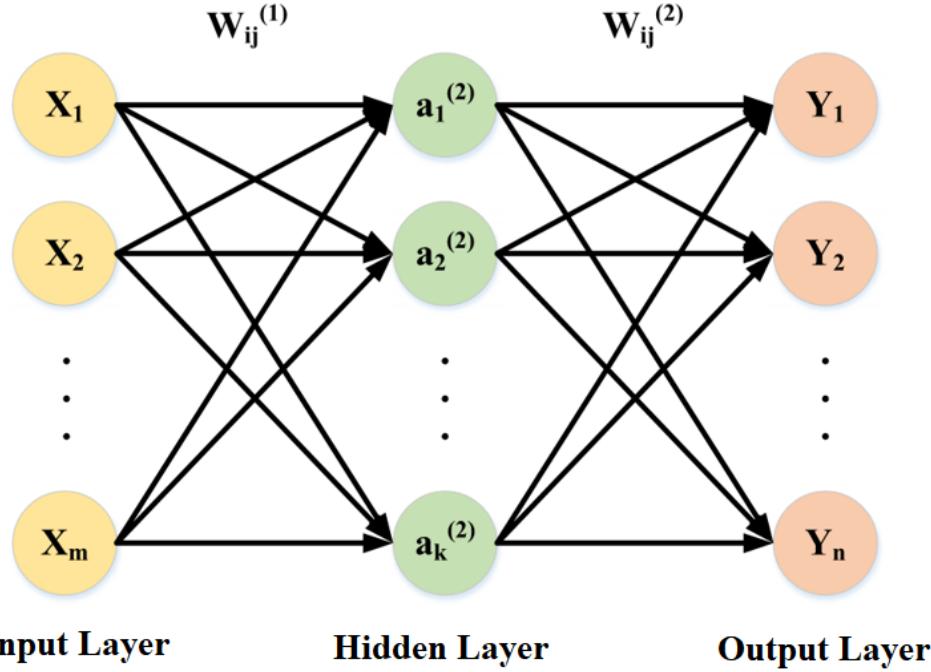


Figure 1.4: A basic neural network with three layers: an input layer, a hidden layer and an output layer.

random NN and other neural networks is that neurons in random NN exchange spiking signals probabilistically. In random NN, the internal excitatory state of each neuron is represented by an integer called “potential”. The potential value of each neuron rises when it receives an excitatory spiking signal and drops when it receives an inhibitory spiking signal. Neurons whose potential values are strictly positive are allowed to send out excitatory or inhibitory spiking signals to other neurons according to specific neuron-dependent spiking rates. When a neuron sends out a spiking signal, its potential value drops one. The random NN has been used in classification and pattern recognition [74].

b) Deep NN: Neural networks with a single hidden layer are generally referred to as shallow NNs. In contrast, neural networks with multiple hidden layers between the input layer and the output layer are called deep NNs [75, 76]. To process high-dimensional data and to learn increasingly complex models, deep NNs with more hidden layers and neurons are needed. However, deep NNs increase the training difficulties and require more computing resources. In recent years, the development of hardware data processing capabilities and the evolved activation functions make it possible to train deep NNs [77]. In deep NNs, each layer’s neurons train a feature representation based on the previ-

ous layer's output, which is known as feature hierarchy. The feature hierarchy makes deep NNs capable of handling large high-dimensional datasets. Due to the multiple-level feature representation learning, compared to other machine learning techniques, deep NNs generally provide much better performance [77].

c) Convolutional NN: Convolutional NN and recurrent NN are two major types of deep NNs. Convolutional NN [78, 79] is a feed-forward neural network. Local sparse connections among successive layers, weight sharing and pooling are three basic ideas of convolutional NN. Weight sharing means that weight parameters of all neurons in the same convolution kernel are same. Local sparse connections and weight sharing can reduce the number of training parameters. Pooling can be used to reduce the feature size while maintaining the invariance of features. The three basic ideas reduce the training difficulties of convolutional NNs greatly.

d) Recurrent NN: In feed-forward neural networks, the information is transmitted directionally from the input layer to the output layer. However, recurrent NN is a stateful network, which can use internal state (memory) to handle sequential data. Unlike a traditional deep NN, which uses different parameters at each layer, the recurrent NN shares the same parameters across all time steps. This means that at each time step, the recurrent NN performs the same task, just with different inputs. In this way, the total number of parameters needed to be trained is reduced greatly. Long Short-Term Memory (LSTM) [80] is the most commonly-used type of recurrent NNs, which has a good ability to capture long-term dependencies. LSTM uses three gates (i.e., an input gate, an output gate and a forget gate) to compute the hidden state.

5) Support Vector Machine (SVM): SVM is invented by Vapnik and others [81], which has been widely used in classification and pattern recognition. The basic idea of SVM is to map the input vectors into a high-dimensional feature space. This mapping is achieved by applying different kernel functions, such as linear, polynomial and Radial Based Function (RBF). Kernel function selection is an important task in SVM, which has effect on the classification accuracy. The selection of kernel function depends on the training dataset. The linear kernel function works well if the dataset is linearly separable. If the dataset is not linearly separable, polynomial and RBF are two commonly-used kernel functions. In general, the RBF-based SVM classifier has a relatively better performance than the other two kernel functions. The objective of SVM is to find a separating hyperplane in the feature space to maximize

the margin between different classes. The margin is the distance between the hyperplane and the closest data points of each class. The corresponding closest data points are defined as support vectors.

6) *Bayes' Theory*: Bayes' theory uses the conditional probability to calculate the probability of an event occurring given the prior knowledge of conditions that might be related to the event. The Bayes' theory is defined mathematically as the following equation:

$$P(H|E) = \frac{P(E|H)P(H)}{P(E)}$$

where E is a new evidence, H is a hypothesis, $P(H|E)$ is the posterior probability that the hypothesis H holds given the new evidence E , $P(E|H)$ is the posterior probability that of evidence E conditioned on the hypothesis H , $P(H)$ is the prior probability of hypothesis H , independent of evidence E , and $P(E)$ is the probability of evidence E . In a classification problem, the Bayes' theory learns a probability model by using the training dataset. The evidence E is a data sample, and the hypothesis H is the class to assign for the data sample. The posterior probability $P(H|E)$ represents the probability of a data sample belonging to a class. In order to calculate the posterior probability $P(H|E)$, $P(H)$, $P(E)$ and $P(E|H)$ need to be calculated first based on the training dataset using the probability and statistics theories, which is the learning process of the probability model. When classifying a new input data sample, the probability model can be used to calculate multiple posterior probabilities for different classes. The data sample will be classified into the class with the highest posterior probability $P(H|E)$. The advantage of Bayes' theory is that it requires a relatively small number of training samples dataset to learn the probability model [82]. However, there is an important independence assumption when using the Bayes' theory. To facilitate the calculation of $P(E|H)$, the features of data samples in the training dataset are assumed to be independent of each other [83].

7) *Hidden Markov Models* (HMM): HMM is one kind of Markov models. Markov models are widely used in randomly dynamic environments which obey the memoryless property. The memoryless property of Markov models means that the conditional probability distribution of future states only relates to the value of the current state and is independent of all previous states [84, 85]. There are other Markov models, such as Markov Chains (MC). The main difference between HMM and other models is that HMM is often applied in environments where system states are partially visible or not visible at all.

1.2.2 Unsupervised Learning

In contrast to supervised learning, an unsupervised learning algorithm is given a set of inputs without labels, thus there is no output. An unsupervised learning algorithm aims to find patterns, structures, or knowledge in unlabeled data by clustering sample data into different groups according to the similarity between them. The unsupervised learning techniques are widely used in clustering and data aggregation. In the following, we will give a representation of widely-used unsupervised learning algorithms.

1) *k-Means*: The k-means algorithm is used to recognize a set of unlabeled data into different clusters. To implement the kmeans algorithm, only two parameters are needed: the initial dataset and the desired number of clusters. If the desired number of clusters is k , the steps to resolve node clustering problem by using k-means algorithms are:

- a) initialize k cluster centroids by randomly choosing k nodes;
- b) use a distance function to label each node with the closest centroid;
- c) assign new centroids according to the current node memberships;
- d) stop the algorithm if the convergence condition is valid, otherwise go back to step b).

2) *Self-Organizing Map (SOM)*: SOM, also known as SelfOrganizing Feature Map (SOFM) [86], is one of the most popular unsupervised neural network models. SOM is often applied to perform dimensionality reduction and data clustering. In general, SOM has two layers, an input layer and a map layer. When SOM is used to perform data clustering, the number of neurons in the map layer is equal to the desired number of clusters. Each neuron has a weight vector. The steps to resolve data clustering problem by using SOM algorithm are:

- a) initialize the weight vector of each neuron in the map layer;
- (b) choose a data sample from the training dataset;
- (c) use a distance function to calculate the similarity between the input data sample and all weight vectors. The neuron whose weight vector has the highest similarity is called the Best Matching Unit (BMU). The SOM algorithm is based on competitive learning;
- (d) The neighborhood of the BMU is calculated;

- (e) The weight vectors of the neurons in the BMU's neighborhood are adjusted towards the input data sample;
- (f) Stop the algorithm if the convergence condition is valid, otherwise go back to step (b).

1.2.3 Semi-Supervised Learning

Semi-supervised learning is a type of learning which uses both labeled and unlabeled data. Semi-supervised learning is useful due the fact that in many real-world applications, the acquisition of labeled data is expensive and difficult while acquiring a large amount of unlabeled data is relatively easy and cheap. Moreover effective use of unlabeled data during the training process actually tends to improve the performance of the trained model. In order to make the best use of unlabeled data, assumptions have to be held in semisupervised learning, such as smoothness assumption, cluster assumption, low-density separation assumption, and manifold assumption. Pseudo Labeling [87] is a simple and efficient semi-supervised learning technique. The main idea of Pseudo Labeling is simple. Firstly, use the labeled data to train a model. Then, use the trained model to predict pseudo labels of the unlabeled data. Finally, combine the labeled data and the newly pseudo-labeled data to train the model again. There are other semi-supervised learning methods, such as Expectation Maximization (EM), co-training, transductive SVM and graph-based methods. Different methods rely on different assumptions. For example, EM builds on cluster assumption, transductive SVM builds on low-density separation assumption, while graph-based methods build on the manifold assumption.

1.2.4 Reinforcement Learning

Supervised learning algorithms are generally applied to conduct classification and regression tasks, while unsupervised and reinforcement learning algorithms are applied to conduct clustering and decision-making tasks respectively.

I) Reinforcement Learning (RL): RL [88, 89] involves an agent, a state space S and an action space A . The agent is a learning entity which interacts with its environment to learn the best action to maximize its long-term reward. The long-term reward is a cumulative discounted reward and relates to both the immediate reward and future rewards. When applying RL to SDN, the controller generally works as an agent and the network is the environment. The controller monitors the network status and learns to make decisions to control data forwarding. Specifically, at each time step

t , the agent monitors a state s_t and chooses an action a_t from the action space A , receives an immediate reward r_t which indicates how good or bad the action is, and transitions to the next state s_{t+1} . The objective of the agent is to learn the optimal behavior policy π which is a direct map from the state space S to the action space $A(\pi : S \rightarrow A)$ to maximize the expected long-term reward. From the behavior policy π , the agent can determine the best corresponding action given a particular state. In RL, value function is used to calculate the long-term reward of an action given a state. The most well-known value function is Q-function, which is used by Q-learning to learn a table storing all state-action pairs and their long-term rewards.

2) *Deep Reinforcement Learning (DRL)*: The main advantage of RL is that it works well without prior knowledge of an exact mathematical model of the environment. However, the traditional RL approach has some shortcomings, such as low convergence rate to the optimal behavior policy π and its inability to solve problems with high-dimensional state space and action space. These shortcomings can be addressed by DRL. The key idea of DRL is to approximate the value function by leveraging the powerful function approximation property of deep NNs. After training the deep NNs, given a state-action pair as input, DRL is able to estimate the long-term reward. The estimation result can guide the agent to choose the best action.

3) *RL-Based Game Theory*: Game theory is a mathematical tool that focuses on strategic interactions among rational decision-makers. A game generally involves a set of players, a set of strategies and a set of utility functions. Players are decision-makers. Utility functions are used by players to select optimal strategies. In cooperative games, players cooperate and form multiple coalitions. Players choose strategies that maximize the utility of their coalitions. In non-cooperative games, players compete against each other and choose strategies individually to maximize their own utility. In the network field, it is often assumed that nodes are selfish. In non-cooperative games, players do not communicate with each other, and at the beginning of each play round, players do not have any information about the strategies selected by the other players. At the end of each play round, all players broadcast their selected strategies, which are the only external information. However, each player's utility can be affected by the other players' strategies. In this case, adaptive learning methods should be used to predict the strategies of the other players, based on which each player chooses its optimal strategy. RL is a widely-used adaptive learning method,

which can help players select their optimal strategies by learning from historical information such as network status, the other players' strategies and the corresponding utility. Thus, RL-based game theory is an effective decision-making technique.

1.3 Related Works

The problems of modeling, managing and optimizing resources in a heterogeneous communication network is a very challenging engineering problem because of its inherent complexity [1, 2, 90, 91].

Numerous studies have been conducted to maximize the performance of the controller and OpenFlow switch of SDN, however, few results and methodologies exist to model and perform optimal control of SDN switches with priority scheduling. When analyzing the literature regarding traffic management in SDN and priority queueing, we can distinguish three main different approaches.

Heuristic approaches, where algorithms to both identify and control traffic within a queue are based on rule-of-thumbs and empirical approaches that do not take into account any particular model: in [92] a heuristic method is proposed to balance the packet load among queues in order to reduce packet losses, which does not aim at providing an optimal solution; in [93] authors provide a scheduling algorithm for handling the incoming data traffic by enqueueing packets into the corresponding queue based on priority and High Priority queue is dequeued first; in [94] the authors define multiple queues with different priority classes, which are used to prioritize VoIP packet based on delay, and the controller decides where to enqueue the packet based on delay and considering 5 different decision thresholds.

Parametric approaches, where the control of queues is based on less heuristics and more objective methods. More precisely, one or more parameters that describe the QoS of an SDN are chosen and optimization is performed based on *static* models characterised by such parameters: both in [95] and [96] the authors consider different approaches to model and control queuing delays with specific network parameters; in [97] QoE is taken into account in the context of VOIP and the decision metric for selecting the best link for establishing a new VoIP call is based on the MOS quality metric, which is a typical measure of the level of a user's satisfaction of the quality of a call. These approaches despite the fact that are easy to understand and to implement, may not be often suitable to describe and control traffic flows in large and complex networks as they are not based on a *dynamical* network model.

Model-based approach, where a *dynamical* mathematical model of packet flows within a queue is considered, is the one most related to the research conducted in this thesis. In the classical literature of queuing theory, in particular applied to SDN, most of the approaches are based on classic structures for models [98] and many techniques are exploited to estimate the parameters and the state of a queue [99]. In [100], the authors emphasized that switch performance depends on multiple factors such as: flow-table size, packet arrival rate, etc., and they took these key factors into account for the design of their M/Geo/1 system where the arriving packets follow a Poisson distribution and the service times follows a Geometric distribution; in [101], beside describing a comprehensive review of the literature (mostly M/M/k and M/G/k), authors derived a new model for a queueing network, based on Quasi Bird Death (QBD) processes; another approach based on a dynamical model for Model Predictive Control is described in [102], where the authors derive a Discrete Time Markov Jump Linear System to model a queueing network with the aim of defining predictive control policies based on MPC; finally, in [103] the author proposes a new congestion control algorithm based on MPC, called MPAQM, where the queue length is predicted based on the extended TCP/AQM system model and a state estimator. The main drawback of the approach proposed by the authors here is that they linearize and discretize the model of a TCP/AQM interconnection system as illustrated in [104]; Nonlinear MPC can be applied, but the problem is that the resulting optimization problem can be nonconvex and so hard to solve. In such scenarios, linearization is a solution but not always a good solution because of the fact that linearize a model of a complex system not always ensure adequate control performance, especially when the system is going to operate far from of the linearization point.

Obviously the most interesting are the Model-based approaches. However, their main drawback is related to the identification of the model. One issue is related to the need of having access to the queue's buffer data to identify the model and, at least with commercial hardware, this is not possible in general. A second issue is that classical models, such as the ones previously discussed, are usually designed to provide best accuracy for one step prediction. When applying MPC one wants to forecast and exploit the value of state variables for multiple steps ahead: using classical approaches this is achieved by using the prediction computed at $k + 1$ to predict the state at $k + 2$, and so on. This approach is not always accurate when a long prediction horizon is considered, since many additional issues arise such as error propagation and increased uncertainty. In such situations, a multiple output strategy where a model is able to directly predict the state at different future time steps, or one model for each time step as we will propose in this work, can increase the MPC performance. Of course this comes with additional computational complexity, especially

when the number of time steps to be forecasted increases: however, as will be shown later on, this is not an issue in our methodology since the future predictions can be computed exploiting the advantages of binary decision trees and parallel computation.

The last and most important issue is that the mathematical models proposed in such literature do not allow to directly exploit MPC methods with, simultaneously, good accuracy and a realistically implementable computational complexity, i.e. using Quadratic Programming (QP) solvers. Tackling such research challenges is the main topic of this thesis. Indeed, to the best of the author's knowledge, the state of the art in deriving accurate dynamical models of communication networks still lacks of methods that exploit historical network data to learn (identify) a dynamical network model that can be directly used for optimal control (e.g. of segment routing and/or queue management) and is practical from the computational complexity point of view [1, 2, 105, 106, 107, 108, 109]. This manuscript provides a novel methodology to fill this gap.

In this scenario, computing technologies such as graphic processing and tensor processing units represent a good opportunity to implement advanced control theoretic (e.g. MPC) and machine learning algorithms (e.g. decision trees, deep neural networks, etc.) in the communication networks [110, 111, 11, 112]. In summary, the real-time programmability of SDN controllers and the availability of massive historical data enable the exploitation of data analysis and optimization techniques for improving networks efficiency and performance.

Other Machine Learning (ML) approaches applied to telecommunication networks has been investigated over the years. A Knowledge Plane (KP) approach [113] has been proposed to enable automation, recommendation and intelligence by applying ML and cognitive techniques. However the KP approach has not been prototyped nor deployed because each node of traditional network systems, such as routers or switches, can only view and act over a small portion of the system. This implies that each node can learn only from a (small) part of the complete system and therefore it is very complex to design control algorithms beyond the local domain [114]. Thanks to SDN the network resources are managed by a logically centralized controller that have a global view of the network [3, 13, 46, 115, 116]. This feature provides the capacity of collecting data on the network state and opens the possibility of improving the characteristics of each network device with ML algorithms.

Indeed, the most difficult challenge to be addressed in order to apply optimization techniques is to derive a predictive model of the queues of the switch behaviour [117, 118, 119, 120, 121]. On this line of research, Cello *et al.* provide in [122] a predictive model for estimating QoS in order to detect the need for a re-routing strategy due to link saturation. However, this framework cannot be used to apply traffic optimization techniques. In [123]

an initial effort is conducted to derive a general hybrid systems framework to model the flow of traffic in communication networks. In [124] the authors provide a first formulation and implementation, based on hybrid systems theory, of a mathematical and simulative environment to formally model the effect of router/link failures on the dynamics of TCP and UDP packet flows belonging to different end-user services (i.e. http, ftp, mailing and video streaming). However, even though hybrid systems are very effective in modelling a network of routers, using such framework for implementing traffic optimization is out of question for computational complexity issues. A further research question focuses on designing strategies for periodic updating of network models, in order to maintain good performance despite the evolution of the real system [125].

Deriving accurate dynamical models of communication networks that can be directly used for optimal control by the exploitation of historical network data is still a challenging and actual problem [105, 106, 107, 108, 109].

Recent advances in computing technologies such as Graphics Processing Unit and Tensor Processing Unit provide a good opportunity to apply promising machine learning techniques (e.g., deep neural networks) in the network field [110, 111]. Data is the key to the data-driven machine learning algorithm: the centralized SDN controller has a global network view, and is able to collect various network data. The potential impact of machine learning in networks is evident from the huge literature on the topic: Patcha and Park [126] have given a detailed description of machine learning techniques in the domain of intrusion detection; Nguyen and Armitage [127] focus on IP traffic classification; Bkassiny et al. [128] have surveyed existing machine learning based methods in Cognitive Radio Networks; [129] investigated how machine learning techniques can be applied in wireless sensor networks; Wang et al. [130] have presented the state-of-the-art on Artificial Intelligence based techniques applied to evolve heterogeneous networks and discussed future research challenges; Buczak and Guven [131] investigated data mining methods for cyber-security intrusion detection; Klaine et al. [132] have surveyed machine learning algorithms for self organizing cellular networks; [133] investigated how to improve network traffic control using machine learning techniques; Hodo et al. [134] focus on machine learning based Intrusion Detection System; Zhou et al. [135] focus on cognitive radio technologies enforced by machine learning techniques to enhance spectrum utilization and energy efficiency of wireless networks; Chen et al. [136] have studied neural networks solutions applied in wireless networks for virtual reality and edge caching; Usama et al. [111] have applied unsupervised learning techniques in the general domain of networking. Although machine learning techniques have been widely investigated in the communication scientific community, to the best of our knowledge no existing work focuses on the applications

of machine learning and control theory for identifying models of network devices in the domain of Software Defined Network (SDN), with the aim of efficiently apply Model Predictive Control.

Chapter 2

RT- and RF-based models of SDN switched for Priority Queueing

This chapter first describes a SDN Mininet emulation environment that we use to generate traffic packets (starting from real data patterns) and to retrieve network (feedback) information via the SDN controller, thanks to the OF protocol. Then we shown how to use data collected from such environment to create a predictive model of the switch priority queueing scheuling.

2.1 Mininet network emulation environment and control problem

The Mininet environment [137] has been used to emulate a SDN network to validate our methodology in terms of prediction accuracy and control performance. This software runs a collection of virtual network elements (i.e. end-hosts, switches, routers, and links) on a single Linux kernel using lightweight virtualization. To generate traffic we used the D-ITG generator [138, 139, 140].

For the purposes of this work, various network configurations were tested. Since similar results has been obtained on all configurations, it is possible to consider the generic case as the architecture in Figure 2.1, which aims to represent a portion of a larger network where a bottleneck occurs. More precisely, we consider a switch s_0 with one input port and one output port, and a remote controller [14, 20] that dynamically manages the configuration of the queues of s_0 . The input of s_0 is fed with an instance of D-ITG generating stochastic traffic, whose mean value follows the pattern of a real data set (where packets are differentiated by their ToS - Type of Service - priority index) extracted from two days logs of a router of a large service provider network. Namely, the original real data set contains

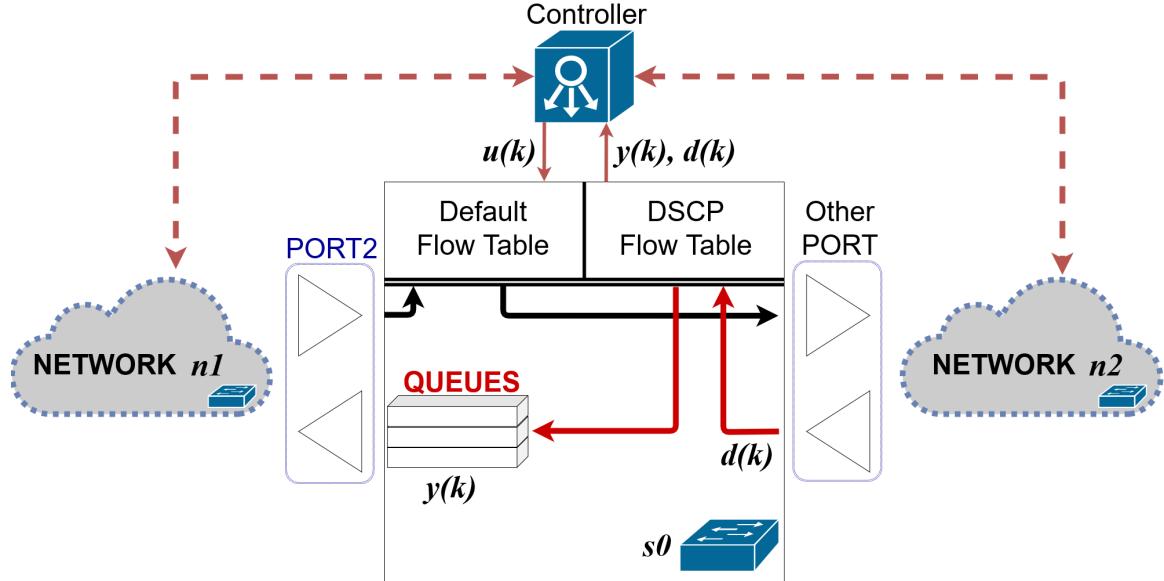


Figure 2.1: Mininet emulated network architecture.

traffic of a real network incoming from a source geographic area and terminating in a destination geographic area, and is divided for each value of Differentiated Services Code Point (DSCP) with a sampling time of 5 minutes [141, 142]. We recall that DSCP is the modern definition of the Type of Service (ToS) field, in which the first 6 bits are the Differentiated Services field that are in common with ToS field, and the last 2 bits regard explicit congestion notification. The ToS field can specify the priority of a datagram and the request for a low delay addressing, a high throughput or a high reliability service. Following the implementation of many national service provider networks (see e.g. [143]), we partition the 8 different values of the DSCP in three classes: the *Default* class (DSCPs 0, 1, 3), the *Premium* (DSCPs 2, 4, 6, 7), and the *Gold* class (DSCP 5): to each class we will assign a single queue, associated with a different priority.

Using D-ITG Sender and Receiver SW modules it has been possible to establish a connection between networks n_1 and n_2 . In particular, 16 ITG modules have been initialized: 8 for each network, and within each network one for each DSCP index. These modules handle the sampling time interval (5 minutes), the inter-departure time stochastic distribution associated with the packet rate, the packet size stochastic distribution, the IP and port destinations, and the DSCP index. Regarding the controller SW module we used Ryu, which provides software components with well defined Application Programming Interfaces (API) that give the possibility to easily create new network management and control applications. Ryu supports various protocols for managing network devices, such as OpenFlow, Netconf, OF-config, etc. About OF, Ryu supports fully 1.0, 1.2, 1.3, 1.4, 1.5 and

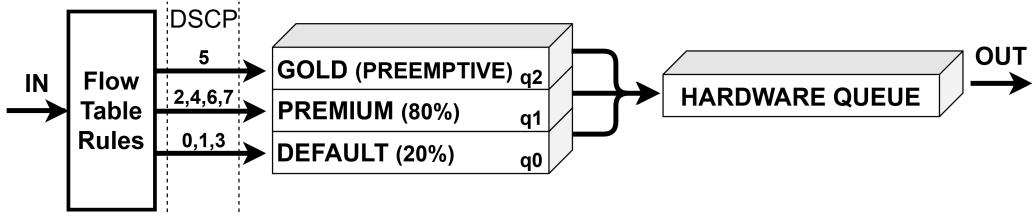


Figure 2.2: Static queues rate with routed packets relative to DSCP.

Nicira Extensions. For our test-bed the 1.3 version has been chosen. In particular, APIs were used for queue control and counter recovery from the switches [144, 145]. The feedback information collected for the purposes of this work are the descriptions of switches, ports and queues, the number of packets received and transmitted on each port of a switch, the packets passing through the flow tables, the packet rate values of each queue and the packets transmitted by each single queue. In summary, the variables associated to the traffic and control signals in our closed-loop architecture are as follows:

- $d(k) \in \mathbb{R}^{10}$ is a measurable disturbance vector, i.e. representing variables we cannot control. The first 8 components $d_1(k), \dots, d_8(k)$ consist of the number of packets incoming in the switch s_0 differentiated with respect to the 8 different values of the DSCPs. $d_9(k)$ and $d_{10}(k)$ are proxy variables, i.e. the hours and minutes of the day, which are very useful to the predictive model since traffic dynamics are tightly correlated with them, e.g. they are substantially different between night and day;
- $y(k) \in \mathbb{R}^3$ is the measured output vector, i.e. the variables we want to regulate. They consist of the number of packets outgoing from switch s_0 differentiated with respect to the corresponding service class: $y_1(k)$ is the Default Queue output, $y_2(k)$ is the Premium Queue output and $y_3(k)$ is the Gold Queue output;
- $u(k) \in \mathbb{R}^3$ is the control input vector. Each component corresponds to the queue configuration of each service class: $u_1(k)$ is the Default Queue configuration, i.e. the maximum admitted bandwidth; $u_2(k)$ is the Premium Queue configuration, i.e. the maximum admitted bandwidth; $u_3(k)$ is the Gold Queue configuration, i.e. the minimum admitted bandwidth;

In this work we first applied in our emulative scenario the static control of queues used in the Italian service provider network of *Telecom Italia* [143], which is depicted in Figure 2.2. To this aim we defined 3 queues in s_0 and configured the queues as follows: packets with the DSCP values 0, 1 and 3 (Default queue) are routed via queue 0, with maximum rate $u_1(k) = 20MB/s, \forall k$; the packets with values 2, 4, 6 and 7 (Premium queue) are routed

on queue 1, with maximum rate $u_2(k) = 80MB/s, \forall k$; the packets with value 5 (Gold queue) are routed on queue 2, with minimum rate $u_3(k) = 100MB/s, \forall k$. To obtain this prioritization it has also been necessary to set the flow tables of $s0$ to discriminate incoming packets based on the DSCP value and the destination IP address, and re-route them to the desired queue. Also, to obtain a bottleneck situation in $s0$, we have chosen the bandwidth of the output port of switch $s0$ at $100 MB/s$. Using this configuration queue 2 uses the maximum capacity of the port to forward packets with preemptive priority, while the other two queues use the remaining bandwidth from $0 MB/s$ to the specified maximum bandwidth based on needs. To instantiate the chosen topology in Mininet it has been necessary to run the A.8 code from the Ubuntu command line with the following text:

```
sudo python Topology_qos.py
```

In this script all network devices (remote controller, switches and hosts), their attributes and their connections are defined. Furthermore this script is also responsible for the traffic generated between hosts thanks to the function defined within the A.9 code that is iteratively used every five minutes to read the provided real data traffic database and generate the synthetic traffic inside the network. In lines 49 to 53 of A.8 code Mininet adds an external controller, Ryu in our case, to specified IP address and port. Obviously it is necessary that the controller is already instantiated before launching the network topology, and it is possible to do this through the following command:

```
ryu run main_controller_TOS.py rest_qos.py rest_conf_switch.py ofctl_rest.py
```

This command executes several scripts, the first one (A.1) is the Ryu controller which invokes additional functions designed for saving network information and setting queue bandwidths (from A.2 to A.4). The remaining scripts are all the rest APIs required for these features to work properly (A.5, A.6 and A.7 codes).

As we will see in Section 2.3, using static priority control the queues will not be able to send all the packets incoming from network $n1$, and a dramatic amount of packets will be lost. This motivates the application of optimization techniques, which are enabled by the predictive models derived using the methodology described in section 2.2.

2.2 Regression Trees and Random Forest based models for MPC

In this section a methodology to apply the results proposed in [146, 147] is illustrated, to identify, starting from a set of collected historical data $\mathcal{D} = \{y(k), u(k), d(k)\}_{k=0}^{\ell}$ (generated as described in the previous section), a switching ARX model of the input-output

behavior of the traffic flow in a switch of a SDN network as follows:

$$x(k+j+1) = A'_{\sigma_j(x(k), d(k))} x(k) + \sum_{\alpha=0}^j B'_{\sigma_j(x(k), d(k)), \alpha} u(k+\alpha) + f'_{\sigma_j(x(k), d(k))}, \quad (2.1)$$

$j = 0, \dots, N-1$, where $x(k) \doteq [y^\top(k) \cdots y^\top(k-\delta_y) u^\top(k-1) \cdots u^\top(k-\delta_u)]^\top \in \mathbb{R}^{n_x}$ is an extended state to characterize a switching ARX model, with $x_\iota(k) \doteq [y_\iota(k) \cdots y_\iota(k-\delta_y) u^\top(k-1) \cdots u^\top(k-\delta_u)]^\top \in \mathbb{R}^{\delta_y+1+3\delta_u}$, $\iota = 1, 2, 3, N$ is the chosen future predictive horizon, and $\sigma_j : \mathbb{R}^{n_x+10} \rightarrow \mathcal{M} \subset \mathbb{N}$ is a switching signal that associates an operating mode in a finite set \mathcal{M} to each pair $(x(k), d(k))$ and each prediction step j of the horizon. It is possible to directly use model (2.1) to setup the following problem, which can be solved using standard Quadratic Programming (QP) solvers:

Problem 1

$$\begin{aligned} & \underset{u_0, \dots, u_{N-1}}{\text{minimize}} && \sum_{j=0}^{N-1} \left((x_{j+1} - x_{\text{ref}})^\top Q (x_{j+1} - x_{\text{ref}}) + u_j^\top R u_j \right) \\ & \text{subject to} && x_{j+1} = A'_{\sigma_j(x_0, d_0)} x_0 + \sum_{\alpha=0}^j B'_{\sigma_j(x_0, d_0), \alpha} u_\alpha + f'_{\sigma_j(x_0, d_0)} \\ & && u_j \in \mathcal{U} \\ & && x_0 = x(k), d_0 = d(k) \\ & && j = 0, \dots, N-1. \end{aligned}$$

As it is well known [148], Problem 1 is solved at each time step k using QP to compute the optimal sequence u_0^*, \dots, u_{N-1}^* , but only the first input is applied to the system, i.e. $u(k) = u_0^*$. Note that, for any prediction step j , x_{j+1} only depends on the measurements $x_0 = x(k), d_0 = d(k)$ at time k , since they are the only available measurements at time-step k .

2.2.1 RT and RF background

Let us consider a dataset $\{y(k), x_1(k), \dots, x_\eta(k)\}_{k=0}^\ell$, with $y, x_1, \dots, x_\eta \in \mathbb{R}$. Let us suppose to estimate, using Regression Trees, the prediction of the (response) variable $y(k)$ using the values of predictor variables $x_1(k), \dots, x_\eta(k)$. The CART algorithm [149] creates a RT structure via optimal partition of the dataset. It solves a Least Square problem by optimally choosing recursively a variable to split and a corresponding splitting point. After several steps the algorithm converges to the optimal solution, and the dataset is partitioned in hyper-rectangular regions (the leaves of the tree) R_1, R_2, \dots, R_ν . In each partition $y(k)$

is estimated with a different constant \hat{y}_i $i = 1, \dots, \nu$, given by the average of the samples of $y(k)$ falling in R_i , i.e.

$$\hat{y}_i = \frac{\sum_{\{k|(x_1(k), \dots, x_\eta(k)) \in R_i\}} y(k)}{|R_i|} \quad (2.2)$$

Random Forests [150] are instead an averaging method that exploits a combination of tree predictors such that each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest. The output prediction is given by averaging the predictions provided by all trees in the forest. It is possible to show that the error introduced by the forests quickly and almost surely converges to a limit as the number of trees in the forest becomes large. Such error also depends on the strength of the individual trees in the forest and the correlation between them. Thus, due to the Law of Large Numbers, Random Forests (differently from Regression Trees) do not suffer much variance and overfitting problems. For more details the reader is referred to [149, 150].

2.2.2 Switching ARX (SARX) model identification via RT

To derive a model as in (2.1), a new dataset $\mathcal{X} = \{x(k), u(k), d(k)\}_{k=0}^\ell$ has to be defined starting from \mathcal{D} . In order to apply MPC it is needed, for each component of $y(k)$, a model that can predict system's dynamics over a horizon of length N . The idea is to create $3N$ predictive trees $\{\mathcal{T}_{\iota,j}\}$, $\iota = 1, 2, 3$, $j = 0, \dots, N - 1$, each one to predict the 3 outputs components of the system over the N steps of the horizon. To create the tree structure, the RT algorithm (CART) partitions the dataset \mathcal{X} into regions \mathcal{X}_i , such that $\biguplus \mathcal{X}_i = \mathcal{X}$, and assigns to each region a constant value given by the average of the output values of the samples that ended up in that leaf. In run-time, once the trees are created, and given a real-time measurement $(x(k), u(k), d(k))$ belonging to leaf i , the CART algorithm provides as a prediction the averaged value associated to the leaf as in (2.2). However, since the prediction provided by the RT is a constant value, it cannot be used to setup an MPC problem, thus the learning procedure needs to be modified to identify a modeling framework as in (2.7). To this end, \mathcal{X} is partitioned in two disjoint sets $\mathcal{X}_c = \{u(k)\}_{k=0}^\ell$ of data associated to the control variables, and $\mathcal{X}_{nc} = \{(x(k), d(k))\}_{k=0}^\ell$ of data associated to remaining variables, and then apply the CART algorithm only on \mathcal{X}_{nc} (this is to avoid that the MPC problem turns out into a Mixed Integer Quadratic Program, see [146, 147] for details); thus, $3N$ RTs $\{\mathcal{T}_{\iota,j}\}$ have been created, each constructed to predict the variable $y_\iota(k + j + 1)$, and

consequently $x_\iota(k + j + 1)$. In particular, it is associated to each leaf ι, i_j , corresponding to the partition $\mathcal{X}_{nc,\iota,i_j}$, of each tree $\mathcal{T}_{\iota,j}$ the following affine model is associated

$$x_\iota(k + j + 1) = A'_{\iota,i_j} x(k) + \sum_{\alpha=0}^j B'_{\iota,i_j,\alpha} u(k + \alpha) + f'_{\iota,i_j}, \quad (2.3)$$

$$A'_{\iota,i_j} = \begin{bmatrix} a_1 & a_2 & \cdots & a_{\delta_y} & a_{\delta_y+1} & b_{\delta_y+2} & \cdots & b_{\delta_y+1+3(\delta_u-1)} & \cdots & b_{\delta_y+1+3\delta_u} \\ 1 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & 0 & \cdots & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 1 & 0 & 0 & \cdots & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 0 & 1 & \cdots & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 1 & \cdots & 0 \end{bmatrix},$$

$$B'_{\iota,i_j,\alpha} = \begin{bmatrix} b_{1,\alpha} & b_{2,\alpha} & b_{3,\alpha} \\ 0 & 0 & 0 \\ \vdots & \vdots & \vdots \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ \vdots & \vdots & \vdots \\ 0 & 0 & 0 \end{bmatrix}, \alpha < j, B'_{\iota,i_j,j} = \begin{bmatrix} b_{1,\alpha} & b_{2,\alpha} & b_{3,\alpha} \\ 0 & 0 & 0 \\ \vdots & \vdots & \vdots \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ \vdots & \vdots & \vdots \\ 0 & 0 & 0 \end{bmatrix}, f'_{\iota,i_j} = \begin{bmatrix} f \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad (2.4)$$

where the coefficients of matrices A'_{ι,i_j} , $B'_{\iota,i_j,\alpha}$ and f'_{ι,i_j} are obtained in each leaf ι, i_j by fitting the corresponding set of samples solving the following Least Squares with inequality constraints problem:

Problem 2

$$\begin{aligned} & \underset{\xi_{\iota,i_j}}{\text{minimize}} \quad \| \Lambda_{\iota,i_j} \xi_{\iota,i_j} - \lambda_{\iota,i_j} \|_2^2 \\ & \text{subject to } f_{\min} \leq f \leq f_{\max} \\ & \quad a_{\min} \leq a_j \leq a_{\max} \\ & \quad b_{\min} \leq b_{\iota,\alpha} \leq b_{\max}, \end{aligned} \quad (2.5)$$

where ξ_{ι,i_j} , λ_{ι,i_j} , and Λ_{ι,i_j} contain respectively the parameters of matrices in (2.4), the predictions $x_\iota(k + j + 1)$, and the current measurements of $x(k)$ and $u(k + \alpha)$. Linear

inequalities (2.5) are used to constrain elements in ξ_{ι,i_j} to take into account physical system constraints and improve prediction accuracy. Model (2.3) can be easily compacted in the following form taking into account all the states $\iota = 1, 2, 3$:

$$x(k+j+1) = A'_{i_j} x(k) + \sum_{\alpha=0}^j B'_{i_j,\alpha} u(k+\alpha) + f'_{i_j}. \quad (2.6)$$

In particular, with the specific choice of $\delta_u = 0$ model (2.1) can be rewritten in the following state-space formulation

$$x(k+j+1) = A_{\sigma_j(x(k), \mathbf{u}^-(k), d(k))} x(k+j) + B_{\sigma_j(x(k), \mathbf{u}^-(k), d(k))} u(k+j) + f_{\sigma_j(x(k), \mathbf{u}^-(k), d(k))}, \quad (2.7)$$

where $\mathbf{u}^-(k) = [u^\top(k-1) \cdots u^\top(k-\delta)]^\top$ is the vector with the regressive terms of the input used to only grow the trees, and $\sigma_j : \mathbb{R}^{3(\delta_y+1)+3\delta+10} \rightarrow \mathcal{M} \subset \mathbb{N}$. Thanks to this new formulation the following proposition shows that model (2.6) is equivalent to model (2.7) for any initial condition, any switching signal and any control sequence.

Proposition 1 [147] *Let A'_{i_j} , $B'_{i_j,\alpha}$ and f'_{i_j} , $\alpha = 0, \dots, j$, $j = 0, \dots, N-1$, be given. If A'_{i_j} is invertible for $j = 0, \dots, N-1$, then there exists a model in the form*

$$\bar{x}(k+j+1) = A_{\sigma_j(\bar{x}(k), \mathbf{u}^-(k), d(k))} \bar{x}(k+j) + B_{\sigma_j(\bar{x}(k), \mathbf{u}^-(k), d(k))} u(k+j) + f_{\sigma_j(\bar{x}(k), \mathbf{u}^-(k), d(k))}$$

such that for any initial condition $\bar{x}(k) = x(k) = x_k$, any switching sequence i_0, \dots, i_{N-1} and any control sequence $u(k), \dots, u(k+N-1)$, then $\bar{x}(k+j+1) = x(k+j+1)$, $\forall j = 0, \dots, N-1$.

As discussed in [147], from experimental findings it is possible to notice that the contribution in terms of model accuracy introduced by the choice of $\delta_u = 0$ is negligible, since previous control inputs are already considered in the tree structure choosing $\delta > 0$. Thus, in the rest of the thesis it will be considered $\delta_u = 0$ and $\delta > 0$, i.e. only the regressive terms of the input in the tree structure learning will be used and not in the state definition.

2.2.3 SARX model identification via RF

RF-based models can be constructed exploiting the RT-based formulation: in particular, let us consider $3N$ RFs $\mathcal{F}_{\iota,j}$, $\iota = 1, 2, 3$, $j = 0, \dots, N-1$. For each tree $\mathcal{T}_\tau^{\mathcal{F}_{\iota,j}}$ of the forest $\mathcal{F}_{\iota,j}$, it is possible to estimate the coefficients a_* , b_* and f in (2.4) for each leaf ι, j, i_τ , i.e. $\tilde{\xi}_{\iota,j,i_\tau}$, solving Problem 2. With a small abuse of notation, let us indicate by $|\mathcal{F}_{\iota,j}|$ the

number of trees in the forest ι, j . Then $\forall \iota, j$, the parameters to build matrices in (2.9) can be obtained by averaging parameters a_* , b_* and f , $\forall \tau = 1, \dots, |\mathcal{F}_{\iota,j}|$, i.e.

$$\tilde{\xi}_{\iota,j} = \frac{\sum_{\tau=1}^{|\mathcal{F}_{\iota,j}|} \tilde{\xi}_{\iota,j,i_\tau}}{|\mathcal{F}_{\iota,j}|}, \quad (2.8)$$

over all the trees of forest $\mathcal{F}_{\iota,j}$. At this point, starting from (2.3), it can be easily construct the following model, as in (2.6) to be used in the MPC formulation by combining for $\iota = 1, 2, 3$ the matrices in (2.4) coming either from the RTs or from the RFs:

$$x(k+j+1) = A'_{ij}x(k) + \sum_{\alpha=0}^j B'_{ij,\alpha}u(k+\alpha) + f'_{ij}. \quad (2.9)$$

2.2.4 MPC problem formulation.

Model (2.9) is used to formalize Problem 1 according to the SDN priority queueing problem:

Problem 3

$$\begin{aligned} & \underset{\mathbf{u}}{\text{minimize}} \quad \sum_{j=0}^{N-1} \left[(x_{j+1} - x_{\text{ref},j})^\top Q (x_{j+1} - x_{\text{ref},j}) + u_j^\top R u_j \right] \\ & \text{subject to} \quad x_{j+1} = A_{\sigma_j(k)}x_j + B_{\sigma_j(k)}u_j + f_{\sigma_j(k)} \\ & \quad \Delta u_{\iota}^{\min} \leq u_{\iota,j} - u_{\iota,j-1} \leq \Delta u_{\iota}^{\max} \\ & \quad u_{\iota}^{\min} \leq u_{\iota,j} \leq u_{\iota}^{\max} \\ & \quad u_{1,j} + u_{2,j} \leq 100 \\ & \quad x_0 = x(k), \quad \mathbf{u}_0^- = [u^\top(-1) \cdots u^\top(-\delta)]^\top, \quad d_0 = d(k), \\ & \quad j = 0, \dots, N-1, \quad \iota = 1, 2, 3, \end{aligned}$$

where $\sigma_j(k) = \sigma_j(x(k), \mathbf{u}^-(k), d(k))$ (with a slight abuse of notation), $u_{\iota,j}$ is the ι^{th} component of the input u at time $k+j$; Δu_{ι}^{\min} and Δu_{ι}^{\max} are used to limit large variations on the inputs in 2 consecutive steps, in order to avoid that the queues drastically set to the minimum value and thus potentially increase packet losses during the next period; u_{ι}^{\min} and u_{ι}^{\max} define the bandwidth limits induced by the QoS requirements of the corresponding priority class. At each time step k the measurements of the variables in \mathcal{X}_{nc} are collected, select the current matrices of (2.9) narrowing down the leaves of the trees, for $j = 0, \dots, N-1$, solve Problem (3), and finally apply only the first input of the optimal sequence \mathbf{u}^* found, i.e. $u(k) = u_0^*$.

2.2.5 Disturbance forecast

The knowledge at each time k of the future input traffic ($d(k+1), \dots, d(k+N-1)$) can greatly improve the MPC performance. However, while the future values of the proxy variables (hours and minutes) are clearly well known, the knowledge of the future values of the first 8 component of the disturbance, i.e. number of packets incoming in the switches for each DSCP index are unknown at the current instant k . To address this problem $8(N-1)$ RFs $\mathcal{F}_{\iota,j}^d$, $\iota = 1, \dots, 8$, $j = 0, \dots, N-1$ have been built in order to provide a prediction $\hat{d}_\iota(k+j)$ of the 8 disturbance components $d_\iota(k+j)$ over the future time horizon: as widely illustrated in [146, 147] the technique previously described can be easily modified by appropriately redefining the dataset as $\mathcal{X} = \{(x(k), u(k), d(k), \dots, d(k+N-1))\}_{k=1}^\ell$ for the training phase, and considering a switching signal in (2.7) given by $\sigma_j(k) = \sigma_j(x(k), \mathbf{u}^-(k), d(k), \hat{d}(k+1), \dots, \hat{d}(k+j)), \forall j = 0, \dots, N-1$, i.e. also depending at time k on the future input traffic.

2.3 Simulation results

In this section simulation results of the application of the proposed approach to SDN Priority Queueing identification and control will be provided. Standard RFs are used to derive predictive models of the disturbance components $d_1(k), \dots, d_8(k)$, i.e. the switch input differentiated for each DSCP index, and validate the accuracy. Then the validation of accuracy of the predictive model of the output variable $y(k)$ derived as illustrated in Section 2.2 is shown: the predictive models (based on RTs and RFs) will be compared with Artificial Neural Networks, showing that RFs represent the ideal solution both in terms of prediction accuracy and computational complexity; then the effect of iterative dataset updates in the prediction accuracy is illustrated, both with and without prediction of the future disturbances. Finally the proposed predictive models will be used to setup a MPC problem (see Problem 3), and validate the control performance in terms of packet losses reduction and bandwidth saving, both with and without prediction of the future disturbances. It will also be shown, as expected, that using accurate predictive models and applying MPC provides dramatic reduction of packet losses and increase of bandwidth saving with respect to the static bandwidth allocation policy used in Service Provider Networks as described in Section 2.1: even though this result is not surprising, it is decided to quantify the gap to emphasize that much better performance can be obtained in real networks just collecting historical data and applying a controller that can be directly implemented using the accurate models of the proposed identification algorithm and Quadratic Programming solvers (which are well known to be very efficient).

In each of the aforementioned validations, 4 different predictive models have been exploited, using iteratively enriched data sets. More precisely, **OLD** is a predictive model identified with a data set of 5124 samples, collected with a sampling time of 5 minutes and obtained from network emulation with random values of the input $u(k)$; **1UP** is a predictive model identified with the **OLD** data set enriched with 3456 new samples obtained from network emulation when applying closed-loop MPC to define the input $u(k)$; **2UP** is a predictive model identified with the **1UP** data set enriched with 3168 new samples obtained from network emulation when applying closed-loop MPC to define the input $u(k)$; **3UP** is a predictive model identified with the **2UP** data set enriched with 6336 new samples obtained from network emulation when applying closed-loop MPC to define the input $u(k)$. An independent data-set composed by 1684 samples is used to validate the above models. All simulations have been ran on a UDOO x86 Advanced with an Intel Braswell N3160 processor up to 2.24 GHz and 4 GB of RAM [151].

2.3.1 Disturbance predictive model validation

Having an accurate model of the variable $d(k)$ (i.e. the switch input differentiated for each DSCP index) can be helpful to improve the model identification performance as well as the reference input x_{ref} to follow in Problem 3. In this section we apply standard RF algorithms, with a regressive index of 15 steps and 30 trees for each forest, to obtain a predictive model of the disturbance over a predictive horizon of $N = 5$ (25 minutes): this choice of N has been taken considering the tradeoff between time complexity of the identification algorithm and the obtained identification accuracy.

Figure 2.3 shows the Normalised Root Mean Square Error (NRMSE) of the predictive model of the disturbance signals (one for each of the 8 DSCP indices) over a time horizon of $N = 5$: the prediction error is worse for Service 0 (4 – 6%) since it includes the majority of the packets that transit through the switch. For other services the NRMSE is at most 2.2% (Service 7) over all the predictive horizons. The improvement of the model accuracy when using larger (updated) data sets is evident, until a *saturation* is reached and further data do not help to improve the model accuracy: the NRMSE significantly reduces and for Service 0 it is even halved. Figure 2.4 plots, for Service 0 and in a time window of 500 samples (almost two days), the predictions of OLD, 1UP, 2UP and 3UP as well as the original data, and clearly highlights the better prediction of 2UP and 3UP with respect to OLD and 1UP.

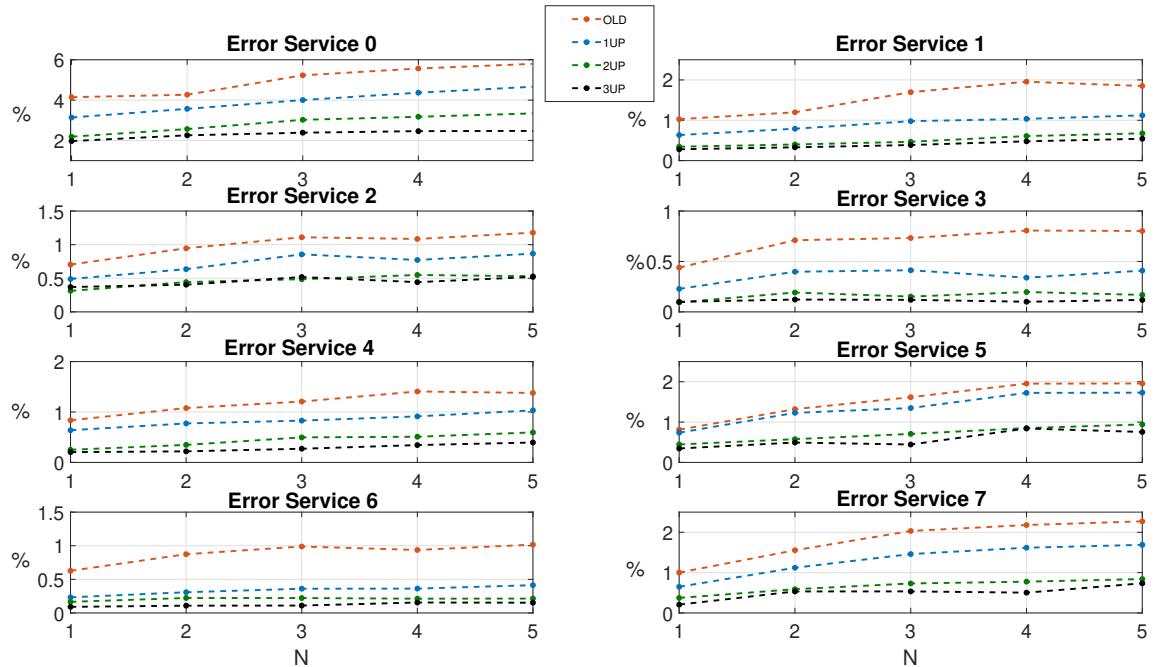


Figure 2.3: NRMSE of the disturbance predictive model over a time horizon of $N = 5$.

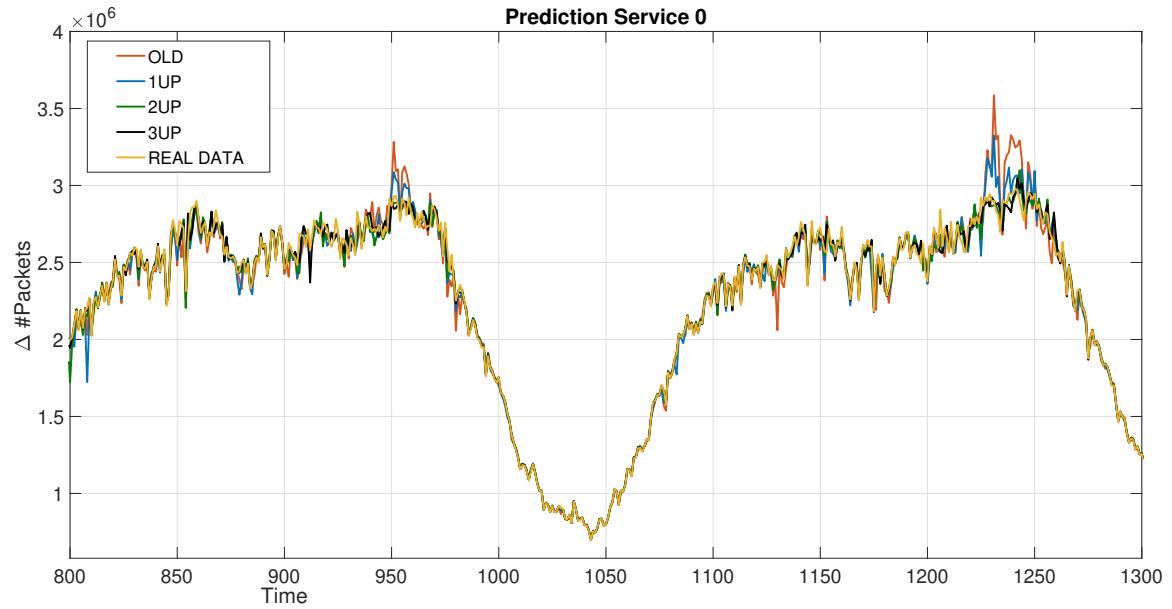


Figure 2.4: Comparison between the real traffic (YELLOW LINE) and the traffic prediction for the different models for Service 0.

2.3.2 Queues predictive model validation

In this section we first compare the accuracy of our predictive models with Artificial Neural Networks. We recall that a neural network is a collection of algorithms that aim to identify underlying relations in a dataset: it consists of groups of connected neurons organized in layers, where the connections between neurons are modeled using weights. The signal produced with this linear composition is then fed into an activation function that is in general nonlinear. The reader is referred to [152] and references therein for more details. A wide number of tools to build Neural Networks have been developed during recent years, e.g. [153, 154, 155] just to mention a few: in this work we exploit the Deep Learning Toolbox of Matlab to compare predictive models based on NNs with the methodology proposed in this work, based on ARX combined RTs and RFs. We consider here just OLD as the learning dataset and chose a predictive horizon $N = 5$.

To identify a RT (resp. RF) based predictive model of the queues we trained a Regression Tree (resp. a Random Forest) for each output and for each time horizon, with a total of 15 trees (resp. 15 forests each consisting of 30 trees). The main parameters used for the identification algorithm (see Section 2.2 and Problem 2) are summarized in Table 2.1. In particular, the regressive terms (δ_d , δ_x , δ_u) and the minimum number of samples

Table 2.1: Identification parameters

Parameters	Value	Parameters	Values
N	5	f_{min}	-100
ν	1	f_{max}	100
δ_x	5	a_{min}	-100
δ_u	5	a_{max}	100
δ_d	5	b_{min}	0
Minleaf	13	b_{max}	10000
$ \mathcal{F}_{ij} $	30		

for each tree of each forest (MinLeaf) have been chosen, with a trial and error approach, considering that very small regressive horizons and very large values for MinLeaf may lead to inaccurate prediction (as they do not provide sufficient information on the past) but very large regressive horizons and very small values for MinLeaf also lead to inaccurate prediction (as they interpolate very old data that might negatively affect the results and produce overfitting).

Regarding specific parameters used for running NN, and for the sake of a fair comparison, we tuned them to obtain the best performance: in particular we considered shallow networks of 2 layers since deeper networks did not improve the accuracy and, instead,

have the negative effect of increasing the sensitivity of the accuracy with respect to the initial conditions of the weights. Among the many algorithms for optimizing the weights of the neurons we exploited the *Scaled conjugate gradient back-propagation* described in [156], which provided the best accuracy with respect to our dataset. Regarding the activation functions, we used both the classical sigmoid function (*LogSig*) and the Hyperbolic tangent sigmoid transfer function (*TanSig*).

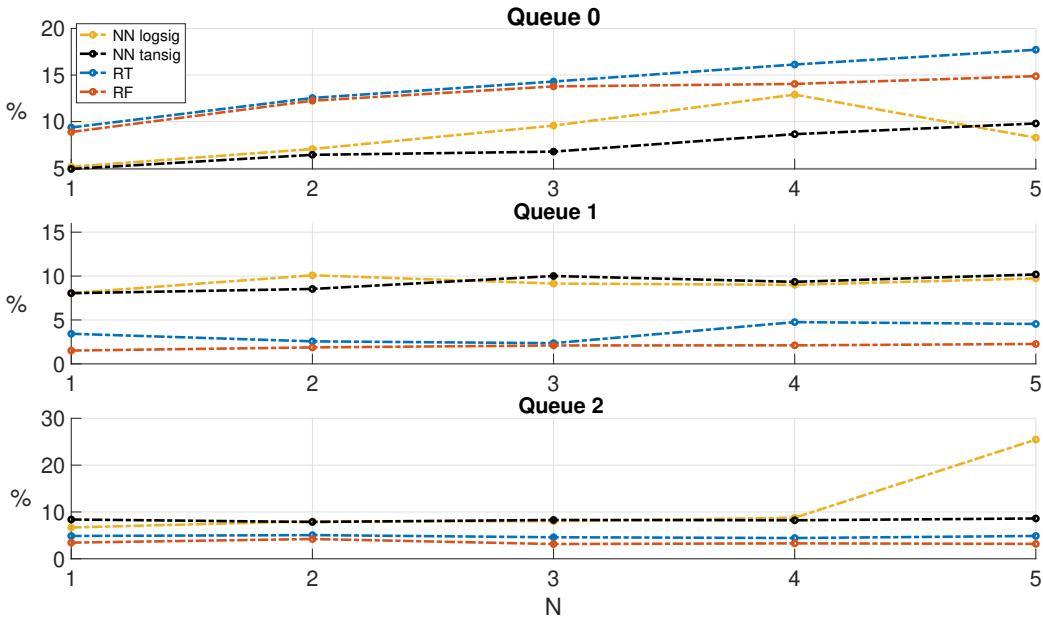


Figure 2.5: NRMSE, up to $N = 5$ and for each priority class, for RT (blue), RF (red), NN with sigmoids as activation function (yellow) and NN with hyperbolic tangent as activation function (black).

As a metric of the prediction accuracy we compared in Figure 2.5 the Normalized Root Mean Square Errors (NRMSE) of the different identification approaches for each priority class and over a horizon up to $N = 5$. Regarding queue 0 (Default) NNs perform better than RT and RF, but in queues 1 (Premium) and 2 (Gold), characterised by higher priority, RF provides the best performance. Queue 0 is characterised by a larger NRMSE with all identification techniques: this is due to the fact that, having the lowest priority, it suffers more packet losses and this can negatively affect the prediction accuracy. Our validation emphasizes that RTs, even though very simple and fast to compute, are often affected by overfitting and variance issues, i.e. small variations of the training data result in large variations of the tree structure and, consequently, of the predictions. Regarding NNs, they provide a less accurate model in 2 cases over 3. Indeed, by analyzing the dataset distribution (see Figure 2.6), we noticed a peculiar regular grid pattern that can be very well approxi-

mated by hyper-rectangles: since RTs and RFs base their prediction on hyper-rectangular dataset partitions, the better performance with respect to NNs is reasonable. For queue 0,

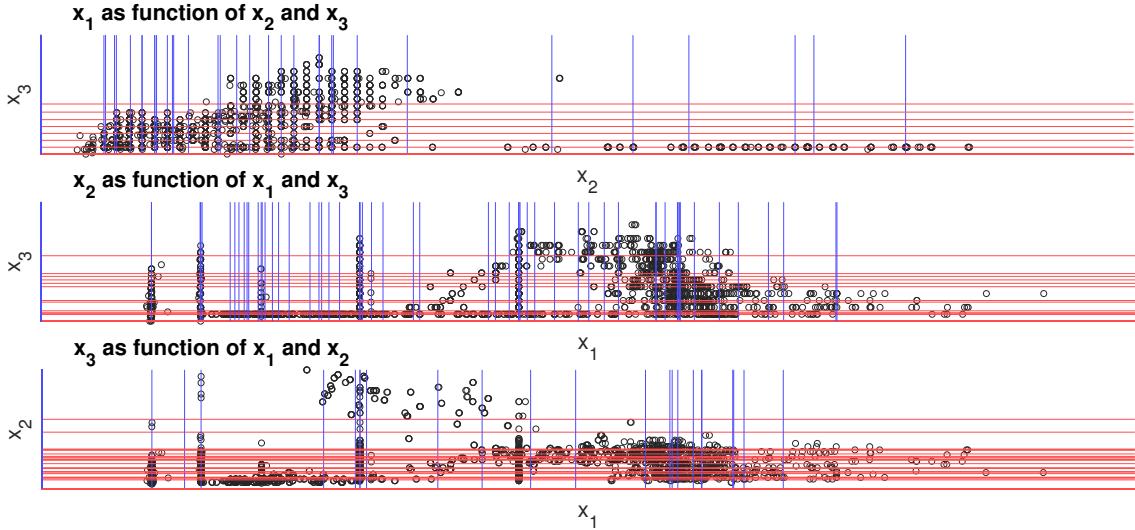


Figure 2.6: Grid pattern emerged from dataset distribution.

even though NNs perform better, we need to remark that their predictive model is based on nonlinear functions: this makes the derived model impractical for real-time control as the corresponding MPC formulation turns into a nonlinear optimization problem, for which there is no approach that can guarantee neither a global optimal solution nor a reasonable computation time. In addition to this, even obtaining a closed mathematical form of the predictive function of a Neural Network starting from neurons and weights is not always an easy task, because of the highly nonlinear interconnections between the different layers. For all these reasons we decided to only use from now on RF-based models, which provide the best choice both from the accuracy and the computational complexity points of view. In the following we illustrate the effect of iterative dataset updates in the prediction accuracy, both with and without knowledge of the future disturbances.

Figure 2.7 and Figure 2.8 plot the NRMSEs respectively without and with knowledge of the future disturbances. The assumption of future disturbance forecast, as expected, provides much better prediction accuracy. The positive effect of updated data sets is also clear, providing accuracy improvements up to 50%: as will be also discussed in the next section, the most relevant prediction accuracy improvement takes place moving from OLD to 1UP or from 1UP to 2UP, while the 3UP model does not improve much.

Remark 1 We wish to highlight that in our simulations we generated data without major modifications of the traffic daily pattern: for this reason enriching the data set converges

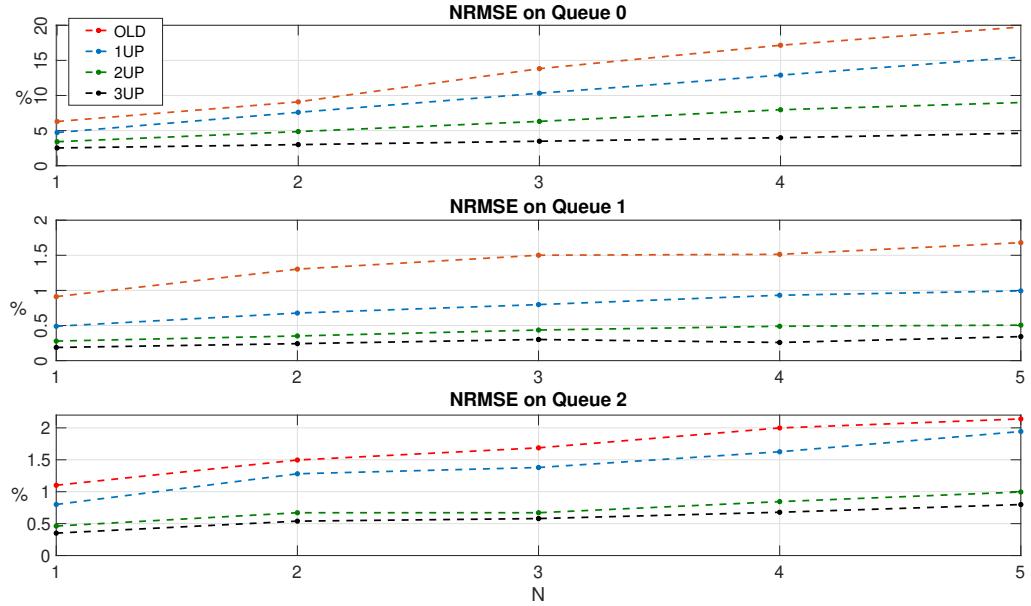


Figure 2.7: NRMSE of the queues output predictive model over a time horizon of $N = 5$, without knowledge of the future disturbances

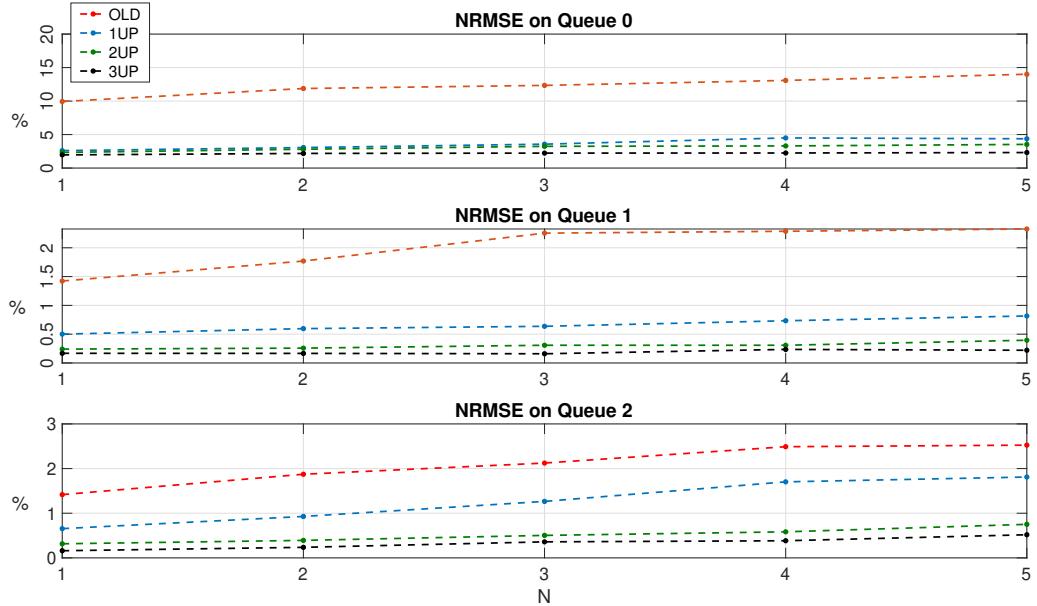


Figure 2.8: NRMSE of the queues output predictive model over a time horizon of $N = 5$, with knowledge of the 4-steps future disturbances

to a saturation of the model accuracy, as discussed above. Nevertheless, the capability of our methodology to iteratively learn from new data is fundamental as, in real life, changes

in the traffic patterns do occur; and model updates are necessary to maintain the model accuracy and the control performance.

2.3.3 Control performance

In this section we setup a control loop where the (Mininet) network emulator and the (Ryu) controller run in two different computers, and synchronize/exchange data using a shared file. Namely, our SW controller module is, in principle, ready to be directly used on a real SDN-based network, with just some minor modifications in the data exchange with the switch devices. The controller implements MPC using the predictive models validated in the previous sections: at each time step, it solves Problem 3 and optimally updates the bandwidth of the different queues. The cost matrices Q and R of Problem 3 respectively weight the output $y(k)$ of the system (i.e. the packet transmission rate for each queue) and the control input $u(k)$ (i.e. the bandwidth assigned to each queue). Since R is required to be positive definite but it makes no sense assigning a penalty to the choice of $u(k)$, we define $R = 10^{-5} \cdot \mathbb{I}$, where the identity matrix \mathbb{I} multiplies a very small value. Matrix $Q = \text{diag}(1, 10^4, 10)$ has been assigned as a diagonal matrix, where the choice of the different diagonal components is related to the priority level of each queue. The remaining constraints of Problem 3 are reported in Table 2.2. In what follows we validate the control

Table 2.2: Constraints in Problem 3

Parameters	Value	Parameters	Values
Δu_1^{\min}	1	Δu_1^{\max}	30
Δu_2^{\min}	20	Δu_2^{\max}	30
Δu_3^{\min}	20	Δu_3^{\max}	20
u_1^{\min}	10	u_1^{\max}	45
u_2^{\min}	55	u_2^{\max}	80
u_3^{\min}	80	u_3^{\max}	100

performance both without and with knowledge of the future disturbances. The value of x_{ref} in the optimization problem represents the reference value we chose for tracking system output: indeed, as we wish to minimize packet losses, we minimize the difference between the packets received by the hosts $d(k)$ and those transmitted by the queues $y(k)$ over the horizon N . In case we have no knowledge of future disturbances, we consider x_{ref} equal to the current disturbance measurement $d(k)$ and constant over all the predictive horizon; if instead we have knowledge of future disturbances, we consider x_{ref} equal to such future disturbances. In this section we decided to only compare models OLD, 1UP and 2UP, since model 3UP does not provide any substantial improvement.

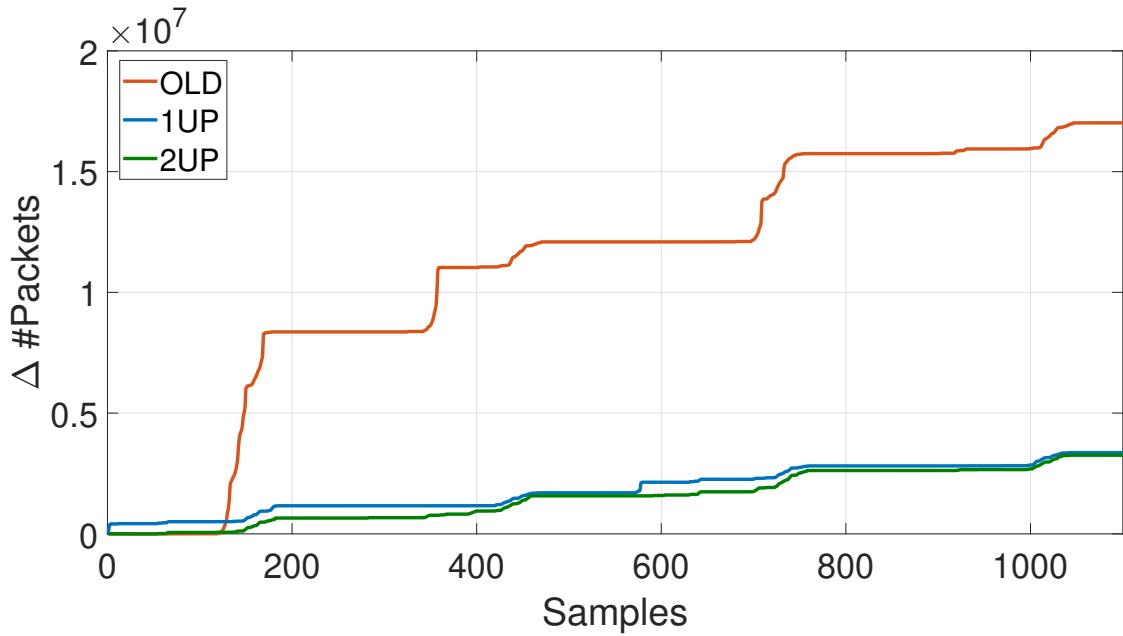


Figure 2.9: Cumulative Packet Losses without knowledge of the future disturbance.

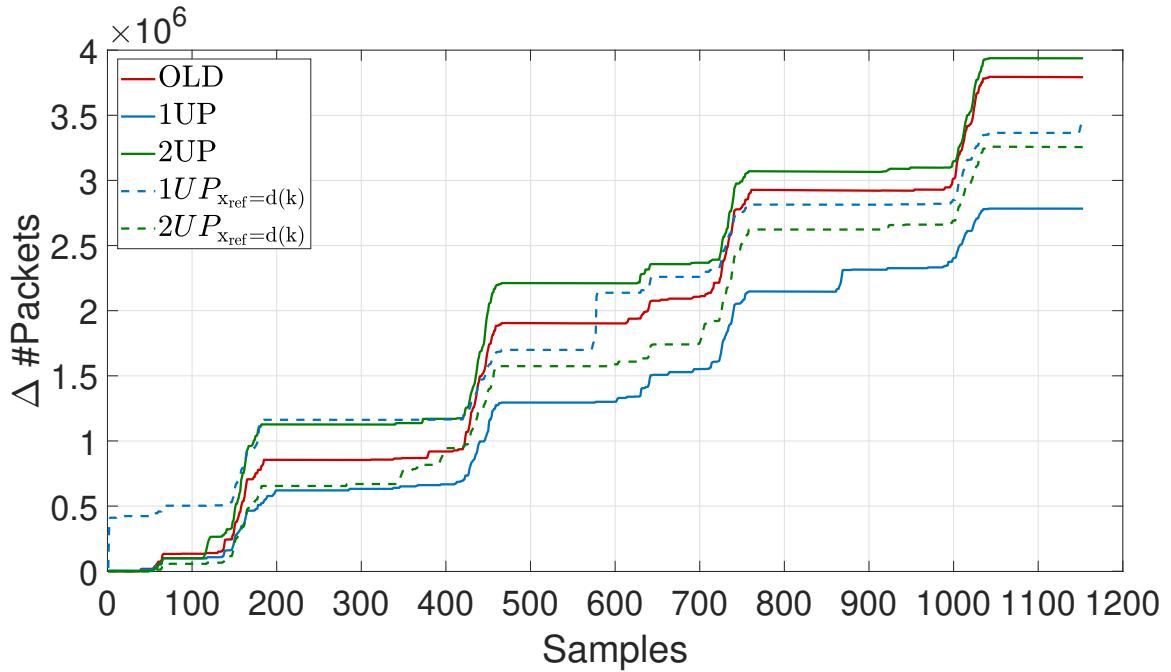


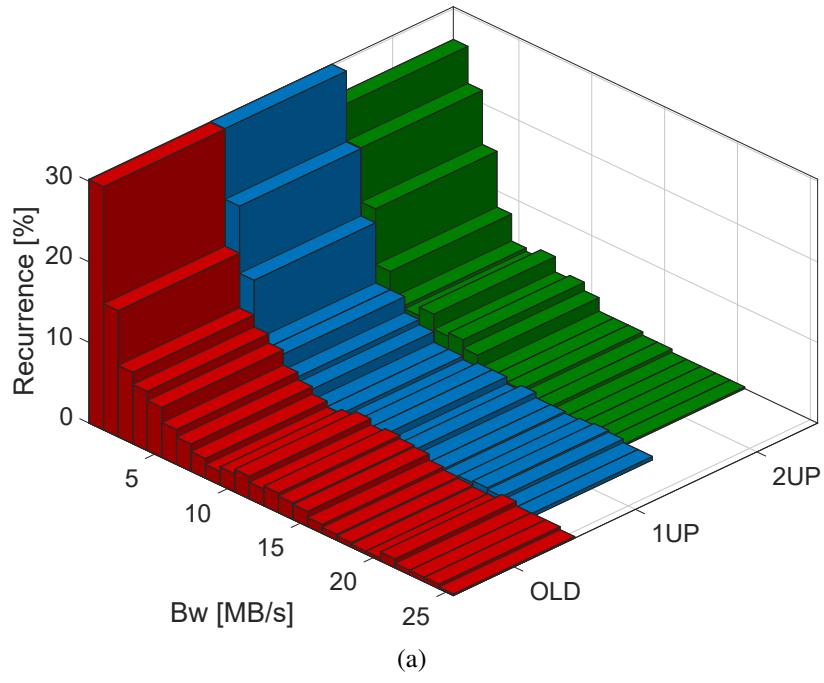
Figure 2.10: Comparison between Cumulative Packet Losses with (solid lines) and without (dashed lines) knowledge of the future disturbance.

Figures 2.9 and 2.10 plot the cumulative packet losses respectively without and with knowledge of the future disturbances. The packet loss rate when the control is performed exploiting the OLD model and without disturbance forecast is around 123% larger than all

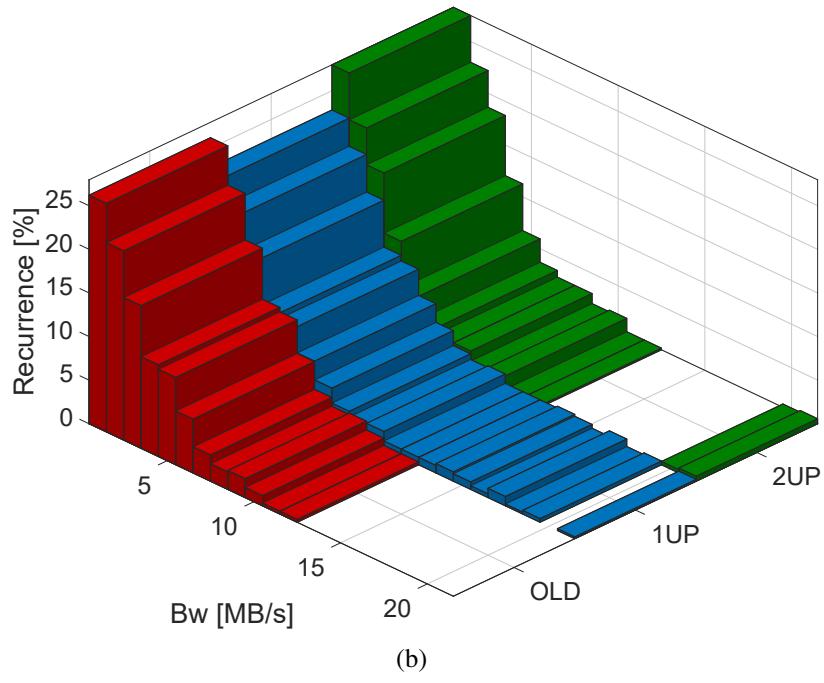
other cases (and, of course, incomparably smaller than the static control case [143]). It is also clear from the plots that 1UP and 2UP without disturbance forecast and OLD, 1UP and 2UP with disturbance forecast provide very similar performance. Our interpretation is that OLD models without disturbance forecast have not enough information to provide good accuracy, but they can be easily improved either with a data set update (which however requires 10 days for 1UP and 20 days for 2UP of additional data) or using a predictive disturbance model.

Figure 2.11 illustrates the bandwidth savings showing the recurrence of the different bandwidth usage during the simulations, respectively without and with knowledge of the future disturbances. Without disturbance forecast we exploited up to $25MB/s$ using the OLD model, while we exploited at most $22MB/s$ and $21MB/s$ respectively for models 1UP and 2UP. Using disturbance forecast, as expected, even less bandwidth is exploited.

We conclude this thesis by quantifying the gap between priority queueing control performance of MPC, obtained solving Problem 3 and based on our RF predictive model, with the static control policy adopted by service provider networks in [143]. Figure 2.12 highlights the dramatic improvement of MPC with respect to static control: the red line shows the incoming traffic, the blue line shows the sum of the packets sent from the queues, and their difference represents packet losses. Until the 400th static control has been implemented as in [143], generating many packet losses due to queues saturation. From that sample to the end of our experimentation we implemented MPC using our RF-based model, drastically reducing packet losses: quantitatively, after 700 sampling periods the cumulative number of dropped packets with the static policy is about $5.5 \cdot 10^8$ versus $6.6 \cdot 10^6$ with MPC, with a decrease of $5.434 \cdot 10^8$ lost packets (-88%). We remark that, even though the improvement of MPC with respect to static control is not surprising, much better performance can be obtained in real networks just collecting historical data and applying a controller that can be directly implemented using the accurate models of our identification algorithms and Quadratic Programming standard solvers.



(a)



(b)

Figure 2.11: Bandwidth saving comparison without (a) and with (b) knowledge of the future disturbances.

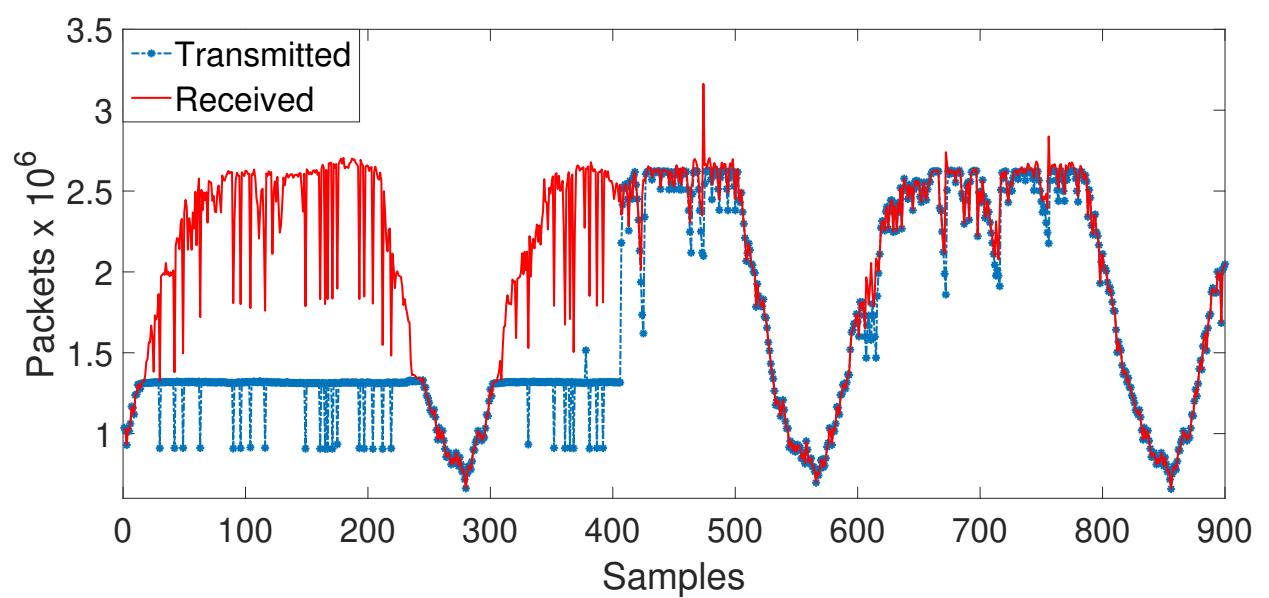


Figure 2.12: Static controller up to the 400th, then MPC controller.

Chapter 3

RT- and RF-based predictive models of multi-service traffic in a real Service Provider Network

This chapter shows how to apply the methods presented in Chapter 2 to derive a predictive model of multi-service traffic in a real network with dedicated hardware. We finally validate the performance of the derived predictive models on the traffic obtained from an Italian internet service provider (Sonicate S.r.l.).

3.1 Control performance validation over dedicated hardware network

Given the limitations of the Mininet software, it has been decided to test the previously explained algorithms on a real network. Due the fact that the SDN network devices are much expensive, it has been choosen to take an intermediate step in which each device is associated with a dedicated hardware (unlike Mininet, where multiple virtual devices shared single hardware resorces). The used architecture is shown in Figure 3.1, where two Raspberry pi 3 [157] have been used has hosts connected on a single switch. The SDN switch resides in a third Raspberry, where the firmware of an OVS has been installed and two ethernet/usb adapter have been used as ethernet ports (eth1 and eth2) to connect the hosts. The built in ethernet port (eth0) has been used to connect the SDN controller (RYU) to the switch. An Arduino UNO with a DS1307 module has been used to share single time schedule from the controller to the hosts. By using the python script in B.6 it has been possible to iteratively update the time on DS1307 module every hour, then each raspberry can update its internal time by asking the time to module via I^2C serial communication. Every update is performed in different minutes of the same hour to avoid

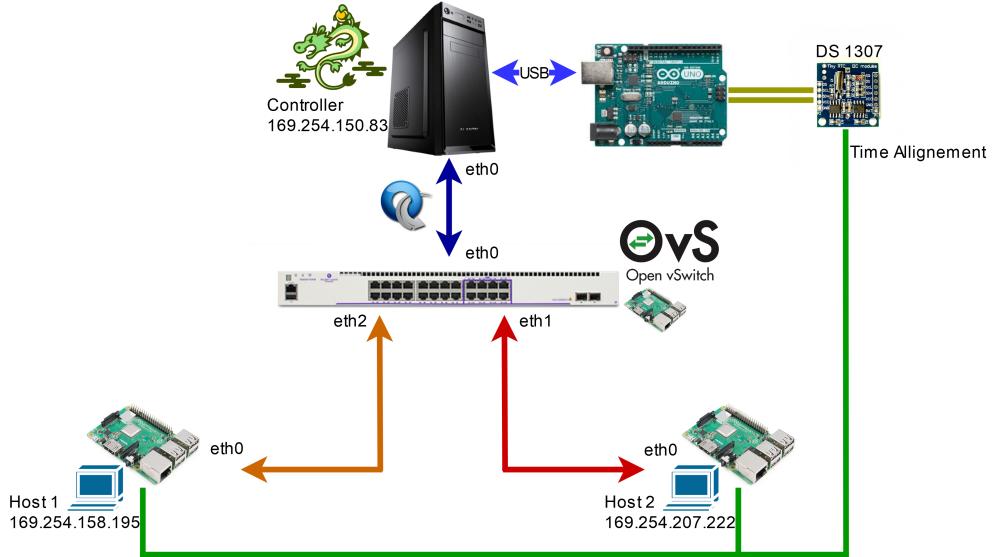
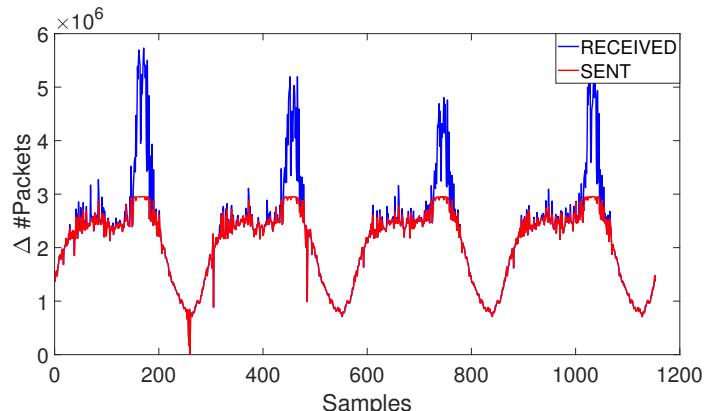


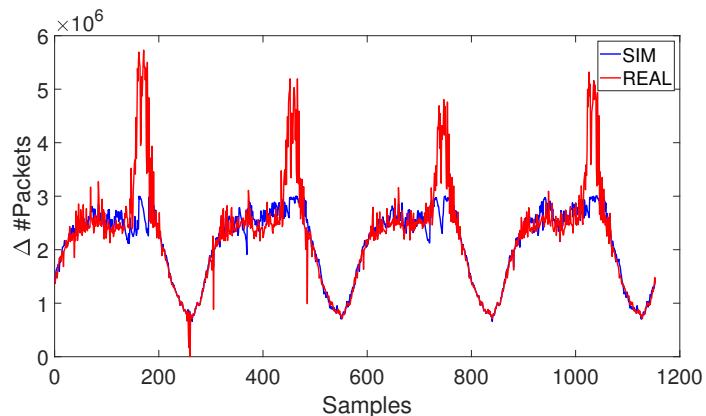
Figure 3.1: Network architecture with dedicated hardware devices.

collisions caused by multiple requests. This time sharing has been necessary to ensure time synchronization between traffic generation and data harvesting. In a real network the time synchronization is ensure to internet time sharing. Even in this environment a Ryu controller has been used as SDN controller with the same Python script previously described for the Mininet network (e.g. A.2, A.1 and A.3). The custom flow table rules are set after the deployment of the controller and after the join of the switch by a single run of the code in B.4. With this operation we have the default set up as in section 2.1 (e.g. three queues on port 2 with a bandwidth of $100Mbps$ and 20% of port bandwidth for queue 0, 80% of port bandwidth for queue 1 and 100% of port bandwidth for queue 2). While for the Mininet emulator we had a single code in which both the network topology and the traffic generation of each host were present (e.g. A.8), in the case of dedicated hardware it was necessary to write a new Python script that included the generation and reception of traffic and infinitely run it on the two Raspberry, which act as hosts. The code B.1 contains two different threads: the first one is very similar to the part of code contained in A.8 for the generation of the traffic, the only difference is the IP address of the hosts. The second thread is very simple and is needed to start the DITG Receiver module on each host. After some tests it has been possible to see that in the hours characterized by a larger number of packets incoming in the switch the computational capacity of a Raspberry pi 3 is not sufficient to elaborate all the necessary operations to route the packets to the hosts. Thus it has been decided to send packets only from *Host2*, with IP address 169.254.207.222, to *Host1*, with IP address 169.254.158.195. With this traffic generation no anomaly was found on the information obtained from the OF counters as showed in

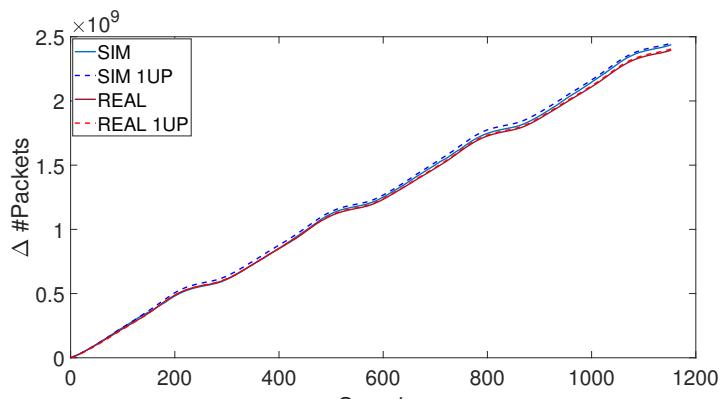
Figure 3.2(a). This figure shows packets coming into the switch generated by *Host2* (blue line) and packets outgoing from port 2 of the switch to *Host1* (red line), in case the MPC control policies explained in Section 2.2 are applied with the same mathematical model used in Section 2.3.3. This figure makes it clear that the traffic generation is different than the traffic generated by virtual hosts in Mininet. In particular, in the last hours of the day it is possible to notice a larger traffic peak than in the emulation environment, as shown in Figure 3.2(b). This is probably caused by sharing multiple network devices on a single hardware, which decreases the performance of each network element. With this traffic generation differences is difficult to compare the results obtained from those environment. Due the chosen limitations of the queues bandwidth it has been possible to see in Figure 3.2(c) that the amount of packets transmitted from the controlled port is about the same between the two network setups, even after one model update like the one explained in Section 2.3. This last very important result demonstrates how it is possible to use a virtual network to train a mathematical model aimed to controlling some device features and then apply this controller to a likely real network while maintaining the same traffic profile. Furthermore, another interesting aspect that emerges from these data is that even after an update of the control model, therefore after adding a traffic profile different from the one already present, the overall behavior of the queues remains practically unchanged. This set of results paves the way for experimentation on real networks.



(a)



(b)



(c)

Figure 3.2: Packets that transit inside the switch (a), comparison between packets incoming in the switch in emulation environment and in dedicated hardware (b) and sum of packets sent by the switch port with different control model (c).

3.2 Traffic predictive model validation on Italian Internet provider network

In addition to the validation of our predictive models of the incoming traffic over the Mininet environment and dedicated hardware devices network, the accuracy has been also tested on data measured from a real network device (Ubiquiti EP-16) of an Italian internet provider (Sonicatel S.r.l.). Data collection has been performed using the software Cacti [158].

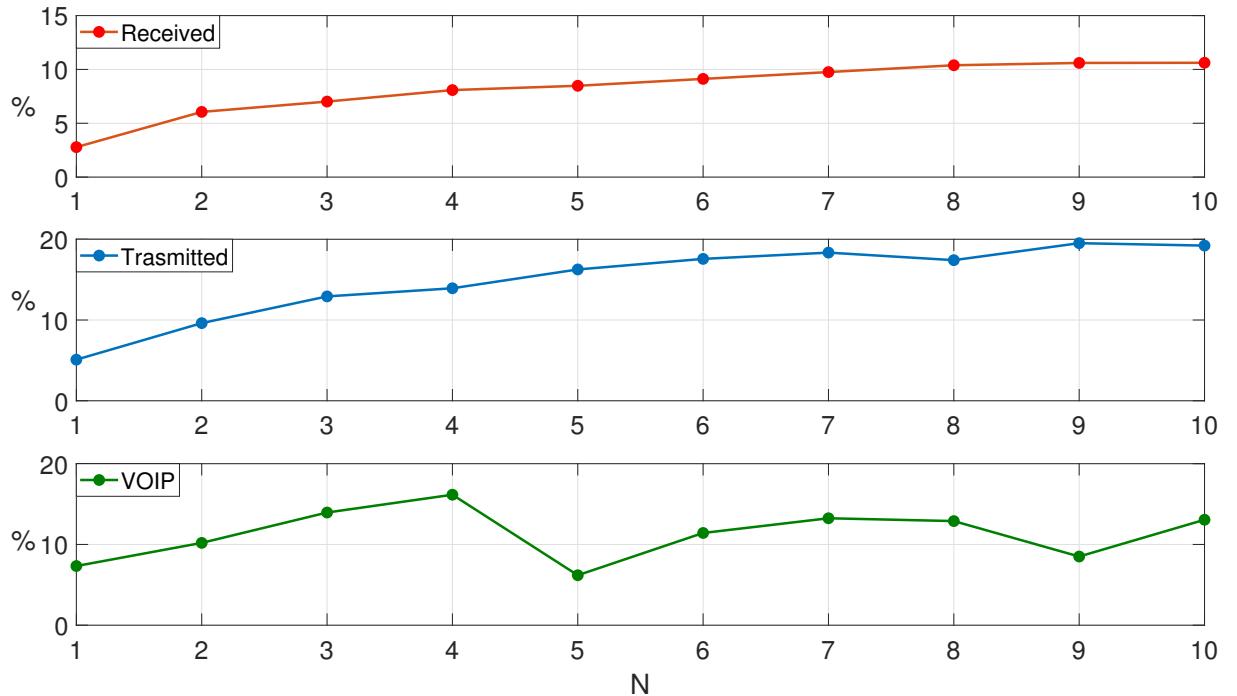


Figure 3.3: NRMSE of the packets predictive model over a time horizon of $N = 10$.

Since this device is part of a running commercial network, some constraints in data collection have forced to only measure the sum of all packets entering and leaving the device, and it has been possible to extract from such traffic only incoming VOIP packets: i.e., it has not been possible to extract packets differentiated for each DSCP. Moreover, it is not currently possible to apply any type of closed-loop control on the network device. For the above 2 reasons the control performance has not been validated.

About data analysis, 53 days of data measurements have been used for RF training and about 3 days for model validation. Figure 3.3 shows the prediction on three classes of packets: all packets received, all packets transmitted, VOIP packets received. The plots

show that our methodology provides a very accurate prediction even on a real internet service provider network.

After few months it has been possible to extend our trained model, using a total of 26138 samples, with the same method previously explained in section 2.3. This model has been validated on a test of 19144 samples, i.e. on more than two months of measurements. Even though the test pool is so large, thanks to the variety of train data it has been possible to obtain very low error values on packet prediction, as shown in Figure 3.4.

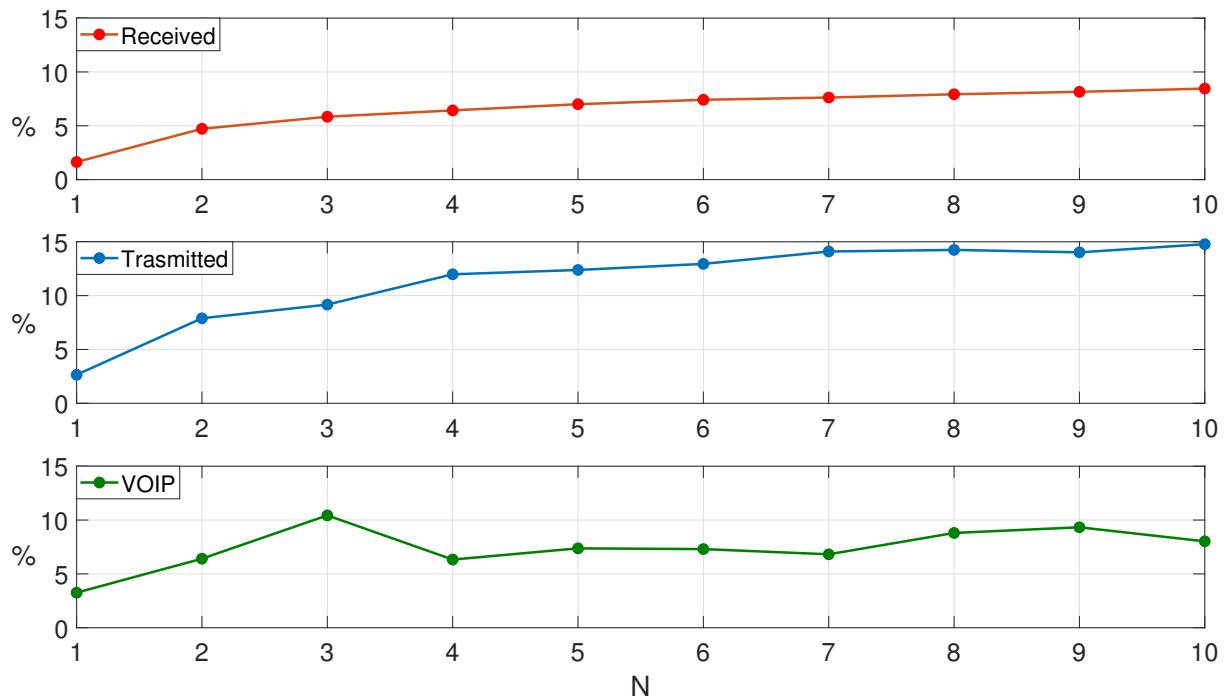


Figure 3.4: NRMSE of the packets predictive model over a time horizon of $N = 10$ after one data update.

At a later time Sonicate S.r.l. needed to move the monitored device to another location on their network by connecting different users with different contracts and therefore with different traffic profiles. This has been a great opportunity to validate the proposed method as it has been possible to test the network device computed model with a traffic pattern totally unrelated to the training data-set. This test could lead to two conclusions:

- 1) The prediction error could have been too large and a new long training period would have been needed;
- 2) The prediction error could be comparable to the prediction of the previous model and therefore there would be no need for further updates.

The final result is halfway between the two expectations. In fact, on a test of 846 samples

(about 3 days) with new setup data and the same model used for prediction in Figure 3.4, excellent results has been already obtained for Received and Transmitted packets (5% and 7.1% of NRMSE respectively) and sufficient results for the Voip (12% of NRMSE). Figure 3.5 shows the prediction error for $N = 1$ for different models. The case just specified is the one with the null abscissa, that is the model trained with the old setup data. For the abscissa equal to 1 we have an update of the old model with 1 day of new data (about 288 samples). For the abscissa equal to 2 there is an update of the old model with 2 days of new data and so on for the other values up to 9 updates. As can be seen from the error trend compared to the model updates, very low prediction errors has been obtained already after adding a day of data to the initial model. This very important result indicates how it is possible to adapt the predictive model for a network device. Furthermore, combining the results of Chapter 2 it is evident that this technique would allow an optimal control of the device, guaranteeing QoS without the need to increase the characteristics of the network hardware.

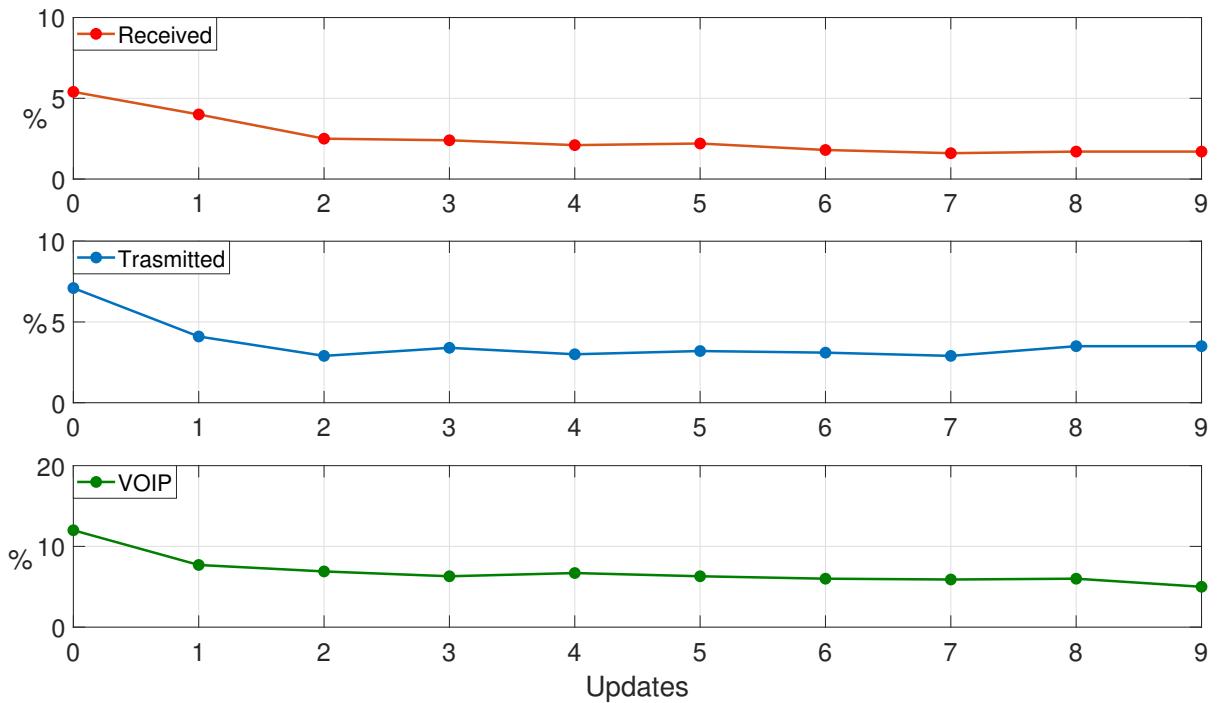


Figure 3.5: NRMSE of the packets predictive model at prediction horizon of $N=1$ with different updated models. Every update add 288 samples (about 1 day) to the previous model training data.

Conclusion

In this work a new methodology to derive accurate models for priority queueing in Software Defined Networks, in order to enable the application of advanced optimization techniques such as MPC, has been developed and validated over the Mininet network emulator framework and over a dedicated hardware network. The obtained results validate the prediction accuracy both of the incoming traffic and of the input/output behavior of a switch device in a SDN-based network. They also provide promising insights on the potential impact of predictive models combined with MPC in terms of packet losses reduction and bandwidth savings. Furthermore, it has been shown that this method obtains excellent prediction results on data coming from a real network of an Italian internet provider (Sonicatele s.r.l.), proving that the proposed control would be very useful to current communications networks to guarantee the required QoS. In future work it has been planned to validate the proposed controller over real network devices, instead of using Mininet.

References

- [1] M. J. Neely, Stochastic network optimization with application to communication and queueing systems, *Synthesis Lectures on Communication Networks* 3 (1) (2010) 1–211.
- [2] O. Lemeshko, T. Lebedenko, A. Al-Dulaimi, Improvement of method of balanced queue management on routers interfaces of telecommunication networks, in: 2019 3rd International Conference on Advanced Information and Communications Technologies (AICT), IEEE, 2019, pp. 170–175.
- [3] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, S. Uhlig, Software-defined networking: A comprehensive survey, *Proceedings of the IEEE* 103 (1) (2015) 14–76. doi:10.1109/JPROC.2014.2371999.
- [4] P. Amaral, J. Dinis, P. Pinto, L. Bernardo, J. Tavares, H. S. Mamede, Machine Learning in Software Defined Networks: Data collection and traffic classification, in: 2016 IEEE 24th International Conference on Network Protocols (ICNP), 2016, pp. 1–5. doi:10.1109/ICNP.2016.7785327.
- [5] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, OpenFlow: Enabling Innovation in Campus Networks, *SIGCOMM Comput. Commun. Rev.* 38 (2) (2008) 69–74. doi:10.1145/1355734.1355746.
- [6] OpenFlow Switch Specification, Version 1.3.0, The Open Networking Foundation, 2012.
URL <https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf>
- [7] J. Carner, A. Mestres, E. Alarcón, A. Cabellos, Machine learning-based network modeling: An artificial neural network model vs a theoretical inspired model, in: 2017 Ninth International Conference on Ubiquitous and Future Networks (ICUFN), 2017, pp. 522–524. doi:10.1109/ICUFN.2017.7993839.

- [8] S. Jain, M. Khandelwal, A. Katkar, J. Nygate, Applying big data technologies to manage QoS in an SDN, in: 2016 12th International Conference on Network and Service Management (CNSM), 2016, pp. 302–306. doi:10.1109/CNSM.2016.7818437.
- [9] R. Pasquini, R. Stadler, Learning end-to-end application QoS from openflow switch statistics, in: 2017 IEEE Conference on Network Softwarization (NetSoft), 2017, pp. 1–9. doi:10.1109/NETSOFT.2017.8004198.
- [10] E. Reticcioli, G. D. Girolamo, F. Smarra, A. Carmenini, A. D’Innocenzo, F. Graziosi, Learning SDN traffic flow accurate models to enable queue bandwidth dynamic optimization, European Conference on Networks and Communications (EuCNC), 2020 (2020).
- [11] J. Xie, F. R. Yu, T. Huang, R. Xie, J. Liu, C. Wang, Y. Liu, A Survey of Machine Learning Techniques Applied to Software Defined Networking (SDN): Research Issues and Challenges, *IEEE Communications Surveys Tutorials* 21 (1) (2019) 393–430. doi:10.1109/COMST.2018.2866942.
- [12] Open networking foundation.
URL <https://www.opennetworking.org/>
- [13] S. Sezer, S. Scott-Hayward, P. K. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, N. Rao, Are we ready for SDN? Implementation challenges for software-defined networks, *IEEE Communications Magazine* 51 (7) (2013) 36–43. doi:10.1109/MCOM.2013.6553676.
- [14] Open vSwitch, 2019.
URL <https://www.openvswitch.org/>
- [15] Indigo: Open source openflow switches.
URL <https://github.com/floodlight/indigo>
- [16] Pantou: Openflow 1.3 for open wrt.
URL <https://github.com/CPqD/ofsoftswitch13/wiki/OpenFlow1.3-for-OpenWRT>
- [17] J. W. Lockwood, N. McKeown, G. Watson, G. Gibb, P. Hartke, J. Naous, R. Raghu-raman, J. Luo, NetFPGA—An Open Platform for Gigabit-Rate Network Switching and Routing, in: 2007 IEEE International Conference on Microelectronic Systems Education (MSE’07), 2007, pp. 160–161.

- [18] M. B. Anwer, M. Motiwala, M. b. Tariq, N. Feamster, SwitchBlade: a platform for rapid deployment of network protocols on programmable hardware, in: Proceedings of the ACM SIGCOMM 2010 conference, SIGCOMM '10, Association for Computing Machinery, New York, NY, USA, 2010, pp. 183–194. doi: 10.1145/1851182.1851206.
 URL <https://doi.org/10.1145/1851182.1851206>
- [19] G. Lu, C. Guo, Y. Li, Z. Zhou, T. Yuan, H. Wu, Y. Xiong, R. Gao, Y. Zhang, Server-Switch: A Programmable and High Performance Platform for Data Center Networks (Mar. 2011).
- [20] RYU Controller, 2019.
 URL <https://osrg.github.io/ryu/>
- [21] J. Medved, R. Varga, A. Tkacik, K. Gray, OpenDaylight: Towards a Model-Driven SDN Controller architecture, in: Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014, 2014, pp. 1–6. doi: 10.1109/WoWMoM.2014.6918985.
- [22] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, S. Shenker, Nox: Towards an operating system for networks, SIGCOMM Comput. Commun. Rev., vol. 38, no. 3, pp. 105–110, (2008).
- [23] Pox.
 URL <https://github.com/noxrepo/pox>
- [24] Floodlight.
 URL <https://github.com/floodlight/floodlight>
- [25] D. Erickson, The beacon openflow controller, in: Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking, HotSDN '13, Association for Computing Machinery, New York, NY, USA, 2013, pp. 13–18. doi:10.1145/2491185.2491189.
 URL <https://doi.org/10.1145/2491185.2491189>
- [26] B. Pfaff, B. Davie, The open vswitch database management protocol (2013).
 URL <https://rfc-editor.org/rfc/rfc7047.txt>
- [27] H. Song, Protocol-oblivious forwarding: unleash the power of SDN through a future-proof forwarding plane, in: Proceedings of the second ACM SIGCOMM

workshop on Hot topics in software defined networking, HotSDN '13, Association for Computing Machinery, New York, NY, USA, 2013, pp. 127–132. doi: 10.1145/2491185.2491190.

URL <https://doi.org/10.1145/2491185.2491190>

- [28] G. Bianchi, M. Bonola, A. Capone, C. Cascone, OpenState: programming platform-independent stateful openflow applications inside the switch, ACM SIGCOMM Computer Communication Review 44 (2) (2014) 44–51. doi:10.1145/2602204.2602211.
URL <https://doi.org/10.1145/2602204.2602211>
- [29] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, S. Shenker, Onix: A distributed control platform for large-scale production networks, Proc. OSDI, vol. 10. (2010).
- [30] A. Tootoonchian, Y. Ganjali, Hyperflow: A distributed control plane for openflow, Proc. Enterprise Netw (2010).
- [31] H. Yin, H. Xie, T. Tsou, D. Lopez, P. Aranda, R. Sidi, SDNi: A Message Exchange Protocol for Software Defined Networks (SDNS) across Multiple Domains (draft-yin-sdn-sdni-00) (June 2012).
URL <http://tools.ietf.org/id/draft-yin-sdn-sdni-00.txt>
- [32] P. Lin, J. Bi, Y. Wang, East-West Bridge for SDN Network Peering, in: J. Su, B. Zhao, Z. Sun, X. Wang, F. Wang, K. Xu (Eds.), Frontiers in Internet Technologies, Communications in Computer and Information Science, Springer, Berlin, Heidelberg, 2013, pp. 170–181. doi:10.1007/978-3-642-53959-6_16.
- [33] F. Benamrane, M. Ben mamoun, R. Benaini, An East-West interface for distributed SDN control plane: Implementation and evaluation, Computers & Electrical Engineering 57 (2017) 162–175. doi:10.1016/j.compeleceng.2016.09.012.
URL <http://www.sciencedirect.com/science/article/pii/S0045790616302798>
- [34] A. Mendiola, J. Astorga, E. Jacob, M. Higuero, A Survey on the Contributions of Software-Defined Networking to Traffic Engineering, IEEE Communications Surveys Tutorials 19 (2) (2017) 918–953. doi:10.1109/COMST.2016.2633579.

- [35] I. Ahmad, S. Namal, M. Ylianttila, A. Gurtov, Security in Software Defined Networks: A Survey, *IEEE Communications Surveys Tutorials* 17 (4) (2015) 2317–2346. doi:10.1109/COMST.2015.2474118.
- [36] S. Scott-Hayward, S. Natarajan, S. Sezer, A Survey of Security in Software Defined Networks, *IEEE Communications Surveys Tutorials* 18 (1) (2016) 623–654. doi:10.1109/COMST.2015.2453114.
- [37] D. B. Rawat, S. R. Reddy, Software Defined Networking Architecture, Security and Energy Efficiency: A Survey, *IEEE Communications Surveys Tutorials* 19 (1) (2017) 325–346. doi:10.1109/COMST.2016.2618874.
- [38] S. T. Ali, V. Sivaraman, A. Radford, S. Jha, A Survey of Securing Networks Using Software Defined Networking, *IEEE Transactions on Reliability* 64 (3) (2015) 1086–1097. doi:10.1109/TR.2015.2421391.
- [39] Q. Yan, F. R. Yu, Q. Gong, J. Li, Software-Defined Networking (SDN) and Distributed Denial of Service (DDoS) Attacks in Cloud Computing Environments: A Survey, Some Research Issues, and Challenges, *IEEE Communications Surveys Tutorials* 18 (1) (2016) 602–622. doi:10.1109/COMST.2015.2487361.
- [40] T. Dargahi, A. Caponi, M. Ambrosin, G. Bianchi, M. Conti, A Survey on the Security of Stateful SDN Data Planes, *IEEE Communications Surveys Tutorials* 19 (3) (2017) 1701–1725. doi:10.1109/COMST.2017.2689819.
- [41] P. C. Fonseca, E. S. Mota, A Survey on Fault Management in Software-Defined Networks, *IEEE Communications Surveys Tutorials* 19 (4) (2017) 2284–2321. doi:10.1109/COMST.2017.2719862.
- [42] J. W. Guck, A. Van Bemten, M. Reisslein, W. Kellerer, Unicast QoS Routing Algorithms for SDN: A Comprehensive Survey and Performance Evaluation, *IEEE Communications Surveys Tutorials* 20 (1) (2018) 388–415. doi:10.1109/COMST.2017.2749760.
- [43] R. Alvizu, G. Maier, N. Kukreja, A. Pattavina, R. Morro, A. Capello, C. Cavazzoni, Comprehensive Survey on T-SDN: Software-Defined Networking for Transport Networks, *IEEE Communications Surveys Tutorials* 19 (4) (2017) 2232–2283. doi:10.1109/COMST.2017.2715220.

- [44] A. S. Thyagaturu, A. Mercian, M. P. McGarry, M. Reisslein, W. Kellerer, Software Defined Optical Networks (SDONs): A Comprehensive Survey, *IEEE Communications Surveys Tutorials* 18 (4) (2016) 2738–2786. doi:10.1109/COMST.2016.2586999.
- [45] I. T. Haque, N. Abu-Ghazaleh, Wireless Software Defined Networking: A Survey and Taxonomy, *IEEE Communications Surveys Tutorials* 18 (4) (2016) 2713–2737. doi:10.1109/COMST.2016.2571118.
- [46] T. Chen, M. Matinmikko, X. Chen, X. Zhou, P. Ahokangas, Software defined mobile networks: concept, survey, and research directions, *IEEE Communications Magazine* 53 (11) (2015) 126–133. doi:10.1109/MCOM.2015.7321981.
- [47] S. Bera, S. Misra, A. V. Vasilakos, Software-Defined Networking for Internet of Things: A Survey, *IEEE Internet of Things Journal* 4 (6) (2017) 1994–2008. doi:10.1109/JIOT.2017.2746186.
- [48] A. C. Baktir, A. Ozgovde, C. Ersoy, How Can Edge Computing Benefit From Software-Defined Networking: A Survey, Use Cases, and Future Directions, *IEEE Communications Surveys Tutorials* 19 (4) (2017) 2359–2391. doi:10.1109/COMST.2017.2717482.
- [49] O. Michel, E. Keller, SDN in wide-area networks: A survey, in: 2017 Fourth International Conference on Software Defined Systems (SDS), 2017, pp. 37–42. doi:10.1109/SDS.2017.7939138.
- [50] R. Jain, S. Paul, Network virtualization and software defined networking for cloud computing: a survey, *IEEE Communications Magazine* 51 (11) (2013) 24–31. doi:10.1109/MCOM.2013.6658648.
- [51] Y. Li, M. Chen, Software-Defined Network Function Virtualization: A Survey, *IEEE Access* 3 (2015) 2542–2553. doi:10.1109/ACCESS.2015.2499271.
- [52] C. Liang, F. R. Yu, Wireless Network Virtualization: A Survey, Some Research Issues and Challenges, *IEEE Communications Surveys Tutorials* 17 (1) (2015) 358–380. doi:10.1109/COMST.2014.2352118.
- [53] B. A. A. Nunes, M. Mendonca, X. Nguyen, K. Obraczka, T. Turletti, A survey of software-defined networking: Past, present, and future of programmable networks, *IEEE Communications Surveys Tutorials* 16 (3) (2014) 1617–1634.

- [54] Y. Jarraya, T. Madi, M. Debbabi, A Survey and a Layered Taxonomy of Software-Defined Networking, *IEEE Communications Surveys Tutorials* 16 (4) (2014) 1955–1980. doi:10.1109/COMST.2014.2320094.
- [55] W. Xia, Y. Wen, C. H. Foh, D. Niyato, H. Xie, A Survey on Software-Defined Networking, *IEEE Communications Surveys Tutorials* 17 (1) (2015) 27–51. doi:10.1109/COMST.2014.2330903.
- [56] F. Hu, Q. Hao, K. Bao, A Survey on Software-Defined Network and OpenFlow: From Concept to Implementation, *IEEE Communications Surveys Tutorials* 16 (4) (2014) 2181–2206. doi:10.1109/COMST.2014.2326417.
- [57] J. Xie, D. Guo, Z. Hu, T. Qu, P. Lv, Control plane of software defined networks: A survey, *Computer Communications* 67 (2015) 1–10. doi:10.1016/j.comcom.2015.06.004.
URL <http://www.sciencedirect.com/science/article/pii/S0140366415002200>
- [58] C. Trois, M. D. Del Fabro, L. C. E. de Bona, M. Martinello, A Survey on SDN Programming Languages: Toward a Taxonomy, *IEEE Communications Surveys Tutorials* 18 (4) (2016) 2687–2712. doi:10.1109/COMST.2016.2553778.
- [59] T. Huang, F. R. Yu, C. Zhang, J. Liu, J. Zhang, Y. Liu, A Survey on Large-Scale Software Defined Networking (SDN) Testbeds: Approaches and Challenges, *IEEE Communications Surveys Tutorials* 19 (2) (2017) 891–917. doi:10.1109/COMST.2016.2630047.
- [60] A. Blenk, A. Basta, M. Reisslein, W. Kellerer, Survey on Network Virtualization Hypervisors for Software Defined Networking, *IEEE Communications Surveys Tutorials* 18 (1) (2016) 655–685. doi:10.1109/COMST.2015.2489183.
- [61] M. Mohammed, M. B. Khan, E. B. M. Bashier, *Machine Learning: Algorithms and Applications*, CRC Press, 2016, google-Books-ID: X8LBDAAAQBAJ.
- [62] S. Marsland, *Machine Learning: An Algorithmic Perspective*, Second Edition, CRC Press, 2015, google-Books-ID: y_oYCwAAQBAJ.
- [63] E. Alpaydin, *Introduction to Machine Learning*, MIT Press, 2020, google-Books-ID: tZnSDwAAQBAJ.

- [64] I. Z. SB Kotsiantis, Supervised machine learning: A review of classification techniques, Emerging Artificial Intelligence Applications in Computer Engineering (2007).
- [65] J. F. Trevor Hastie, Robert Tibshirani, The Elements of Statistical Learning : Data Mining, Inference, and Prediction, 2009.
- [66] T. Cover, P. Hart, Nearest neighbor pattern classification, IEEE Transactions on Information Theory 13 (1) (1967) 21–27. doi:10.1109/TIT.1967.1053964.
- [67] J. Han, J. Pei, M. Kamber, Data Mining: Concepts and Techniques, Elsevier, 2011, google-Books-ID: pQws07tdpjoC.
- [68] J. R. Quinlan, Induction of decision trees, Machine Learning 1 (1) (1986) 81–106. doi:10.1007/BF00116251.
URL <https://doi.org/10.1007/BF00116251>
- [69] S. Karatsiolis, C. N. Schizas, Region based Support Vector Machine algorithm for medical diagnosis on Pima Indian Diabetes dataset, in: 2012 IEEE 12th International Conference on Bioinformatics Bioengineering (BIBE), 2012, pp. 139–144. doi:10.1109/BIBE.2012.6399663.
- [70] W. R. Burrows, M. Benjamin, S. Beauchamp, E. R. Lord, D. McCollor, B. Thomson, CART Decision-Tree Statistical Analysis and Prediction of Summer Season Maximum Surface Ozone for the Vancouver, Montreal, and Atlantic Regions of Canada, Journal of Applied Meteorology 34 (8) (1995) 1848–1862. doi:10.1175/1520-0450(1995)034<1848:CDTSAA>2.0.CO;2.
URL <https://journals.ametsoc.org/jamc/article/34/8/1848/15131/CART-Decision-Tree-Statistical-Analysis-and>
- [71] L. Breiman, Random forests (1999).
- [72] S. Haykin, Neural Networks: A Comprehensive Foundation.
- [73] K. Lee, D. Booth, P. Alam, A comparison of supervised and unsupervised neural networks in predicting bankruptcy of Korean firms, Expert Systems with Applications 29 (1) (2005) 1–16. doi:10.1016/j.eswa.2005.01.004.
URL <http://www.sciencedirect.com/science/article/pii/S0957417405000023>

- [74] S. Timotheou, The random neural network: a survey, *The computer journal* 53 (3) (2010) 251–267.
- [75] G. H. Yann LeCun, Yoshua Bengio, Deep learning, *Nature* (2015).
- [76] J. Schmidhuber, Deep learning in neural networks: An overview, *Neural Networks* 61 (2015) 85–117. doi:10.1016/j.neunet.2014.09.003.
 URL <http://www.sciencedirect.com/science/article/pii/S0893608014002135>
- [77] A. D. Gaurav Pandey, Learning by stretching deep networks, *Proceedings of the 31st International Conference on MachineLearning*, Beijing, China (2014).
- [78] A. Krizhevsky, I. Sutskever, G. E. Hinton, ImageNet Classification with Deep Convolutional Neural Networks, in: F. Pereira, C. J. C. Burges, L. Bottou, K. Q. Weinberger (Eds.), *Advances in Neural Information Processing Systems 25*, Curran Associates, Inc., 2012, pp. 1097–1105.
- [79] C. Li, Y. Wu, X. Yuan, Z. Sun, W. Wang, X. Li, L. Gong, Detection and defense of DDoS attack–based on deep learning in OpenFlow-based SDN, *International Journal of Communication Systems* 31 (5) (2018) e3497. doi:10.1002/dac.3497.
 URL <https://onlinelibrary.wiley.com/doi/abs/10.1002/dac.3497>
- [80] X. Li, X. Wu, Constructing long short-term memory based deep recurrent neural networks for large vocabulary speech recognition, in: 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2015, pp. 4520–4524, iSSN: 2379-190X. doi:10.1109/ICASSP.2015.7178826.
- [81] V. Vapnik, An overview of statistical learning theory, *IEEE Transactions on Neural Networks* 10 (5) (1999) 988–999. doi:10.1109/72.788640.
- [82] G. E. P. Box, G. C. Tiao, *Bayesian Inference in Statistical Analysis*, John Wiley & Sons, 2011, google-Books-ID: T8Askeyk1k4C.
- [83] J. Bakker, Intelligent Traffic Classification for Detecting DDoS Attacks using SDN/OpenFlow (2017).
 URL <http://researcharchive.vuw.ac.nz/handle/10063/6645>

- [84] L. Rabiner, A tutorial on hidden Markov models and selected applications in speech recognition, *Proceedings of the IEEE* 77 (2) (1989) 257–286. doi:10.1109/5.18626.
- [85] P. Holgado, V. A. Villagrá, L. Vázquez, Real-Time Multistep Attack Prediction Based on Hidden Markov Models, *IEEE Transactions on Dependable and Secure Computing* 17 (1) (2020) 134–147. doi:10.1109/TDSC.2017.2751478.
- [86] T. Kohonen, *Self-Organizing Maps*, Springer Science & Business Media, 2012.
- [87] H. Wu, S. Prasad, Semi-Supervised Deep Learning Using Pseudo Labels for Hyperspectral Image Classification, *IEEE Transactions on Image Processing* 27 (3) (2018) 1259–1270. doi:10.1109/TIP.2017.2772836.
- [88] R. S. Sutton, A. G. Barto, *Reinforcement Learning*, second edition: An Introduction, MIT Press, 2018, google-Books-ID: uWV0DwAAQBAJ.
- [89] L. P. Kaelbling, M. L. Littman, A. W. Moore, Reinforcement Learning: A Survey, *Journal of Artificial Intelligence Research* 4 (1996) 237–285. doi:10.1613/jair.301.
URL <https://www.jair.org/index.php/jair/article/view/10166>
- [90] L. Tan, *Resource Allocation and Performance Optimization in Communication Networks and the Internet*, CRC press, 2017.
- [91] A. M. Abdelmoniem, B. Bensaou, Hysteresis-based active queue management for tcp traffic in data centers, in: *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, IEEE, 2019, pp. 1621–1629.
- [92] L. Boero, M. Cello, C. Garibotto, M. Marchese, M. Mongelli, BeaQoS: Load balancing and deadline management of queues in an OpenFlow SDN switch, *Computer Networks* 106 (2016) 161–170.
- [93] K. S. Umadevi, M. S. S. Pranay, K. Rachana, Multilevel queue scheduling in software defined networks, in: *2017 Innovations in Power and Advanced Computing Technologies (i-PACT)*, 2017, pp. 1–4. doi:10.1109/IPACT.2017.8245144.

- [94] C. Olariu, M. Zuber, C. Thorpe, Delay-based priority queueing for voip over software defined networks, in: 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), 2017, pp. 652–655. doi:[10.23919/INM.2017.7987352](https://doi.org/10.23919/INM.2017.7987352).
- [95] M. Haiyan, Y. Jinyao, P. Georgopoulos, B. Plattner, Towards sdn based queuing delay estimation, *China Communications* 13 (3) (2016) 27–36. doi:[10.1109/CC.2016.7445500](https://doi.org/10.1109/CC.2016.7445500).
- [96] Y. Chen, L. Wang, F. Lin, B. P. Lin, Deterministic quality of service guarantee for dynamic service chaining in software defined networking, *IEEE Transactions on Network and Service Management* 14 (4) (2017) 991–1002. doi:[10.1109/TNSM.2017.2758328](https://doi.org/10.1109/TNSM.2017.2758328).
- [97] A. Al-Najjar, S. Layeghy, M. Portmann, J. Indulska, Enhancing quality of experience of voip traffic in sdn based end-hosts, in: 2018 28th International Telecommunication Networks and Applications Conference (ITNAC), 2018, pp. 1–8. doi:[10.1109/ATNAC.2018.8615286](https://doi.org/10.1109/ATNAC.2018.8615286).
- [98] Q.-L. Li, J.-Y. Ma, R.-N. Fan, L. Xia, An Overview for Markov Decision Processes in Queues and Networks, 2019, pp. 44–71. doi:[10.1007/978-981-15-0864-6_3](https://doi.org/10.1007/978-981-15-0864-6_3).
- [99] A. Asanjarani, Y. Nazarathy, P. K. Pollett, Parameter and state estimation in queues and related stochastic models: A bibliography (2017). arXiv:[1701.08338](https://arxiv.org/abs/1701.08338).
- [100] K. Sood, S. Yu, Y. Xiang, Performance analysis of software-defined network switch using \$m/geo/1\$ model, *IEEE Communications Letters* 20 (2016) 2522–2525.
- [101] D. Singh, B. Ng, Y.-C. Lai, Y.-D. Lin, W. K. Seah, Modelling software-defined networking: Software and hardware switches, *Journal of Network and Computer Applications* 122 (2018) 24 – 36.
- [102] R. Schoffauer, G. Wunder, Model-predictive control for discrete-time queueing networks with varying topology, ArXiv abs/2004.01985 (2020).
- [103] P. Wang, H. Chen, X. Yang, Y. Ma, Design and analysis of a model predictive controller for active queue management, *ISA Transactions* 51 (1) (2012) 120 – 131.

- [104] Ki Baek Kim, Design of feedback controls supporting tcp based on the state-space approach, *IEEE Transactions on Automatic Control* 51 (7) (2006) 1086–1099. doi: 10.1109/TAC.2006.878749.
- [105] J. Kim, G. Hwang, Prediction based efficient online bandwidth allocation method, *IEEE Communications Letters* 23 (12) (2019) 2330–2334. doi:10.1109/LCOMM.2019.2947895.
- [106] W. Aljoby, X. Wang, T. Z. J. Fu, R. T. B. Ma, On sdn-enabled online and dynamic bandwidth allocation for stream analytics, *IEEE Journal on Selected Areas in Communications* 37 (8) (2019) 1688–1702. doi:10.1109/JSAC.2019.2927062.
- [107] T. Lebedenko, O. Yeremenko, S. Harkusha, A. S. Ali, Dynamic model of queue management based on resource allocation in telecommunication networks, in: 2018 14th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET), IEEE, 2018, pp. 1035–1038.
- [108] L. Le, J. Aikat, K. Jeffay, F. D. Smith, The effects of active queue management and explicit congestion notification on web performance, *IEEE/ACM Transactions on Networking* 15 (6) (2007) 1217–1230. doi:10.1109/TNET.2007.910583.
- [109] and Sourav Ghosh, R. Rajkumar, J. Hansen, J. Lehoczky, Scalable QoS-based resource allocation in hierarchical networked environment, in: Proc. 11th IEEE Real Time and Embedded Technology and Applications Symp, 2005, pp. 256–267. doi:10.1109/RTAS.2005.47.
- [110] M. Wang, Y. Cui, X. Wang, S. Xiao, J. Jiang, Machine learning for networking: Workflow, advances and opportunities, *IEEE Network* 32 (2) (2018) 92–99. doi: 10.1109/MNET.2017.1700200.
- [111] M. Usama, J. Qadir, A. Raza, H. Arif, K.-L. A. Yau, Y. Elkhatib, A. Hussain, A. Al-Fuqaha, Unsupervised Machine Learning for Networking: Techniques, Applications and Research Challenges, arXiv:1709.06599 [cs]ArXiv: 1709.06599 (Sep. 2017).
- [112] G. Xu, Y. Mu, J. Liu, Inclusion of artificial intelligence in communication networks and services (Jan. 2018).
- [113] D. D. Clark, C. Partridge, J. C. Ramming, J. T. Wroclawski, A knowledge plane for the internet, in: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications, ACM, 2003, pp. 3–10.

- [114] A. Mestres, A. Rodriguez-Natal, J. Carner, P. Barlet-Ros, E. Alarcón, M. Solé, V. Muntés-Mulero, D. Meyer, S. Barkai, M. J. Hibbett, et al., Knowledge-defined networking, *ACM SIGCOMM Computer Communication Review* 47 (3) (2017) 2–10.
- [115] M. Jarschel, T. Zinner, T. Hossfeld, P. Tran-Gia, W. Kellerer, Interfaces, attributes, and use cases: A compass for sdn, *IEEE Communications Magazine* 52 (6) (2014) 210–217. doi:10.1109/MCOM.2014.6829966.
- [116] P. Ameigeiras, J. J. Ramos-munoz, L. Schumacher, J. Prados-Garzon, J. Navarro-Ortiz, J. M. Lopez-soler, Link-level access cloud architecture design based on sdn for 5g networks, *IEEE Network* 29 (2) (2015) 24–31. doi:10.1109/MNET.2015.7064899.
- [117] O. Yeremenko, O. Simonenko, Mathematical model of queue management with flows aggregation and bandwidth allocation, *Advances in Computer Science for Engineering and Education* 754 (2018) 165.
- [118] J. John, R. Balan, Priority queuing technique promoting deadline sensitive data transfers in router based heterogeneous networks, *International Journal of Applied Engineering Research* 12 (15) (2017) 4899–4903.
- [119] J. Yang, S. Ulukus, Trading Rate for Balanced Queue Lengths for Network Delay Minimization, *IEEE Journal on Selected Areas in Communications* 29 (5) (2011) 988–996. doi:10.1109/JSAC.2011.110509.
- [120] M. Bahnasy, H. Elbiaze, B. Boughzala, Zero-queue ethernet congestion control protocol based on available bandwidth estimation, *Journal of Network and Computer Applications* 105 (2018) 1–20. doi:10.1016/j.jnca.2017.12.016.
URL <http://www.sciencedirect.com/science/article/pii/S108480451730423X>
- [121] J. Zhang, F. Ren, X. Yue, R. Shu, C. Lin, Sharing Bandwidth by Allocating Switch Buffer in Data Center Networks, *IEEE Journal on Selected Areas in Communications* 32 (1) (2014) 39–51. doi:10.1109/JSAC.2014.140105.
- [122] M. Cello, M. Marchese, M. Mongelli, On the QoS Estimation in an OpenFlow Network: The Packet Loss Case, *IEEE Communications Letters* 20 (3) (2016) 554–557.

- [123] J. Lee, S. Bohacek, J. Hespanha, K. Obraczka, Modeling communication networks with hybrid systems, *IEEE/ACM Transactions on Networking* 15 (3) (2007) 630–643.
- [124] M. D. Di Benedetto, A. Di Loreto, A. D’Innocenzo, T. Ionta, Modeling of traffic congestion and re-routing in a service provider network, in: Proc. IEEE Int. Conf. Communications Workshops (ICC), 2014, pp. 557–562. doi:10.1109/ICCW.2014.6881257.
- [125] P. Mulinka, P. Casas, Stream-based machine learning for network security and anomaly detection, in: Proceedings of the 2018 Workshop on Big Data Analytics and Machine Learning for Data Communication Networks, ACM, 2018, pp. 1–7.
- [126] A. Patcha, J.-M. Park, An overview of anomaly detection techniques: Existing solutions and latest technological trends, *Computer networks* 51 (12) (2007) 3448–3470.
- [127] T. T. Nguyen, G. Armitage, A survey of techniques for internet traffic classification using machine learning, *IEEE communications surveys & tutorials* 10 (4) (2008) 56–76.
- [128] M. Bkassiny, Y. Li, S. K. Jayaweera, A survey on machine-learning techniques in cognitive radios, *IEEE Communications Surveys Tutorials* 15 (3) (2013) 1136–1159.
- [129] M. A. Alsheikh, S. Lin, D. Niyato, H. Tan, Machine learning in wireless sensor networks: Algorithms, strategies, and applications, *IEEE Communications Surveys Tutorials* 16 (4) (2014) 1996–2018.
- [130] X. Wang, X. Li, V. C. M. Leung, Artificial intelligence-based techniques for emerging heterogeneous network: State of the arts, opportunities, and challenges, *IEEE Access* 3 (2015) 1379–1391.
- [131] A. L. Buczak, E. Guven, A survey of data mining and machine learning methods for cyber security intrusion detection, *IEEE Communications Surveys Tutorials* 18 (2) (2016) 1153–1176.
- [132] P. V. Klaine, M. A. Imran, O. Onireti, R. D. Souza, A survey of machine learning techniques applied to self-organizing cellular networks, *IEEE Communications Surveys Tutorials* 19 (4) (2017) 2392–2431.

- [133] Z. M. Fadlullah, F. Tang, B. Mao, N. Kato, O. Akashi, T. Inoue, K. Mizutani, State-of-the-art deep learning: Evolving machine intelligence toward tomorrow's intelligent network traffic control systems, *IEEE Communications Surveys Tutorials* 19 (4) (2017) 2432–2455.
- [134] E. Hodo, X. Bellekens, A. Hamilton, C. Tachtatzis, R. Atkinson, Shallow and deep networks intrusion detection system: A taxonomy and survey [arXiv:1701.02145v1](https://arxiv.org/abs/1701.02145v1).
- [135] X. Zhou, M. Sun, G. Y. Li, B.-H. Juang, Intelligent wireless communications enabled by cognitive radio and machine learning [arXiv:1710.11240v4](https://arxiv.org/abs/1710.11240v4).
- [136] M. Chen, U. Challita, W. Saad, C. Yin, M. Debbah, Artificial neural networks-based machine learning for wireless networks: A tutorial [arXiv:1710.02913v2](https://arxiv.org/abs/1710.02913v2).
- [137] Mininet, 2019.
 URL <http://mininet.org/>
- [138] S. Avallone, S. Guadagno, D. Emma, A. Pescapè, G. Ventre, D-itg distributed internet traffic generator, in: First International Conference on the Quantitative Evaluation of Systems, 2004. QEST 2004. Proceedings., IEEE, 2004, pp. 316–317.
- [139] A. Botta, A. Dainotti, A. Pescapé, A tool for the generation of realistic network workload for emerging networking scenarios, *Computer Networks* 56 (15) (2012) 3531–3547.
- [140] A. Botta, W. de Donato, A. Dainotti, S. Avallone, A. Pescape, D-itg 2.8. 1 manual, Computer for Interaction and Communications (COMICS) Group, Department of Electrical Engineering and Information Technologies, University of Naples Federico II, Naples, Italy (www.grid.unina.it/software/ITG/manual) (2013).
- [141] F. Baker, D. Black, S. Blake, K. Nichols, Definition of the differentiated services field (ds field) in the ipv4 and ipv6 headers, Tech. rep., RFC 2474, Dec (1998).
- [142] J. Babiarz, K. Chan, F. Baker, Configuration guidelines for diffserv service classes, RFC 4594 (August 2006).
- [143] A. M. Langellotti, S. Mastropietro, F. T. Moretti, A. Soldati, Notiziario Tecnico Telecom Italia, Tech. rep., Telecom Italia (2004).

- [144] ryu.app.ofctl rest, 2019.
 URL https://ryu.readthedocs.io/en/latest/app/ofctl_rest.html
- [145] QoS Ryubook 1.0 documentation, 2019.
 URL https://osrg.github.io/ryu-book/en/html/rest_qos.html
- [146] F. Smarra, A. Jain, R. Mangharam, A. D’Innocenzo, Data-driven switched affine modeling for model predictive control, in: IFAC Conference on Analysis and Design of Hybrid Systems (ADHS’18), IFAC, 2018, pp. 199–204.
- [147] F. Smarra, G. D. Di Girolamo, V. De Iuliis, A. Jain, R. Mangharam, A. D’Innocenzo, Data-driven switching modeling for mpc using regression trees and random forests, Nonlinear Analysis: Hybrid Systems 36C (2020).
- [148] F. Borrelli, et al., Predictive control for linear and hybrid systems, Cambridge University Press, 2017.
- [149] L. Breiman, Classification and regression trees, Routledge, 2017.
- [150] L. Breiman, Random forests, Machine learning 45 (1) (2001) 5–32.
- [151] UDOO x86, 2019.
 URL <https://www.udoo.org/>
- [152] O. I. Abiodun, A. Jantan, A. E. Omolara, K. V. Dada, N. A. Mohamed, H. Arshad, State-of-the-art in artificial neural network applications: A survey, Heliyon 4 (11) (2018) e00938.
- [153] M. Abadi, et al., TensorFlow: Large-scale machine learning on heterogeneous systems (2015).
 URL <https://www.tensorflow.org/>
- [154] F. Chollet, et al., Keras, <https://keras.io> (2015).
- [155] A. I. Techniques, opennn, www.opennn.net (2019).
- [156] M. F. Møller, A scaled conjugate gradient algorithm for fast supervised learning, Aarhus University, Computer Science Department, 1990.

[157] Teach, Learn, and Make with Raspberry Pi – Raspberry Pi.

URL <https://www.raspberrypi.org/>

[158] Cacti.

URL <https://www.cacti.net/>

Publications

- [1] Enrico Reticcioli, Tommaso Campi, Valerio De Santis, An Automated Scanning System for the Acquisition of Non-Uniform Time-Varying Magnetic Fields, IEEE Transactions on Instrumentation and Measurement, 2019
- [2] Achin Jain, Francesco Smarra, Enrico Reticcioli, Alessandro D'Innocenzo, Manfred Morari, NeurOpt: Neural network based optimization for building energy management and climate control, Learning for Dynamics and Control (L4DC) conference, 2020
- [3] Enrico Reticcioli, Giovanni Domenico Di Girolamo, Francsco Smarra, Alessio Carmenini, Alessandro D'Innocenzo and Fabio Graziosi, Learning SDN traffic flow accurate models to enable queue bandwidth dynamic optimization, European Conference on Networks and Communications (EuCNC 2020) conference, 2020
- [4] Enrico Reticcioli, Giovanni Domenico Di Girolamo, Francsco Smarra, Fabio Graziosi and Alessandro D'Innocenzo, Model Identification and Control of Priority Queueing in Software Defined Networks, Submitted, 2021

Appendix A

Python Codes for Mininet Environment

A.1 main_controller_TOS.py

```
1 from ryu.base import app_manager
2 from ryu.controller import ofp_event
3 from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER
4 from ryu.controller.handler import set_ev_cls
5 from ryu.ofproto import ofproto_v1_3
6 from qos_simple_switch_13 import *
7 from datapath_monitor_TOS import *
8 from ryu.lib.packet import arp
9 from ryu.lib.packet.arp import ARP_REQUEST, ARP_REPLY
10 from ryu.lib.packet import ipv4
11
12
13 class MainControllerMonitor(app_manager.RyuApp):
14     OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]
15
16     def __init__(self, *args, **kwargs):
17         super(MainControllerMonitor, self).__init__(*args, **kwargs)
18         self.device_behaviour = SimpleSwitch13(*args, **kwargs)
19         self.datapath_id_list = []
20         self.mac_to_port = {}
21         STEP = 300
22         args = {
23             "step":STEP,
24             "logger":self.logger,
25             "dp":self.datapath_id_list
26         }
27         self.monitor = DatapathMonitor(args)
28         self.monitor.start()
29
30     @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
31     def switch_feature_handler(self, ev):
32         self.device_behaviour.switch_features_handler(ev)
33         datapath = ev.msg.datapath
34         if datapath not in self.datapath_id_list:
35             self.datapath_id_list.append(datapath)
```

```

36         self.monitor.update(self.datapath_id_list)
37
38     @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
39     def _packet_in_handler(self, ev):
40         self.device_beaviour._packet_in_handler(ev)
41
42     @set_ev_cls(ofp_event.EventOFPDescStatsReply, MAIN_DISPATCHER)
43     def desc_stat_reply_handler(self, ev):
44         self.monitor.desc_reply(ev)

```

A.2 datapath_monitor_TOS.py

```

1 from ryu.controller import ofp_event
2 from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER
3 from ryu.ofproto import ofproto_v1_3
4 from ryu.lib.packet import packet
5 from ryu.lib.packet import ethernet
6 from ryu.lib.packet import ether_types
7 from operator import attrgetter
8 import threading
9 import time
10 import datetime
11
12 import subprocess
13 import json
14 import sys
15
16 from Controller_commands import *
17 import numpy as np
18
19 from subprocess import call
20 import random
21 import os
22
23 now=datetime.datetime.now()
24 year=str(now.year)
25 month=str(now.month)
26 day=str(now.day)
27 date=year+"-"+month+"-"+day+" "
28 computername="jedi"
29 dpb_folder="/home/jedi/Dropbox/DataNetworkShared/"
30 file = open("/home/"+computername+"/Scrivania/RyuDatapathMonitor-master/
31             DataLog/"+date+"flowstat.txt","a")
32 stringa="time"\t+datapath"\t"+in-port"\t"+eth-dst"\t"+out-
33             port"\t"+packets"\t"+bytes"\t"+ip-dscp"\t"+SET_QUEUE\n"
34 file.write(stringa)
35 file.close()
36
37 file = open("/home/"+computername+"/Scrivania/RyuDatapathMonitor-master/
38             DataLog/"+date+"queuestat.txt","a")

```

```

36 stringa="time"+'\t'+ "datapath"+'\t'+ "port_no"+'\t'+ "queue_id"+'\t'+ "
37     tx_bytes"+'\t'+ "tx_packets"+'\t'+ "tx_errors"+'\t'+ "duration_sec"+'\t'+ "
38     "+ "duration_nsec"\n"
39 file .write(stringa)
40 file .close()
41
42 file = open("/home/" + computername + "/Scrivania/RyuDatapathMonitor-master/
43             DataLog/" + date + "portstat.txt", "a")
44 stringa="time"+'\t'+ "datapath"+'\t'+ "port"+'\t'+ "rx_pkts"+'\t'+ "rx_bytes"
45     "+'\t'+ "rx_error"+'\t'+ "tx_pkts"+'\t'+ "tx_bytes"+'\t'+ "tx_error"+'\t'
46     "+ 'rx_dropped"+'\t'+ "tx_dropped"+'\t'+ "rx_crc_err"+'\t'+ "collisions"
47     "+ '\n"
48 file .write(stringa)
49 file .close()
50
51 file = open("/home/" + computername + "/Scrivania/RyuDatapathMonitor-master/
52             DataLog/" + date + "queueconfig.txt", "a")
53 stringa="time"+'\t'+ "datapath"+'\t'+ "queue_id"+'\t'+ "type_of_rule"+'\t'+ "
54     "+ "rate\n"
55 file .write(stringa)
56 file .close()
57
58 f = open(dpb_folder+"FlagData.txt","w")
59 f.write("0")
60 f.close()
61
62 f = open(dpb_folder+"FlagU.txt","w")
63 f.write("0")
64 f.close()
65
66 def get_switchis():
67     try:
68         output = subprocess.check_output(
69             "curl -X GET http://localhost:8080/stats/switches",
70             stderr=subprocess.STDOUT,
71             shell=True)
72         output=output[output.find("["):]
73         end_response=output.find("]")
74         list1=list(output)
75         list1[end_response]=','
76         output=''.join(list1)+"]"
77     except:
78         output="No NET"
79     return output
80
81 def save_flow_stat(datapath):
82     datapath_in=datapath
83     mom_datapath= ['0' for i in range(16-len(datapath))]
84     mom_datapath=''.join(mom_datapath)
85     datapath=mom_datapath+datapath
86     try:
87         output = subprocess.check_output(
88             "curl -X GET http://localhost:8080/stats/flow/" + datapath ,
89             shell=True)

```

```

82     i=0
83     output = output[output.find("{"")+1:]
84     end_response = output.find("}")]+2
85     list1=list(output)
86     list1[end_response-2]='}'
87     list1[end_response-1]=','
88     output=''.join(list1)
89     while i<end_response:
90         output_i=output[i:]
91         i=i+output[i:].find("},")+2
92         otp = output_i[output_i.find("{"):]
93         otp = otp[0:otp.find("},")]+1
94         otp =eval(otp)
95         data = otp
96         json_str = json.dumps(data)
97         jsonList = json.loads(json_str)
98         if jsonList['priority']!=0 and jsonList['match'].get(
99             'in_port'):
100            porta=str(jsonList['actions'])
101            prc = porta.find('T:')
102            #Check if there is "OUTPUT:" in the string
103            if prc >= 0:
104                porta=porta[prc+2:]
105                porta = int(porta[0:porta.find(']')-1])
106                file = open("/home/"+computername+"/Scrivania/
RyuDatapathMonitor-master/DataLog/"+date+"flowstat.txt","a")
107                now=datetime.datetime.now()
108                stringa=str(now)+"\t"+datapath_in+"\t"+str(jsonList[
109                    'match'].get('in_port'))+"\t"+str(jsonList['match'].get('dl_dst'))+
110                    "\t"+str(porta)+"\t"+str(jsonList['packet_count'])+"\t"+str(jsonList[
111                        'byte_count'])+"\t None"+"\t None"+"\n"
112                file.write(stringa)
113                file.close()
114                file = open(dpb_folder+"DataFlow.txt","a")
115                file.write(stringa)
116                file.close()
117                file = open(dpb_folder+"DataFlowPrec.txt","a")
118                file.write(stringa)
119                file.close()
120                if jsonList['priority']!=0 and str(jsonList['actions']).find(
121                    ('UE:'))>=0:
122                    SET_QUEUE=str(str(jsonList['actions'])[str(jsonList['
123                        actions']).find('UE:')+3:str(jsonList['actions']).find(',',)-1])
124                    file = open("/home/"+computername+"/Scrivania/
RyuDatapathMonitor-master/DataLog/"+date+"flowstat.txt","a")
125                    now=datetime.datetime.now()
126                    stringa=str(now)+"\t"+datapath_in+"\t"+str(jsonList[
127                        'match'].get('in_port'))+"\t"+str(jsonList['match'].get('nw_dst'))+
128                            "\t"+None+"\t"+str(jsonList['packet_count'])+"\t"+str(jsonList[
129                                'byte_count'])+"\t"+str(jsonList['match'].get('ip_dscp'))+"\t"+
SET_QUEUE+"\n"
130                    file.write(stringa)
131                    file.close()
132                    file = open(dpb_folder+"DataFlow.txt","a")

```

```

124         file . write( stringa )
125         file . close()
126         file = open(dpb_folder+”DataFlowPrec . txt”, ”a”)
127         file . write( stringa )
128         file . close()
129
130     except:
131         print ”FlowStat: No NET”
132
133 def save_port_stat(datapath):
134     datapath_in=datapath
135     mom_datapath= [’0’ for i in range(16-len(datapath)) ]
136     mom_datapath= ’. join(mom_datapath)
137     datapath=mom_datapath+datapath
138
139     try:
140         output = subprocess . check_output(
141             ”curl -X GET http://localhost:8080/stats/port/”+datapath ,
142             shell=True)
143         i=0
144         output = output[output . find(”{”)+1:]
145         end_response = output . find(”}”)+2
146         list1=list(output)
147         list1 [end_response -2]=’}’,
148         list1 [end_response -1]=’,’
149         output=’’. join(list1)
150         while i<end_response:
151             output_i=output[i:]
152             i=i+output[i:]. find(”},”)+2
153             otp = output_i[output_i . find(”{”):]
154             otp = otp[0:otp . find(”},”)+1]
155             otp =eval(otp)
156             data = otp
157             json_str = json . dumps(data)
158             jsonList = json . loads(json_str)
159             if jsonList[ ‘port_no’ ]!=”LOCAL”:
160                 file = open(”/home/”+computername+”/Scrivania/
RyuDatapathMonitor-master/DataLog/”+date+”portstat.txt”, ”a”)
161                 now=datetime . datetime . now()
162                 stringa=str(now)+”\t”+datapath_in+”\t”+str(jsonList[ ‘
163                 port_no’ ])+”\t”+str(jsonList[ ‘rx_packets’ ])+”\t”+str(jsonList[ ‘
164                 rx_bytes’ ])+”\t”+str(jsonList[ ‘rx_errors’ ])+”\t”+str(jsonList[ ‘
165                 tx_packets’ ])+”\t”+str(jsonList[ ‘tx_bytes’ ])+”\t”+str(jsonList[ ‘
166                 tx_errors’ ])+”\t”+str(jsonList[ ‘rx_dropped’ ])+”\t”+str(jsonList[ ‘
167                 tx_dropped’ ])+”\t”+str(jsonList[ ‘rx_crc_err’ ])+”\t”+str(jsonList[ ‘
168                 collisions’ ])+”\n”
169                 file . write( stringa )
170                 file . close()

```

```

171     except:
172         print "PortStat: No NET"
173
174 def save_queue_stat(datapath):
175     datapath_in=datapath
176     mom_datapath= ['0' for i in range(16-len(datapath))]
177     mom_datapath=''.join(mom_datapath)
178     datapath=mom_datapath+datapath
179     try:
180         output = subprocess.check_output(
181             "curl -X GET http://localhost:8080/qos/queue/status /"+
182             datapath,
183             shell=True)
184         i=0
185         output = output[output.find("ult")+1:]
186         output = output[output.find("{")+1:]
187         end_response = output.find("}]})"+2
188         list1=list(output)
189         list1[end_response-2]='}'
190         list1[end_response-1]=','
191         output=''.join(list1)
192         if output[output.find(":")+2:output.find(":")+4]!="[]":
193             while i<end_response:
194                 output_i=output[i:]
195                 i=i+output[i:].find("},")+2
196                 otp = output_i[output_i.find("{"):]
197                 otp = otp[0:otp.find("},")]+1
198                 otp = eval(otp)
199                 data = otp
200                 json_str = json.dumps(data)
201                 jsonList = json.loads(json_str)
202                 file = open("/home/"+computername+"/Scrivania/
RyuDatapathMonitor-master/DataLog/"+date+"queuestat.txt","a")
203                 now=datetime.datetime.now()
204                 stringa=str(now)+"\t"+datapath_in+"\t"+str(jsonList['
port_no'])+"\t"+str(jsonList['queue_id'])+"\t"+str(jsonList['
tx_bytes'])+"\t"+str(jsonList['tx_packets'])+"\t"+str(jsonList['
tx_errors'])+"\t"+str(jsonList['duration_sec'])+"\t"+str(jsonList['
duration_nsec'])+"\n"
205                 file.write(stringa)
206                 file.close()
207                 file = open(dpb_folder+"DataQueueStat.txt","a")
208                 file.write(stringa)
209                 file.close()
210                 file = open(dpb_folder+"DataQueueStatPrec.txt","a")
211                 file.write(stringa)
212                 file.close()
213
214     except:
215         print "QueueStat: No NET"
216
217 def save_queue_config(datapath):
218     datapath_in=datapath
219     mom_datapath= ['0' for i in range(16-len(datapath))]
```

```

219 mom_datapath=''.join(mom_datapath)
220 datapath=mom_datapath+datapath
221 try:
222     output = subprocess.check_output(
223         "curl -X GET http://localhost:8080/qos/queue/" + datapath ,
224         shell=True)
225     i=0
226     output=output[1:len(output)-1]
227     jsonList = json.loads(output)
228     config = jsonList['command_result'].get('details')
229     for queue in config:
230         if queue=='2':
231             rate = jsonList['command_result'].get('details').get(
232                 queue).get('config').get('min-rate')
233             type_of_rule='min_rate'
234         else:
235             rate = jsonList['command_result'].get('details').get(
236                 queue).get('config').get('max-rate')
237             type_of_rule='max_rate'
238             file = open("/home/" + computername + "/Scrivania/
RyuDatapathMonitor-master/DataLog/" + date + "queueconfig.txt","a")
239             now=datetime.datetime.now()
240             stringa=str(now)+"\t"+datapath_in+"\t"+str(queue)+"\t"+str(
241                 type_of_rule)+"\t"+str(rate)+"\n"
242             file.write(stringa)
243             file.close()
244             file = open(dpb_folder+"DataQueueConfig.txt","a")
245             file.write(stringa)
246             file.close()
247
248 except:
249     print "QueueConfig: No NET"
250
251 class DatapathMonitor():
252     OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]
253
254     def __init__(self, args):
255         self.datapath_list = args["dp"]
256
257         self.monitor = threading.Thread(target=self.
switch_monitor_thread)
258         self.delta = args["step"]
259         self.logger = args["logger"]
260
261         self.started = False
262
263     def start(self):
264         self.monitor.start()
265         self.started = True
266
267     def update(self, dplist):

```

```

268     self.datapath_list = dplist
269
270 def switch_monitor_thread(self):
271     max_rate_queue=100#Mps
272     max_rate_queue=max_rate_queue*1000000
273     minute_wait=20
274     Time_queue=minute_wait*60
275     Change_flag=Time_queue / self.delta
276     counter=Change_flag
277     c_q2=0
278     c_q1=1
279     c_q0=1
280     q2=np.arange(70, 101, 10)*1000000
281     q1=np.arange(0, 101, 10)*1000000
282     q0=np.arange(0, 101, 10)*1000000
283
284     print 'Wait for time alignment'
285     wait=self.delta/60
286     check_time=False
287     while check_time==False:
288         now=datetime.datetime.now()
289         if now.minute%wait==0:
290             check_time=True
291         else:
292             time.sleep(1)
293     print 'Starting Save Data'
294     check_time=False
295     flag=0
296     while True:
297         while check_time==False:
298             now=datetime.datetime.now()
299             if now.minute%wait==0:
300                 check_time=True
301                 print "Save Time: "+str(now)
302             else:
303                 try:
304                     time.sleep(1)
305                     f = open(dpb_folder+"FlagU.txt","r")
306                     flag=int(f.read())
307                 except:
308                     flag=0
309                 try:
310                     if flag==1:#check flag u file
311                         time.sleep(10)
312                         print("Change Queues Bw%")
313                         f.close()
314                         #Read new queue BW
315                         i=0
316                         u=[0,0,0]
317                         with open(dpb_folder+"U.txt") as mytxt:
318                             for line in mytxt:
319                                 print (line)
320                                 u[i]=int(line)*1000000
321                                 i=i+1

```

```

322             #Set New Queue
323             set_queue("2", "s2-eth2", str(max_rate_queue
324 ), {"max_rate": ""+str(u[0])+"", {"max_rate": ""+str(u[1])+
325 "}, {"min_rate": ""+str(u[2])+""})
326             f = open(dpb_folder+"FlagU.txt","w")
327             f.write("0")
328             f.close()
329         except:
330             print("BW File NOT READY")
331 NET=get_switchis()
332 t_sleep = 0.9
333 if NET != "NO NET" and NET!="[ , ]":
334     file = open(dpb_folder+"DataFlowPrec.txt","r")
335     Prec=file.read()
336     file.close()
337     file = open(dpb_folder+"DataFlow.txt","w")
338     file.write(Prec)
339     file.close()
340     file = open(dpb_folder+"DataFlowPrec.txt","w")
341     file.close()
342
343     file = open(dpb_folder+"DataPortPrec.txt","r")
344     Prec=file.read()
345     file.close()
346     file = open(dpb_folder+"DataPort.txt","w")
347     file.write(Prec)
348     file.close()
349     file = open(dpb_folder+"DataPortPrec.txt","w")
350     file.close()
351
352     file = open(dpb_folder+"DataQueueStatPrec.txt","r")
353     Prec=file.read()
354     file.close()
355     file = open(dpb_folder+"DataQueueStat.txt","w")
356     file.write(Prec)
357     file.close()
358     file = open(dpb_folder+"DataQueueStatPrec.txt","w")
359     file.close()
360
361     file = open(dpb_folder+"DataQueueConfigPrec.txt","r")
362     Prec=file.read()
363     file.close()
364     file = open(dpb_folder+"DataQueueConfig.txt","w")
365     file.write(Prec)
366     file.close()
367     file = open(dpb_folder+"DataQueueConfigPrec.txt","w")
368     file.close()
369     i=1
370     while i<NET.find("]"):
371         mom_NET=NET[i:]
372         datapath=NET[i:i+mom_NET.find(",")]

```

```

374     i=i+mom_NET.find(”,”)+2
375     save_flow_stat(datapath)
376     save_port_stat(datapath)
377     save_queue_stat(datapath)
378     save_queue_config(datapath)
379     file = open(dp_b_folder+”FlagData.txt”,”w”)
380     file.write(“1”)
381     file.close()
382     while check_time==True:
383         now=datetime.datetime.now()
384         if now.minute%wait!=0:
385             check_time=False
386         else:
387             time.sleep(1)
388             try:
389                 f = open(dp_b_folder+”FlagU.txt”,”r”)
390                 flag=int(f.read())
391             except:
392                 flag=0
393                 continue
394             try:
395                 if flag==1:#check flag u file
396                     print(“Change Queues Bw%”)
397                     f.close()
398                     #Read new queue BW
399                     i=0
400                     u=[0,0,0]
401                     with open(dp_b_folder+”U.txt”) as mytxt:
402                         for line in mytxt:
403                             print(line)
404                             u[i]=int(line)*1000000
405                             i=i+1
406                     #Set New Queue
407                     set_queue(”2”, ”s2-eth2”, str(
408                         max_rate_queue), ”{”+”max_rate”: ””+str(u[0])+””}, {”+”max_rate”: ””+str(u[1])+””}, {”+”min_rate”: ””+str(u[2])+””}”)
409                     f = open(dp_b_folder+”FlagU.txt”,”w”)
410                     f.write(”0”)
411                     f.close()
412             else:
413                 f.close()
414             except:
415                 print(“INPUT NOT READY”)
416                 continue
417             else:
418                 print ”No Network”

```

A.3 Controller_commands.py

```

1 from subprocess import call
2 import threading

```

```

3 import subprocess
4 import random
5 import os
6 import time
7 import json
8 import sys
9
10 def ovsdb_addr(datapath):
11     datapath_in=datapath
12     mom_datapath= ['0' for i in range(16-len(datapath))]
13     mom_datapath=''.join(mom_datapath)
14     datapath=mom_datapath+datapath
15     print "Set ovsdb on switch "+datapath
16     try:
17         os.popen("sudo -S curl -X PUT -d '\\"tcp:127.0.0.1:6632\\"' http://localhost:8080/v1.0/conf/switches/"+datapath+"/ovsdb_addr", 'w').write("Ao70pa45")
18         print "\n"
19         time.sleep(2)
20     except:
21         print "ovsdb: ERROR"
22
23 def ovssctl_set_bridge(switch_name):
24     print "Set ovssctl on switch "+switch_name
25     try:
26         os.popen("sudo -S ovs-vsctl set Bridge "+switch_name+" protocols=OpenFlow13", 'w').write("Ao70pa45")
27         print "\n"
28     except:
29         print "ovssctl_set_bridge: ERROR"
30
31 def get_switchis():
32     print "Get switches id"
33     try:
34         output = subprocess.check_output(
35             "curl -X GET http://localhost:8080/stats/switches",
36             stderr=subprocess.STDOUT,
37             shell=True)
38         output=output[output.find("["):]
39         end_response=output.find("]")
40         list1=list(output)
41         list1[end_response]=','
42         output=''.join(list1)+"]"
43         print "\n"
44     except:
45         output="NO NET"
46     return output
47
48 def switch_ports_name(datapath):
49     datapath_in=datapath
50     mom_datapath= ['0' for i in range(16-len(datapath))]
51     mom_datapath=''.join(mom_datapath)
52     datapath=mom_datapath+datapath
53     print "Get names on switch "+datapath

```

```

54     try:
55         output = subprocess.check_output(
56             "curl -X GET http://localhost:8080/stats/portdesc/" +
57             datapath,
58             stderr=subprocess.STDOUT,
59             shell=True)
60         i=0
61         output = output[output.find("{") + 1:]
62         end_response = output.find("}") }" ) + 2
63         list1=list(output)
64         list1 [end_response - 2] = '} '
65         list1 [end_response - 1] = ',' '
66         output=''.join(list1)
67         names = []
68         while i < end_response:
69             output_i=output[i:]
70             i=i+output[i:].find("},") + 2
71             otp = output_i[output_i.find("{") : ]
72             otp = otp[0:otp.find("},") + 1]
73             otp = eval(otp)
74             data = otp
75             json_str = json.dumps(data)
76             jsonList = json.loads(json_str)
77             if jsonList['port_no']=="LOCAL":
78                 names.append(str(jsonList['name']))
79             else:
80                 names.append(str(jsonList['name']))
81             print "\n"
82
83
84     except:
85         print "Switch port name: ERROR"
86
87
88 def queue_rule(datapath, port_number, ip_dscp, queue_number):
89     mom_datapath= ['0' for i in range(16-len(datapath))]
90     mom_datapath=''.join(mom_datapath)
91     datapath=mom_datapath+datapath
92     print "Set queue rule on switch "+datapath+" on port "+port_number
93     try:
94         os.popen("curl -X POST -d '{\"match\": {\"ip_dscp\": \""+ip_dscp+
95         "\", \"actions\":{\"queue\": \""+queue_number+"\"}}}', http://
96         localhost:8080/qos/rules/" + datapath, 'w').write("Ao70pa45")
97         time.sleep(0.1)
98         print "\n"
99     except:
100        print "Set queue rule: Error"
101
102
103 def queue_rule_byIP(datapath, port_number, ip_dscp, queue_number, ip_dst):
104     mom_datapath= ['0' for i in range(16-len(datapath))]
105     mom_datapath=''.join(mom_datapath)
106     datapath=mom_datapath+datapath

```

```

104     print "Set queue rule on switch "+datapath+" on port "+port_number
105     try :
106         os.popen("curl -X POST -d '{\"match\": {\"nw_dst\": \"\""+ip_dst+""
107             , \"ip_dscp\": \"\""+ip_dscp+"\""}, \"actions\":[{\"queue\": \""
108             +queue_number+"\"} }' http://localhost:8080/qos/rules/"+datapath , 'w')
109         .write("Ao70pa45")
110         time.sleep(0.1)
111         print "\n"
112     except :
113         print "Set queue rule: Error"
114
115 def set_queue(datapath , port_id , max_rate , queue_rate_list):
116     mom_datapath= ['0' for i in range(16-len(datapath))]
117     mom_datapath= ''.join(mom_datapath)
118     datapath=mom_datapath+datapath
119     print "Set queue on port "+port_id+" of switch "+datapath
120     print queue_rate_list
121     try :
122         output = subprocess.check_output("curl -X POST -d '{\"port_name\":
123             \"\""+port_id+"\", \"type\": \"linux-htb\", \"max_rate\": \""
124             +max_rate+"\", \"queues\": [\""+queue_rate_list+"\"]}' http://localhost
125             :8080/qos/queue/"+datapath ,
126             stderr=subprocess.STDOUT,
127             shell=True)
128         time.sleep(0.1)
129         print "\n"
130     except :
131         print "Set queue: Error"
132
133 def set_Telecom_queue(datapath , port_number , IP_flag , IP_dst):
134     port=port_number
135     mom_datapath= ['0' for i in range(16-len(datapath))]
136     mom_datapath= ''.join(mom_datapath)
137     datapath=mom_datapath+datapath
138     if IP_flag==True:
139         queue_rule_byIP(datapath , port , "0" , "0" , IP_dst)#Service 0
140         queue_rule_byIP(datapath , port , "8" , "0" , IP_dst)#Service 1
141         queue_rule_byIP(datapath , port , "10" , "0" , IP_dst)#Service 1
142         queue_rule_byIP(datapath , port , "12" , "0" , IP_dst)#Service 1
143         queue_rule_byIP(datapath , port , "14" , "0" , IP_dst)#Service 1
144         queue_rule_byIP(datapath , port , "24" , "0" , IP_dst)#Service 3
145         queue_rule_byIP(datapath , port , "26" , "0" , IP_dst)#Service 3
146         queue_rule_byIP(datapath , port , "28" , "0" , IP_dst)#Service 3
147         queue_rule_byIP(datapath , port , "30" , "0" , IP_dst)#Service 3
148
149         queue_rule_byIP(datapath , port , "16" , "1" , IP_dst)#Service 2
150         queue_rule_byIP(datapath , port , "18" , "1" , IP_dst)#Service 2
151         queue_rule_byIP(datapath , port , "20" , "1" , IP_dst)#Service 2
152         queue_rule_byIP(datapath , port , "22" , "1" , IP_dst)#Service 2
153         queue_rule_byIP(datapath , port , "32" , "1" , IP_dst)#Service 4
154         queue_rule_byIP(datapath , port , "34" , "1" , IP_dst)#Service 4
155         queue_rule_byIP(datapath , port , "36" , "1" , IP_dst)#Service 4
156         queue_rule_byIP(datapath , port , "38" , "1" , IP_dst)#Service 4

```

```

152     queue_rule_byIP(datapath, port, "48", "1", IP_dst)#Service 6
153     queue_rule_byIP(datapath, port, "56", "1", IP_dst)#Service 7
154
155     queue_rule_byIP(datapath, port, "40", "2", IP_dst)#Service 5
156     queue_rule_byIP(datapath, port, "46", "2", IP_dst)#Service 5
157
158 if IP_flag==False:
159     #Default Queue (queue_id = 0)
160     queue_rule(datapath, port, "0", "0")#Service 0
161     queue_rule(datapath, port, "8", "0")#Service 1
162     queue_rule(datapath, port, "10", "0")#Service 1
163     queue_rule(datapath, port, "12", "0")#Service 1
164     queue_rule(datapath, port, "14", "0")#Service 1
165     queue_rule(datapath, port, "24", "0")#Service 3
166     queue_rule(datapath, port, "26", "0")#Service 3
167     queue_rule(datapath, port, "28", "0")#Service 3
168     queue_rule(datapath, port, "30", "0")#Service 3
169     #Premium Queue (queue_id = 1)
170     queue_rule(datapath, port, "16", "1")#Service 2
171     queue_rule(datapath, port, "18", "1")#Service 2
172     queue_rule(datapath, port, "20", "1")#Service 2
173     queue_rule(datapath, port, "22", "1")#Service 2
174     queue_rule(datapath, port, "32", "1")#Service 4
175     queue_rule(datapath, port, "34", "1")#Service 4
176     queue_rule(datapath, port, "36", "1")#Service 4
177     queue_rule(datapath, port, "38", "1")#Service 4
178     queue_rule(datapath, port, "48", "1")#Service 6
179     queue_rule(datapath, port, "56", "1")#Service 7
180     #Gold Queue (queue_id = 2)
181     queue_rule(datapath, port, "40", "2")#Service 5
182     queue_rule(datapath, port, "46", "2")#Service 5

```

A.4 qos_simple_switch_13.py

```

1 # Copyright (C) 2011 Nippon Telegraph and Telephone Corporation.
2 #
3 # Licensed under the Apache License, Version 2.0 (the "License");
4 # you may not use this file except in compliance with the License.
5 # You may obtain a copy of the License at
6 #
7 #     http://www.apache.org/licenses/LICENSE-2.0
8 #
9 # Unless required by applicable law or agreed to in writing, software
10 # distributed under the License is distributed on an "AS IS" BASIS,
11 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
12 # implied.
13 # See the License for the specific language governing permissions and
14 # limitations under the License.
15
16 from ryu.base import app_manager
17 from ryu.controller import ofp_event

```

```

18 from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER
19 from ryu.controller.handler import set_ev_cls
20 from ryu.ofproto import ofproto_v1_3
21 from ryu.lib.packet import packet
22 from ryu.lib.packet import ethernet
23 from ryu.lib.packet import ether_types
24
25 import subprocess
26
27
28 class SimpleSwitch13(app_manager.RyuApp):
29     OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]
30
31     def __init__(self, *args, **kwargs):
32         super(SimpleSwitch13, self).__init__(*args, **kwargs)
33         self.mac_to_port = {}
34
35     @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
36     def switch_features_handler(self, ev):
37         datapath = ev.msg.datapath
38         ofproto = datapath.ofproto
39         parser = datapath.ofproto_parser
40
41         # install table-miss flow entry
42         #
43         # We specify NO BUFFER to max_len of the output action due to
44         # OVS bug. At this moment, if we specify a lesser number, e.g.,
45         # 128, OVS will send Packet-In with invalid buffer_id and
46         # truncated packet data. In that case, we cannot output packets
47         # correctly. The bug has been fixed in OVS v2.1.0.
48         match = parser.OFPMatch()
49         actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
50                                         ofproto.OFPCML_NO_BUFFER)]
51         self.add_flow(datapath, 0, match, actions)
52
53     def add_flow(self, datapath, priority, match, actions, buffer_id=None):
54         ofproto = datapath.ofproto
55         parser = datapath.ofproto_parser
56
57         inst = [parser.OFFInstructionActions(ofproto.OFPIT_APPLY_ACTIONS
58                                              ,
59                                              actions)]
60         if buffer_id:
61             mod = parser.OFPFlowMod(datapath=datapath, buffer_id=buffer_id,
62                                     priority=priority, match=match,
63                                     instructions=inst, table_id=1)
64         else:
65             mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
66                                     , match=match, instructions=inst,
67                                     table_id=1)
68         datapath.send_msg(mod)

```

```

67
68     @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
69     def _packet_in_handler(self, ev):
70         # If you hit this you might want to increase
71         # the "miss_send_length" of your switch
72         if ev.msg.msg_len < ev.msg.total_len:
73             self.logger.debug("packet truncated: only %s of %s bytes",
74                               ev.msg.msg_len, ev.msg.total_len)
75         msg = ev.msg
76         datapath = msg.datapath
77         ofproto = datapath.ofproto
78         parser = datapath.ofproto_parser
79         in_port = msg.match['in_port']
80
81         pkt = packet.Packet(msg.data)
82         eth = pkt.get_protocols(ether_types.ETHERNET)[0]
83
84         if eth.ethertype == ether_types.ETH_TYPE_LLDP:
85             # ignore lldp packet
86             return
87         dst = eth.dst
88         src = eth.src
89
90         dpid = datapath.id
91         self.mac_to_port.setdefault(dpid, {})
92
93         # learn a mac address to avoid FLOOD next time.
94         self.mac_to_port[dpid][src] = in_port
95
96         if dst in self.mac_to_port[dpid]:
97             out_port = self.mac_to_port[dpid][dst]
98         else:
99             out_port = ofproto.OFPP_FLOOD
100
101        actions = [parser.OFPActionOutput(out_port)]
102
103        # install a flow to avoid packet_in next time
104        if out_port != ofproto.OFPP_FLOOD:
105            match = parser.OFPMatch(in_port=in_port, eth_dst=dst,
106                                     eth_src=src)
107            # verify if we have a valid buffer_id, if yes avoid to send
108            # both
109            # flow_mod & packet_out
110            if msg.buffer_id != ofproto.OFP_NO_BUFFER:
111                self.add_flow(datapath, 1, match, actions, msg.buffer_id)
112            return
113        else:
114            self.add_flow(datapath, 1, match, actions)
115        data = None
116        if msg.buffer_id == ofproto.OFP_NO_BUFFER:
117            data = msg.data

```

```

117         out = parser.OFPPacketOut(datapath=datapath, buffer_id=msg.
118                                     buffer_id,
119                                     in_port=in_port, actions=actions, data
120                                     =data)
121         datapath.send_msg(out)

```

A.5 ofctl_rest.py

```

1 # Copyright (C) 2012 Nippon Telegraph and Telephone Corporation.
2 #
3 # Licensed under the Apache License, Version 2.0 (the "License");
4 # you may not use this file except in compliance with the License.
5 # You may obtain a copy of the License at
6 #
7 #     http://www.apache.org/licenses/LICENSE-2.0
8 #
9 # Unless required by applicable law or agreed to in writing, software
10 # distributed under the License is distributed on an "AS IS" BASIS,
11 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
12 # implied.
13 # See the License for the specific language governing permissions and
14 # limitations under the License.
15
16 import logging
17 import json
18 import ast
19
20 from ryu.base import app_manager
21 from ryu.controller import ofp_event
22 from ryu.controller import dpset
23 from ryu.controller.handler import MAIN_DISPATCHER
24 from ryu.controller.handler import set_ev_cls
25 from ryu.exception import RyuException
26 from ryu.ofproto import ofproto_v1_0
27 from ryu.ofproto import ofproto_v1_2
28 from ryu.ofproto import ofproto_v1_3
29 from ryu.ofproto import ofproto_v1_4
30 from ryu.ofproto import ofproto_v1_5
31 from ryu.lib import ofctl_v1_0
32 from ryu.lib import ofctl_v1_2
33 from ryu.lib import ofctl_v1_3
34 from ryu.lib import ofctl_v1_4
35 from ryu.lib import ofctl_v1_5
36 from ryu.app.wsgi import ControllerBase
37 from ryu.app.wsgi import Response
38 from ryu.app.wsgi import WSGIApplication
39
40 LOG = logging.getLogger('ryu.app.ofctl_rest')
41
42 # supported ofctl versions in this restful app
43 supported_ofctl = {

```

```

44     ofproto_v1_0.OFP_VERSION: ofctl_v1_0 ,
45     ofproto_v1_2.OFP_VERSION: ofctl_v1_2 ,
46     ofproto_v1_3.OFP_VERSION: ofctl_v1_3 ,
47     ofproto_v1_4.OFP_VERSION: ofctl_v1_4 ,
48     ofproto_v1_5.OFP_VERSION: ofctl_v1_5 ,
49 }
50
51 # REST API
52 #
53
54 # Retrieve the switch stats
55 #
56 # get the list of all switches
57 # GET /stats/switches
58 #
59 # get the desc stats of the switch
60 # GET /stats/desc/<dpid>
61 #
62 # get flows desc stats of the switch
63 # GET /stats/flowdesc/<dpid>
64 #
65 # get flows desc stats of the switch filtered by the fields
66 # POST /stats/flowdesc/<dpid>
67 #
68 # get flows stats of the switch
69 # GET /stats/flow/<dpid>
70 #
71 # get flows stats of the switch filtered by the fields
72 # POST /stats/flow/<dpid>
73 #
74 # get aggregate flows stats of the switch
75 # GET /stats/aggregateflow/<dpid>
76 #
77 # get aggregate flows stats of the switch filtered by the fields
78 # POST /stats/aggregateflow/<dpid>
79 #
80 # get table stats of the switch
81 # GET /stats/table/<dpid>
82 #
83 # get table features stats of the switch
84 # GET /stats/tablefeatures/<dpid>
85 #
86 # get ports stats of the switch
87 # GET /stats/port/<dpid>[/<port>]
88 # Note: Specification of port number is optional
89 #
90 # get queues stats of the switch
91 # GET /stats/queue/<dpid>[/<port>[/<queue_id>]]
92 # Note: Specification of port number and queue id are optional
93 #       If you want to omitting the port number and setting the queue id
94 #       ,
95 #           please specify the keyword "ALL" to the port number
96 #           e.g. GET /stats/queue/1/ALL/1

```

```

97 # get queues config stats of the switch
98 # GET /stats/queueconfig/<dpid>[/<port>]
99 # Note: Specification of port number is optional
100 #
101 # get queues desc stats of the switch
102 # GET /stats/queuedesc/<dpid>[/<port>[/<queue_id>]]
103 # Note: Specification of port number and queue id are optional
104 #       If you want to omitting the port number and setting the queue id
105     ,
106     #       please specify the keyword "ALL" to the port number
107     #       e.g. GET /stats/queuedesc/1/ALL/1
108 #
109 # get meter features stats of the switch
110 # GET /stats/meterfeatures/<dpid>
111 #
112 # get meter config stats of the switch
113 # GET /stats/meterconfig/<dpid>[/<meter_id>]
114 # Note: Specification of meter id is optional
115 #
116 # get meter desc stats of the switch
117 # GET /stats/meterdesc/<dpid>[/<meter_id>]
118 # Note: Specification of meter id is optional
119 #
120 # get meters stats of the switch
121 # GET /stats/meter/<dpid>[/<meter_id>]
122 # Note: Specification of meter id is optional
123 #
124 # get group features stats of the switch
125 # GET /stats/groupfeatures/<dpid>
126 #
127 # get groups desc stats of the switch
128 # GET /stats/groupdesc/<dpid>[/<group_id>]
129 # Note: Specification of group id is optional (OpenFlow 1.5 or later)
130 #
131 # get groups stats of the switch
132 # GET /stats/group/<dpid>[/<group_id>]
133 # Note: Specification of group id is optional
134 #
135 # get ports description of the switch
136 # GET /stats/portdesc/<dpid>[/<port_no>]
137 # Note: Specification of port number is optional (OpenFlow 1.5 or later)
138 #
139 # Update the switch stats
140 #
141 # add a flow entry
142 # POST /stats/flowentry/add
143 #
144 # modify all matching flow entries
145 # POST /stats/flowentry/modify
146 #
147 # modify flow entry strictly matching wildcards and priority
148 # POST /stats/flowentry/modify_strict
149 #
150 # delete all matching flow entries

```

```

150 # POST / stats / flowentry / delete
151 #
152 # delete flow entry strictly matching wildcards and priority
153 # POST / stats / flowentry / delete strict
154 #
155 # delete all flow entries of the switch
156 # DELETE / stats / flowentry / clear / <dpid>
157 #
158 # add a meter entry
159 # POST / stats / meterentry / add
160 #
161 # modify a meter entry
162 # POST / stats / meterentry / modify
163 #
164 # delete a meter entry
165 # POST / stats / meterentry / delete
166 #
167 # add a group entry
168 # POST / stats / groupentry / add
169 #
170 # modify a group entry
171 # POST / stats / groupentry / modify
172 #
173 # delete a group entry
174 # POST / stats / groupentry / delete
175 #
176 # modify behavior of the physical port
177 # POST / stats / portdesc / modify
178 #
179 # modify role of controller
180 # POST / stats / role
181 #
182 #
183 # send a experimenter message
184 # POST / stats / experimenter / <dpid>

185
186
187 class CommandNotFoundError(RyuException):
188     message = 'No such command : %(cmd)s'
189
190
191 class PortNotFoundError(RyuException):
192     message = 'No such port info: %(port_no)s'
193
194
195 def stats_method(method):
196     def wrapper(self, req, dpid, *args, **kwargs):
197         # Get datapath instance from DPSet
198         try:
199             dp = self.dpset.get(int(str(dpid), 0))
200         except ValueError:
201             LOG.exception('Invalid dpid: %s', dpid)
202             return Response(status=400)
203         if dp is None:

```

```

204     LOG.error('No such Datapath: %s', dpid)
205     return Response(status=404)
206
207     # Get lib/ofctl_* module
208     try:
209         ofctl = supported_ofctl.get(dp.ofproto.OFP_VERSION)
210     except KeyError:
211         LOG.exception('Unsupported OF version: %s',
212                         dp.ofproto.OFP_VERSION)
213         return Response(status=501)
214
215     # Invoke StatsController method
216     try:
217         ret = method(self, req, dp, ofctl, *args, **kwargs)
218         return Response(content_type='application/json',
219                         body=json.dumps(ret))
220     except ValueError:
221         LOG.exception('Invalid syntax: %s', req.body)
222         return Response(status=400)
223     except AttributeError:
224         LOG.exception('Unsupported OF request in this version: %s',
225                         dp.ofproto.OFP_VERSION)
226         return Response(status=501)
227
228     return wrapper
229
230
231 def command_method(method):
232     def wrapper(self, req, *args, **kwargs):
233         # Parse request json body
234         try:
235             if req.body:
236                 # We use ast.literal_eval() to parse request json body
237                 # instead of json.loads().
238                 # Because we need to parse binary format body
239                 # in send_experimenter().
240                 body = ast.literal_eval(req.body.decode('utf-8'))
241             else:
242                 body = {}
243         except SyntaxError:
244             LOG.exception('Invalid syntax: %s', req.body)
245             return Response(status=400)
246
247         # Get datapath_id from request parameters
248         dpid = body.get('dpid', None)
249         if not dpid:
250             try:
251                 dpid = kwargs.pop('dpid')
252             except KeyError:
253                 LOG.exception('Cannot get dpid from request parameters')
254                 return Response(status=400)
255
256         # Get datapath instance from DPSet
257         try:

```

```

258         dp = self.dpset.get(int(str(dpid), 0))
259     except ValueError:
260         LOG.exception('Invalid dpid: %s', dpid)
261         return Response(status=400)
262     if dp is None:
263         LOG.error('No such Datapath: %s', dpid)
264         return Response(status=404)
265
266     # Get lib/ofctl_* module
267     try:
268         ofctl = supported_ofctl.get(dp.ofproto.OFP_VERSION)
269     except KeyError:
270         LOG.exception('Unsupported OF version: version=%s',
271                         dp.ofproto.OFP_VERSION)
272         return Response(status=501)
273
274     # Invoke StatsController method
275     try:
276         method(self, req, dp, ofctl, body, *args, **kwargs)
277         return Response(status=200)
278     except ValueError:
279         LOG.exception('Invalid syntax: %s', req.body)
280         return Response(status=400)
281     except AttributeError:
282         LOG.exception('Unsupported OF request in this version: %s',
283                         dp.ofproto.OFP_VERSION)
284         return Response(status=501)
285     except CommandNotFoundError as e:
286         LOG.exception(e.message)
287         return Response(status=404)
288     except PortNotFoundError as e:
289         LOG.exception(e.message)
290         return Response(status=404)
291
292     return wrapper
293
294
295 class StatsController(ControllerBase):
296     def __init__(self, req, link, data, **config):
297         super(StatsController, self).__init__(req, link, data, **config)
298         self.dpset = data['dpset']
299         self.waiters = data['waiters']
300
301     def get_dpids(self, req, **kwargs):
302         dps = list(self.dpset.dps.keys())
303         body = json.dumps(dps)
304         return Response(content_type='application/json', body=body)
305
306     @stats_method
307     def get_desc_stats(self, req, dp, ofctl, **kwargs):
308         return ofctl.get_desc_stats(dp, self.waiters)
309
310     @stats_method
311     def get_flow_desc(self, req, dp, ofctl, **kwargs):

```

```

312     flow = req.json if req.body else {}
313     return ofctl.get_flow_desc(dp, self.waiters, flow)
314
315     @stats_method
316     def get_flow_stats(self, req, dp, ofctl, **kwargs):
317         flow = req.json if req.body else {}
318         return ofctl.get_flow_stats(dp, self.waiters, flow)
319
320     @stats_method
321     def get_aggregate_flow_stats(self, req, dp, ofctl, **kwargs):
322         flow = req.json if req.body else {}
323         return ofctl.get_aggregate_flow_stats(dp, self.waiters, flow)
324
325     @stats_method
326     def get_table_stats(self, req, dp, ofctl, **kwargs):
327         return ofctl.get_table_stats(dp, self.waiters)
328
329     @stats_method
330     def get_table_features(self, req, dp, ofctl, **kwargs):
331         return ofctl.get_table_features(dp, self.waiters)
332
333     @stats_method
334     def get_port_stats(self, req, dp, ofctl, port=None, **kwargs):
335         if port == "ALL":
336             port = None
337
338         return ofctl.get_port_stats(dp, self.waiters, port)
339
340     @stats_method
341     def get_queue_stats(self, req, dp, ofctl,
342                         port=None, queue_id=None, **kwargs):
343         if port == "ALL":
344             port = None
345
346         if queue_id == "ALL":
347             queue_id = None
348
349         return ofctl.get_queue_stats(dp, self.waiters, port, queue_id)
350
351     @stats_method
352     def get_queue_config(self, req, dp, ofctl, port=None, **kwargs):
353         if port == "ALL":
354             port = None
355
356         return ofctl.get_queue_config(dp, self.waiters, port)
357
358     @stats_method
359     def get_queue_desc(self, req, dp, ofctl,
360                       port=None, queue=None, **kwargs):
361         if port == "ALL":
362             port = None
363
364         if queue == "ALL":
365             queue = None

```

```

366     return ofctl.get_queue_desc(dp, self.waiters, port, queue)
367
368     @stats_method
369     def get_meter_features(self, req, dp, ofctl, **kwargs):
370         return ofctl.get_meter_features(dp, self.waiters)
371
372     @stats_method
373     def get_meter_config(self, req, dp, ofctl, meter_id=None, **kwargs):
374         if meter_id == "ALL":
375             meter_id = None
376
377         return ofctl.get_meter_config(dp, self.waiters, meter_id)
378
379     @stats_method
380     def get_meter_desc(self, req, dp, ofctl, meter_id=None, **kwargs):
381         if meter_id == "ALL":
382             meter_id = None
383
384         return ofctl.get_meter_desc(dp, self.waiters, meter_id)
385
386     @stats_method
387     def get_meter_stats(self, req, dp, ofctl, meter_id=None, **kwargs):
388         if meter_id == "ALL":
389             meter_id = None
390
391         return ofctl.get_meter_stats(dp, self.waiters, meter_id)
392
393     @stats_method
394     def get_group_features(self, req, dp, ofctl, **kwargs):
395         return ofctl.get_group_features(dp, self.waiters)
396
397     @stats_method
398     def get_group_desc(self, req, dp, ofctl, group_id=None, **kwargs):
399         if dp.ofproto.OFP_VERSION < ofproto_v1_5.OFP_VERSION:
400             return ofctl.get_group_desc(dp, self.waiters)
401         else:
402             return ofctl.get_group_desc(dp, self.waiters, group_id)
403
404     @stats_method
405     def get_group_stats(self, req, dp, ofctl, group_id=None, **kwargs):
406         if group_id == "ALL":
407             group_id = None
408
409         return ofctl.get_group_stats(dp, self.waiters, group_id)
410
411     @stats_method
412     def get_port_desc(self, req, dp, ofctl, port_no=None, **kwargs):
413         if dp.ofproto.OFP_VERSION < ofproto_v1_5.OFP_VERSION:
414             return ofctl.get_port_desc(dp, self.waiters)
415         else:
416             return ofctl.get_port_desc(dp, self.waiters, port_no)
417
418     @stats_method

```

```

420     def get_role(self, req, dp, ofctl, **kwargs):
421         return ofctl.get_role(dp, self.waiters)
422
423     @command_method
424     def mod_flow_entry(self, req, dp, ofctl, flow, cmd, **kwargs):
425         cmd_convert = {
426             'add': dp.ofproto.OFPFC_ADD,
427             'modify': dp.ofproto.OFPFC MODIFY,
428             'modify strict': dp.ofproto.OFPFC MODIFY_STRICT,
429             'delete': dp.ofproto.OFPFC_DELETE,
430             'delete strict': dp.ofproto.OFPFC_DELETE_STRICT,
431         }
432         mod_cmd = cmd_convert.get(cmd, None)
433         if mod_cmd is None:
434             raise CommandNotFoundError(cmd=cmd)
435
436         ofctl.mod_flow_entry(dp, flow, mod_cmd)
437
438     @command_method
439     def delete_flow_entry(self, req, dp, ofctl, flow, **kwargs):
440         if ofproto_v1_0.OFP_VERSION == dp.ofproto.OFP_VERSION:
441             flow = {}
442         else:
443             flow = {'table_id': dp.ofproto.OFPTT_ALL}
444
445         ofctl.mod_flow_entry(dp, flow, dp.ofproto.OFPFC_DELETE)
446
447     @command_method
448     def mod_meter_entry(self, req, dp, ofctl, meter, cmd, **kwargs):
449         cmd_convert = {
450             'add': dp.ofproto.OFPMC_ADD,
451             'modify': dp.ofproto.OFPMC MODIFY,
452             'delete': dp.ofproto.OFPMC_DELETE,
453         }
454         mod_cmd = cmd_convert.get(cmd, None)
455         if mod_cmd is None:
456             raise CommandNotFoundError(cmd=cmd)
457
458         ofctl.mod_meter_entry(dp, meter, mod_cmd)
459
460     @command_method
461     def mod_group_entry(self, req, dp, ofctl, group, cmd, **kwargs):
462         cmd_convert = {
463             'add': dp.ofproto.OFGC_ADD,
464             'modify': dp.ofproto.OFGC MODIFY,
465             'delete': dp.ofproto.OFGC_DELETE,
466         }
467         mod_cmd = cmd_convert.get(cmd, None)
468         if mod_cmd is None:
469             raise CommandNotFoundError(cmd=cmd)
470
471         ofctl.mod_group_entry(dp, group, mod_cmd)
472
473     @command_method

```

```

474     def mod_port_behavior(self, req, dp, ofctl, port_config, cmd, **kwargs):
475         port_no = port_config.get('port_no', None)
476         port_no = int(str(port_no), 0)
477
478         port_info = self.dpset.port_state[int(dp.id)].get(port_no)
479         if port_info:
480             port_config.setdefault('hw_addr', port_info.hw_addr)
481             if dp.ofproto.OFP_VERSION < ofproto_v1_4.OFP_VERSION:
482                 port_config.setdefault('advertise', port_info.advertised)
483             else:
484                 port_config.setdefault('properties', port_info.properties)
485             else:
486                 raise PortNotFoundError(port_no=port_no)
487
488         if cmd != 'modify':
489             raise CommandNotFoundError(cmd=cmd)
490
491         ofctl.mod_port_behavior(dp, port_config)
492
493     @command_method
494     def send_experimenter(self, req, dp, ofctl, exp, **kwargs):
495         ofctl.send_experimenter(dp, exp)
496
497     @command_method
498     def set_role(self, req, dp, ofctl, role, **kwargs):
499         ofctl.set_role(dp, role)
500
501
502 class RestStatsApi(app_manager.RyuApp):
503     OFP VERSIONS = [ofproto_v1_0.OFP_VERSION,
504                     ofproto_v1_2.OFP_VERSION,
505                     ofproto_v1_3.OFP_VERSION,
506                     ofproto_v1_4.OFP_VERSION,
507                     ofproto_v1_5.OFP_VERSION]
508     _CONTEXTS = {
509         'dpset': dpset.DPSet,
510         'wsgi': WSGIApplication
511     }
512
513     def __init__(self, *args, **kwargs):
514         super(RestStatsApi, self).__init__(*args, **kwargs)
515         self.dpset = kwargs['dpset']
516         wsgi = kwargs['wsgi']
517         self.waiters = {}
518         self.data = {}
519         self.data['dpset'] = self.dpset
520         self.data['waiters'] = self.waiters
521         mapper = wsgi.mapper
522
523         wsgi.registry['StatsController'] = self.data
524         path = '/stats'

```

```

525     uri = path + '/switches'
526     mapper.connect('stats', uri,
527                     controller=StatsController, action='get_dpids',
528                     conditions=dict(method=['GET']))
529
530     uri = path + '/desc/{dpid}'
531     mapper.connect('stats', uri,
532                     controller=StatsController, action='
533 get_desc_stats',
534                     conditions=dict(method=['GET']))
535
536     uri = path + '/flowdesc/{dpid}'
537     mapper.connect('stats', uri,
538                     controller=StatsController, action='
539 get_flow_stats',
540                     conditions=dict(method=['GET', 'POST']))
541
542     uri = path + '/flow/{dpid}'
543     mapper.connect('stats', uri,
544                     controller=StatsController, action='
545 get_flow_stats',
546                     conditions=dict(method=['GET', 'POST']))
547
548     uri = path + '/aggregateflow/{dpid}'
549     mapper.connect('stats', uri,
550                     controller=StatsController,
551                     action='get_aggregate_flow_stats',
552                     conditions=dict(method=['GET', 'POST']))
553
554     uri = path + '/table/{dpid}'
555     mapper.connect('stats', uri,
556                     controller=StatsController, action='
557 get_table_stats',
558                     conditions=dict(method=['GET']))
559
560     uri = path + '/tablefeatures/{dpid}'
561     mapper.connect('stats', uri,
562                     controller=StatsController, action='
563 get_table_features',
564                     conditions=dict(method=['GET']))
565
566     uri = path + '/port/{dpid}'
567     mapper.connect('stats', uri,
568                     controller=StatsController, action='
569 get_port_stats',
570                     conditions=dict(method=['GET']))
571
572     uri = path + '/queue/{dpid}'

```

```

572     mapper.connect('stats', uri,
573                     controller=StatsController, action='
574         get_queue_stats',
575                     conditions=dict(method=[ 'GET' ]))
576
576     uri = path + '/queue/{ dpid }/{ port }'
577     mapper.connect('stats', uri,
578                     controller=StatsController, action='
579         get_queue_stats',
580                     conditions=dict(method=[ 'GET' ]))
581
581     uri = path + '/queue/{ dpid }/{ port }/{ queue_id }'
582     mapper.connect('stats', uri,
583                     controller=StatsController, action='
584         get_queue_stats',
585                     conditions=dict(method=[ 'GET' ]))
586
586     uri = path + '/queueconfig/{ dpid }'
587     mapper.connect('stats', uri,
588                     controller=StatsController, action='
589         get_queue_config',
590                     conditions=dict(method=[ 'GET' ]))
591
591     uri = path + '/queueconfig/{ dpid }/{ port }'
592     mapper.connect('stats', uri,
593                     controller=StatsController, action='
594         get_queue_config',
595                     conditions=dict(method=[ 'GET' ]))
596
596     uri = path + '/queuedesc/{ dpid }'
597     mapper.connect('stats', uri,
598                     controller=StatsController, action='
599         get_queue_desc',
600                     conditions=dict(method=[ 'GET' ]))
601
601     uri = path + '/queuedesc/{ dpid }/{ port }'
602     mapper.connect('stats', uri,
603                     controller=StatsController, action='
604         get_queue_desc',
605                     conditions=dict(method=[ 'GET' ]))
606
606     uri = path + '/queuedesc/{ dpid }/{ port }/{ queue }'
607     mapper.connect('stats', uri,
608                     controller=StatsController, action='
609         get_queue_desc',
610                     conditions=dict(method=[ 'GET' ]))
611
611     uri = path + '/meterfeatures/{ dpid }'
612     mapper.connect('stats', uri,
613                     controller=StatsController, action='
614         get_meter_features',
615                     conditions=dict(method=[ 'GET' ]))
616
616     uri = path + '/meterconfig/{ dpid }'

```

```

617     mapper.connect('stats', uri,
618                     controller=StatsController, action='
619                     get_meter_config',
620                     conditions=dict(method=['GET']))
621
622         uri = path + '/meterconfig/{dpid}/{meter_id}'
623         mapper.connect('stats', uri,
624                     controller=StatsController, action='
625                     get_meter_config',
626                     conditions=dict(method=['GET']))
627
628         uri = path + '/meterdesc/{dpid}'
629         mapper.connect('stats', uri,
630                     controller=StatsController, action='
631                     get_meter_desc',
632                     conditions=dict(method=['GET']))
633
634         uri = path + '/meterdesc/{dpid}/{meter_id}'
635         mapper.connect('stats', uri,
636                     controller=StatsController, action='
637                     get_meter_desc',
638                     conditions=dict(method=['GET']))
639
640         uri = path + '/meter/{dpid}'
641         mapper.connect('stats', uri,
642                     controller=StatsController, action='
643                     get_meter_stats',
644                     conditions=dict(method=['GET']))
645
646         uri = path + '/groupfeatures/{dpid}'
647         mapper.connect('stats', uri,
648                     controller=StatsController, action='
649                     get_group_features',
650                     conditions=dict(method=['GET']))
651
652         uri = path + '/groupdesc/{dpid}'
653         mapper.connect('stats', uri,
654                     controller=StatsController, action='
655                     get_group_desc',
656                     conditions=dict(method=['GET']))
657
658         uri = path + '/groupdesc/{dpid}/{group_id}'
659         mapper.connect('stats', uri,
660                     controller=StatsController, action='
661                     get_group_desc',
662                     conditions=dict(method=['GET']))
663
664         uri = path + '/group/{dpid}'

```

```

662     mapper.connect('stats', uri,
663                     controller=StatsController, action='
664         get_group_stats',
665                     conditions=dict(method=['GET']))
666
667     uri = path + '/group/{dpid}/{group_id}'
668     mapper.connect('stats', uri,
669                     controller=StatsController, action='
670         get_group_stats',
671                     conditions=dict(method=['GET']))
672
673     uri = path + '/portdesc/{dpid}'
674     mapper.connect('stats', uri,
675                     controller=StatsController, action='get_port_desc
676         ',
677                     conditions=dict(method=['GET']))
678
679     uri = path + '/portdesc/{dpid}/{port_no}'
680     mapper.connect('stats', uri,
681                     controller=StatsController, action='get_port_desc
682         ',
683                     conditions=dict(method=['GET']))
684
685     uri = path + '/role/{dpid}'
686     mapper.connect('stats', uri,
687                     controller=StatsController, action='get_role',
688                     conditions=dict(method=['GET']))
689
690     uri = path + '/flowentry/{cmd}'
691     mapper.connect('stats', uri,
692                     controller=StatsController, action='
693         mod_flow_entry',
694                     conditions=dict(method=['POST']))
695
696     uri = path + '/flowentry/clear/{dpid}'
697     mapper.connect('stats', uri,
698                     controller=StatsController, action='
699         delete_flow_entry',
700                     conditions=dict(method=['DELETE']))
701
702     uri = path + '/meterentry/{cmd}'
703     mapper.connect('stats', uri,
704                     controller=StatsController, action='
705         mod_meter_entry',
706                     conditions=dict(method=['POST']))
707
708     uri = path + '/groupentry/{cmd}'
709     mapper.connect('stats', uri,
710                     controller=StatsController, action='
711         mod_group_entry',
712                     conditions=dict(method=['POST']))
713
714     uri = path + '/portdesc/{cmd}'
715     mapper.connect('stats', uri,

```

```

708         controller=StatsController , action='
709     mod_port_behavior' ,
710             conditions=dict(method=[ 'POST' ]))
711
712     uri = path + '/experimenter/{dpid}'
713     mapper.connect('stats' , uri ,
714             controller=StatsController , action='
715     send_experimenter' ,
716             conditions=dict(method=[ 'POST' ]))
717
718     uri = path + '/role'
719     mapper.connect('stats' , uri ,
720             controller=StatsController , action='set_role' ,
721             conditions=dict(method=[ 'POST' ]))
722
723 @set_ev_cls([ ofp_event.EventOFPStatsReply ,
724             ofp_event.EventOFPDescStatsReply ,
725             ofp_event.EventOFPFlowStatsReply ,
726             ofp_event.EventOFPAggregateStatsReply ,
727             ofp_event.EventOFPTableStatsReply ,
728             ofp_event.EventOFPTableFeaturesStatsReply ,
729             ofp_event.EventOFPPortStatsReply ,
730             ofp_event.EventOFPQueueStatsReply ,
731             ofp_event.EventOFPQueueDescStatsReply ,
732             ofp_event.EventOFPMeterStatsReply ,
733             ofp_event.EventOFPMeterFeaturesStatsReply ,
734             ofp_event.EventOFPMeterConfigStatsReply ,
735             ofp_event.EventOFGGroupStatsReply ,
736             ofp_event.EventOFGGroupFeaturesStatsReply ,
737             ofp_event.EventOFGGroupDescStatsReply ,
738             ofp_event.EventOFPPortDescStatsReply
739             ], MAIN_DISPATCHER)
740
741 def stats_reply_handler(self , ev):
742     msg = ev.msg
743     dp = msg.datapath
744
745     if dp.id not in self.waiters:
746         return
747     if msg.xid not in self.waiters[dp.id]:
748         return
749     lock , msgs = self.waiters[dp.id][msg.xid]
750     msgs.append(msg)
751
752     flags = 0
753     if dp.ofproto.OFP_VERSION == ofproto_v1_0.OFP_VERSION:
754         flags = dp.ofproto.OFPSF_REPLY_MORE
755     elif dp.ofproto.OFP_VERSION == ofproto_v1_2.OFP_VERSION:
756         flags = dp.ofproto.OFPSF_REPLY_MORE
757     elif dp.ofproto.OFP_VERSION >= ofproto_v1_3.OFP_VERSION:
758         flags = dp.ofproto.OFPMPF_REPLY_MORE
759
760     if msg.flags & flags:
761         return
762     del self.waiters[dp.id][msg.xid]

```

```

760     lock.set()
761
762     @set_ev_cls([ofp_event.EventOFPSwitchFeatures,
763                 ofp_event.EventOFPQueueGetConfigReply,
764                 ofp_event.EventOFPRoleReply,
765                 ], MAIN_DISPATCHER)
766     def features_reply_handler(self, ev):
767         msg = ev.msg
768         dp = msg.datapath
769
770         if dp.id not in self.waiters:
771             return
772         if msg.xid not in self.waiters[dp.id]:
773             return
774         lock, msgs = self.waiters[dp.id][msg.xid]
775         msgs.append(msg)
776
777         del self.waiters[dp.id][msg.xid]
778         lock.set()

```

A.6 rest_conf_switch.py

```

1 # Copyright (C) 2012 Nippon Telegraph and Telephone Corporation.
2 # Copyright (C) 2012 Isaku Yamahata <isaku.yamahata at private email ne jp>
3 #
4 # Licensed under the Apache License, Version 2.0 (the "License");
5 # you may not use this file except in compliance with the License.
6 # You may obtain a copy of the License at
7 #
8 #     http://www.apache.org/licenses/LICENSE-2.0
9 #
10 # Unless required by applicable law or agreed to in writing, software
11 # distributed under the License is distributed on an "AS IS" BASIS,
12 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
13 # implied.
14 # See the License for the specific language governing permissions and
15 # limitations under the License.
16 """
17 """
18 This module provides a set of REST API for switch configuration.
19 - Per-switch Key-Value store
20
21 Used by OpenStack Ryu agent.
22 """
23
24 import json
25
26 from six.moves import http_client
27
28 from ryu.app.wsgi import ControllerBase
29 from ryu.app.wsgi import Response

```

```

30| from ryu.base import app_manager
31| from ryu.controller import conf_switch
32| from ryu.lib import dpid as dpid_lib
33|
34|
35| # REST API for switch configuration
36| #
37| # get all the switches
38| # GET /v1.0/conf/switches
39| #
40| # get all the configuration keys of a switch
41| # GET /v1.0/conf/switches/<dpid>
42| #
43| # delete all the configuration of a switch
44| # DELETE /v1.0/conf/switches/<dpid>
45| #
46| # set the <key> configuration of a switch
47| # PUT /v1.0/conf/switches/<dpid>/<key>
48| #
49| # get the <key> configuration of a switch
50| # GET /v1.0/conf/switches/<dpid>/<key>
51| #
52| # delete the <key> configuration of a switch
53| # DELETE /v1.0/conf/switches/<dpid>/<key>
54| #
55| # where
56| # <dpid>: datapath id in 16 hex
57|
58|
59| class ConfSwitchController(ControllerBase):
60|     def __init__(self, req, link, data, **config):
61|         super(ConfSwitchController, self).__init__(req, link, data, **config)
62|         self.conf_switch = data
63|
64|     def list_switches(self, _req, **_kwargs):
65|         dpids = self.conf_switch.dpids()
66|         body = json.dumps([dpid_lib.dpid_to_str(dpid) for dpid in dpids])
67|         return Response(content_type='application/json', body=body)
68|
69|     @staticmethod
70|     def _do_switch(dpid, func, ret_func):
71|         dpid = dpid_lib.str_to_dpid(dpid)
72|         try:
73|             ret = func(dpid)
74|         except KeyError:
75|             return Response(status=http_client.NOT_FOUND,
76|                             body='no dpid is found %s' %
77|                                   dpid_lib.dpid_to_str(dpid))
78|
79|         return ret_func(ret)
80|
81|     def delete_switch(self, _req, dpid, **_kwargs):

```

```

82     def _delete_switch(dpid):
83         self.conf_switch.del_dpid(dpid)
84         return None
85
86     def _ret(_ret):
87         return Response(status=http_client.ACCEPTED)
88
89     return self._do_switch(dpid, _delete_switch, _ret)
90
91 def list_keys(self, _req, dpid, **_kwargs):
92     def _list_keys(dpid):
93         return self.conf_switch.keys(dpid)
94
95     def _ret(keys):
96         body = json.dumps(keys)
97         return Response(content_type='application/json', body=body)
98
99     return self._do_switch(dpid, _list_keys, _ret)
100
101 @staticmethod
102 def _do_key(dpid, key, func, ret_func):
103     dpid = dpid_lib.str_to_dpid(dpid)
104     try:
105         ret = func(dpid, key)
106     except KeyError:
107         return Response(status=http_client.NOT_FOUND,
108                         body='no dpid/key is found %s %s' %
109                         (dpid_lib.dpid_to_str(dpid), key))
110     return ret_func(ret)
111
112 def set_key(self, req, dpid, key, **_kwargs):
113     def _set_val(dpid, key):
114         try:
115             val = req.json if req.body else {}
116         except ValueError:
117             return Response(status=http_client.BAD_REQUEST,
118                             body='invalid syntax %s' % req.body)
119         self.conf_switch.set_key(dpid, key, val)
120         return None
121
122     def _ret(_ret):
123         return Response(status=http_client.CREATED)
124
125     return self._do_key(dpid, key, _set_val, _ret)
126
127 def get_key(self, _req, dpid, key, **_kwargs):
128     def _get_key(dpid, key):
129         return self.conf_switch.get_key(dpid, key)
130
131     def _ret(val):
132         return Response(content_type='application/json',
133                         body=json.dumps(val))
134
135     return self._do_key(dpid, key, _get_key, _ret)

```

```

136
137     def delete_key(self, _req, dpid, key, **_kwargs):
138         def _delete_key(dpid, key):
139             self.conf_switch.del_key(dpid, key)
140             return None
141
142         def _ret(_ret):
143             return Response()
144
145         return self._do_key(dpid, key, _delete_key, _ret)
146
147
148 class ConfSwitchAPI(app_manager.RyuApp):
149     CONTEXTS = {
150         'conf_switch': conf_switch.ConfSwitchSet,
151     }
152
153     def __init__(self, *args, **kwargs):
154         super(ConfSwitchAPI, self).__init__(*args, **kwargs)
155         self.conf_switch = kwargs['conf_switch']
156         wsgi = kwargs['wsgi']
157         mapper = wsgi.mapper
158
159         controller = ConfSwitchController
160         wsgi.registry[controller.__name__] = self.conf_switch
161         route_name = 'conf_switch'
162         uri = '/v1.0/conf/switches'
163         mapper.connect(route_name, uri, controller=controller,
164                         action='list_switches',
165                         conditions=dict(method=['GET']))
166
167         uri += '/{dpid}'
168         requirements = {'dpid': dpid_lib.DPID_PATTERN}
169         s = mapper.submapper(controller=controller, requirements=
requirements)
170         s.connect(route_name, uri, action='delete_switch',
171                   conditions=dict(method=['DELETE']))
172         s.connect(route_name, uri, action='list_keys',
173                   conditions=dict(method=['GET']))
174
175         uri += '/{key}'
176         s.connect(route_name, uri, action='set_key',
177                   conditions=dict(method=['PUT']))
178         s.connect(route_name, uri, action='get_key',
179                   conditions=dict(method=['GET']))
180         s.connect(route_name, uri, action='delete_key',
181                   conditions=dict(method=['DELETE']))

```

A.7 rest_qos.py

```
| # Copyright (C) 2014 Kiyonari Harigae <lakshmi at cloudfsunny14.org>
```

```

2  #
3  # Licensed under the Apache License , Version 2.0 (the "License");
4  # you may not use this file except in compliance with the License.
5  # You may obtain a copy of the License at
6  #
7  # http://www.apache.org/licenses/LICENSE-2.0
8  #
9  # Unless required by applicable law or agreed to in writing , software
10 # distributed under the License is distributed on an "AS IS" BASIS,
11 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
12 # implied.
13 # See the License for the specific language governing permissions and
14 # limitations under the License.

15
16
17 import logging
18 import json
19 import re
20
21 from ryu.app import conf_switch_key as cs_key
22 from ryu.app.wsgi import ControllerBase
23 from ryu.app.wsgi import Response
24 from ryu.app.wsgi import route
25 from ryu.app.wsgi import WSGIApplication
26 from ryu.base import app_manager
27 from ryu.controller import conf_switch
28 from ryu.controller import ofp_event
29 from ryu.controller import dpset
30 from ryu.controller.handler import set_ev_cls
31 from ryu.controller.handler import MAIN_DISPATCHER
32 from ryu.exception import OFPUnknownVersion
33 from ryu.lib import dpid as dpid_lib
34 from ryu.lib import mac
35 from ryu.lib import ofctl_v1_0
36 from ryu.lib import ofctl_v1_2
37 from ryu.lib import ofctl_v1_3
38 from ryu.lib.ovs import bridge
39 from ryu.ofproto import ofproto_v1_0
40 from ryu.ofproto import ofproto_v1_2
41 from ryu.ofproto import ofproto_v1_3
42 from ryu.ofproto import ofproto_v1_3_parser
43 from ryu.ofproto import ether
44 from ryu.ofproto import inet

45
46
47 # =====
48 #          REST API
49 # =====
50 #
51 # Note: specify switch and vlan group , as follows .
52 # {switch-id} : 'all' or switchID
53 # {vlan-id}   : 'all' or vlanID
54 #
55 # about queue status

```

```

56 #
57 # get status of queue
58 # GET /qos/queue/status/{ switch-id }
59 #
60 # about queues
61 # get a queue configurations
62 # GET /qos/queue/{ switch-id }
63 #
64 # set a queue to the switches
65 # POST /qos/queue/{ switch-id }
66 #
67 # request body format:
68 # {"port_name": "<name of port>",
69 #   "type": "<linux-htb or linux-other>",
70 #   "max-rate": "<int>",
71 #   "queues": [{"max_rate": "<int>", "min_rate": "<int>"},...]}
72 #
73 # Note: This operation override
74 # previous configurations.
75 # Note: Queue configurations are available for
76 # OpenvSwitch.
77 # Note: port_name is optional argument.
78 # If does not pass the port_name argument,
79 # all ports are target for configuration.
80 #
81 # delete queue
82 # DELETE /qos/queue/{ switch-id }
83 #
84 # Note: This operation delete relation of qos record from
85 # qos colum in Port table . Therefore ,
86 # QoS records and Queue records will remain .
87 #
88 # about qos rules
89 #
90 # get rules of qos
91 # * for no vlan
92 # GET /qos/rules/{ switch-id }
93 #
94 # * for specific vlan group
95 # GET /qos/rules/{ switch-id }/{ vlan-id }
96 #
97 # set a qos rules
98 #
99 # QoS rules will do the processing pipeline ,
100 # which entries are register the first table (by default table id 0)
101 # and process will apply and go to next table .
102 #
103 # * for no vlan
104 # POST /qos/{ switch-id }
105 #
106 # * for specific vlan group
107 # POST /qos/{ switch-id }/{ vlan-id }
108 #
109 # request body format:

```

```

110 # {"priority": "<value>",
111 #   "match": {"<field1>": "<value1>", "<field2>": "<value2>"},...},
112 #   "actions": {"<action1>": "<value1>", "<action2>": "<value2>"},...}
113 #
114 #
115 # Description
116 #   * priority field
117 #     <value>
118 #     "0 to 65533"
119 #
120 # Note: When "priority" has not been set up,
121 #       "priority: 1" is set to "priority".
122 #
123 #   * match field
124 #     <field> : <value>
125 #     "in_port" : "<int>"
126 #     "dl_src" : "<xx:xx:xx:xx:xx:xx>"
127 #     "dl_dst" : "<xx:xx:xx:xx:xx:xx>"
128 #     "dl_type" : "<ARP or IPv4 or IPv6>"
129 #     "nw_src" : "<A.B.C.D/M>"
130 #     "nw_dst" : "<A.B.C.D/M>"
131 #     "ipv6_src": "<xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx/M>"
132 #     "ipv6_dst": "<xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx/M>"
133 #     "nw_proto": "<TCP or UDP or ICMP or ICMPv6>"
134 #     "tp_src" : "<int>"
135 #     "tp_dst" : "<int>"
136 #     "ip_dscp" : "<int>"
137 #
138 #   * actions field
139 #     <field> : <value>
140 #     "mark": <dscp-value>
141 #       sets the IPv4 ToS/DSCP field to tos.
142 #     "meter": <meter-id>
143 #       apply meter entry
144 #     "queue": <queue-id>
145 #       register queue specified by queue-id
146 #
147 # Note: When "actions" has not been set up,
148 #       "queue: 0" is set to "actions".
149 #
150 # delete a qos rules
151 # * for no vlan
152 # DELETE /qos/rule/{switch-id}
153 #
154 # * for specific vlan group
155 # DELETE /qos/{switch-id}/{vlan-id}
156 #
157 # request body format:
158 #   {"<field>": "<value>"}
159 #
160 #     <field> : <value>
161 #     "qos_id" : "<int>" or "all"
162 #
163 # about meter entries

```

```

164 #
165 # set a meter entry
166 # POST /qos/meter/{switch-id}
167 #
168 # request body format:
169 #   {"meter_id": <int>,
170 #    "bands": [{"action": "<DROP or DSCP.REMARK>",
171 #               "flag": "<KBPS or PKTPS or BURST or STATS",
172 #               "burst_size": <int>,
173 #               "rate": <int>,
174 #               "prec_level": <int>} ,...]}}
175 #
176 # delete a meter entry
177 # DELETE /qos/meter/{switch-id}
178 #
179 # request body format:
180 #   {"<field>":<value>"}
181 #
182 #   <field> : <value>
183 #   "meter_id" : "<int>"
184 #
185
186
187 SWITCHID_PATTERN = dpid_lib.DPID.PATTERN + r'|all'
188 VLANID_PATTERN = r'[0-9]{1,4}|all'
189
190 QOS_TABLE_ID = 0
191
192 REST_ALL = 'all'
193 REST_SWITCHID = 'switch_id'
194 REST_COMMAND_RESULT = 'command_result'
195 REST_PRIORITY = 'priority'
196 REST_VLANID = 'vlan_id'
197 REST_PORT_NAME = 'port_name'
198 REST_QUEUE_TYPE = 'type'
199 REST_QUEUE_MAX_RATE = 'max_rate'
200 REST_QUEUE_MIN_RATE = 'min_rate'
201 REST_QUEUES = 'queues'
202 REST_QOS = 'qos'
203 REST_QOS_ID = 'qos_id'
204 REST_COOKIE = 'cookie'
205
206 REST_MATCH = 'match'
207 REST_IN_PORT = 'in_port'
208 REST_SRC_MAC = 'dl_src'
209 REST_DST_MAC = 'dl_dst'
210 REST_DL_TYPE = 'dl_type'
211 REST_DL_TYPE_ARP = 'ARP'
212 REST_DL_TYPE_IPV4 = 'IPv4'
213 REST_DL_TYPE_IPV6 = 'IPv6'
214 REST_DL_VLAN = 'dl_vlan'
215 REST_SRC_IP = 'nw_src'
216 REST_DST_IP = 'nw_dst'
217 REST_SRC_IPV6 = 'ipv6_src'

```

```

218 REST_DST_IPV6 = 'ipv6_dst'
219 REST_NW_PROTO = 'nw_proto'
220 REST_NW_PROTO_TCP = 'TCP'
221 REST_NW_PROTO_UDP = 'UDP'
222 REST_NW_PROTO_ICMP = 'ICMP'
223 REST_NW_PROTO_ICMPV6 = 'ICMPv6'
224 REST_TP_SRC = 'tp_src'
225 REST_TP_DST = 'tp_dst'
226 REST_DSCP = 'ip_dscp'

227
228 REST_ACTION = 'actions'
229 REST_ACTION_QUEUE = 'queue'
230 REST_ACTION_MARK = 'mark'
231 REST_ACTION_METER = 'meter'

232
233 REST_METER_ID = 'meter_id'
234 REST_METER_BURST_SIZE = 'burst_size'
235 REST_METER_RATE = 'rate'
236 REST_METER_PREC_LEVEL = 'prec_level'
237 REST_METER_BANDS = 'bands'
238 REST_METER_ACTION_DROP = 'drop'
239 REST_METER_ACTION_REMARK = 'remark'

240
241 DEFAULT_FLOW_PRIORITY = 0
242 QOS_PRIORITY_MAX = ofproto_v1_3_parser.UINT16_MAX - 1
243 QOS_PRIORITY_MIN = 1

244
245 VLANID_NONE = 0
246 VLANID_MIN = 2
247 VLANID_MAX = 4094
248 COOKIE_SHIFT_VLANID = 32

249
250 BASE_URL = '/qos'
251 REQUIREMENTS = {'switchid': SWITCHID_PATTERN,
252                  'vlanid': VLANID_PATTERN}
253
254 LOG = logging.getLogger(__name__)
255
256
257 class RestQoSAPI(app_manager.RyuApp):
258
259     OFP VERSIONS = [ofproto_v1_0.OFP_VERSION,
260                     ofproto_v1_2.OFP_VERSION,
261                     ofproto_v1_3.OFP_VERSION]
262
263     _CONTEXTS = {
264         'dpset': dpset.DPSet,
265         'conf_switch': conf_switch.ConfSwitchSet,
266         'wsgi': WSGIApplication}
267
268     def __init__(self, *args, **kwargs):
269         super(RestQoSAPI, self).__init__(*args, **kwargs)
270
271         # logger configure

```

```

272     QoSController.set_logger(self.logger)
273     self.cs = kwargs['conf_switch']
274     self.dpset = kwargs['dpset']
275     wsgi = kwargs['wsgi']
276     self.waiters = {}
277     self.data = {}
278     self.data['dpset'] = self.dpset
279     self.data['waiters'] = self.waiters
280     wsgi.registry['QoSController'] = self.data
281     wsgi.register(QoSController, self.data)
282
283     def stats_reply_handler(self, ev):
284         msg = ev.msg
285         dp = msg.datapath
286
287         if dp.id not in self.waiters:
288             return
289         if msg.xid not in self.waiters[dp.id]:
290             return
291         lock, msgs = self.waiters[dp.id][msg.xid]
292         msgs.append(msg)
293
294         flags = 0
295         if dp.ofproto.OFP_VERSION == ofproto_v1_0.OFP_VERSION or \
296             dp.ofproto.OFP_VERSION == ofproto_v1_2.OFP_VERSION:
297             flags = dp.ofproto.OFPSF_REPLY_MORE
298         elif dp.ofproto.OFP_VERSION == ofproto_v1_3.OFP_VERSION:
299             flags = dp.ofproto.OFPMMPF_REPLY_MORE
300
301         if msg.flags & flags:
302             return
303         del self.waiters[dp.id][msg.xid]
304         lock.set()
305
306     @set_ev_cls(conf_switch.EventConfSwitchSet)
307     def conf_switch_set_handler(self, ev):
308         if ev.key == cs_key.OVSDB_ADDR:
309             QoSController.set_ovsdb_addr(ev.dpid, ev.value)
310         else:
311             QoSController.LOGGER.debug("unknown event: %s", ev)
312
313     @set_ev_cls(conf_switch.EventConfSwitchDel)
314     def conf_switch_del_handler(self, ev):
315         if ev.key == cs_key.OVSDB_ADDR:
316             QoSController.delete_ovsdb_addr(ev.dpid)
317         else:
318             QoSController.LOGGER.debug("unknown event: %s", ev)
319
320     @set_ev_cls(dpset.EventDP, dpset.DPSET_EV_DISPATCHER)
321     def handler_datapath(self, ev):
322         if ev.enter:
323             QoSController.regist_ofs(ev.dp, self.CONF)
324         else:
325             QoSController.unregist_ofs(ev.dp)

```

```

326
327 # for OpenFlow version1.0
328 @set_ev_cls(ofp_event.EventOFPFlowStatsReply, MAIN_DISPATCHER)
329 def stats_reply_handler_v1_0(self, ev):
330     self.stats_reply_handler(ev)
331
332 # for OpenFlow version1.2 or later
333 @set_ev_cls(ofp_event.EventOFPStatsReply, MAIN_DISPATCHER)
334 def stats_reply_handler_v1_2(self, ev):
335     self.stats_reply_handler(ev)
336
337 # for OpenFlow version1.2 or later
338 @set_ev_cls(ofp_event.EventOFPQueueStatsReply, MAIN_DISPATCHER)
339 def queue_stats_reply_handler_v1_2(self, ev):
340     self.stats_reply_handler(ev)
341
342 # for OpenFlow version1.2 or later
343 @set_ev_cls(ofp_event.EventOFPMeterStatsReply, MAIN_DISPATCHER)
344 def meter_stats_reply_handler_v1_2(self, ev):
345     self.stats_reply_handler(ev)
346
347
348 class QoSOfsList(dict):
349
350     def __init__(self):
351         super(QoSOfsList, self).__init__()
352
353     def get_ofs(self, dp_id):
354         if len(self) == 0:
355             raise ValueError('qos sw is not connected.')
356
357         dps = {}
358         if dp_id == REST_ALL:
359             dps = self
360         else:
361             try:
362                 dpid = dpid_lib.str_to_dpid(dp_id)
363             except:
364                 raise ValueError('Invalid switchID.')
365
366             if dpid in self:
367                 dps = {dpid: self[dpid]}
368             else:
369                 msg = 'qos sw is not connected. : switchID=%s' % dp_id
370                 raise ValueError(msg)
371
372         return dps
373
374
375 class QoSController(ControllerBase):
376
377     _OFS_LIST = QoSOfsList()
378     _LOGGER = None
379

```

```

380     def __init__(self, req, link, data, **config):
381         super(QoSController, self).__init__(req, link, data, **config)
382         self.dpset = data['dpset']
383         self.waiters = data['waiters']
384
385     @classmethod
386     def set_logger(cls, logger):
387         cls.LOGGER = logger
388         cls.LOGGER.propagate = False
389         hdlr = logging.StreamHandler()
390         fmt_str = '[QoS][%(levelname)s] %(message)s'
391         hdlr.setFormatter(logging.Formatter(fmt_str))
392         cls.LOGGER.addHandler(hdlr)
393
394     @staticmethod
395     def regist_ofs(dp, CONF):
396         if dp.id in QoSController._OFS_LIST:
397             return
398
399         dpid_str = dpid_lib.dpid_to_str(dp.id)
400         try:
401             f_ofs = QoS(dp, CONF)
402             f_ofs.set_default_flow()
403         except OFPUnknownVersion as message:
404             QoSController.LOGGER.info('dpid=%s: %s',
405                                       dpid_str, message)
406         return
407
408         QoSController._OFS_LIST.setdefault(dp.id, f_ofs)
409         QoSController.LOGGER.info('dpid=%s: Join qos switch.', 
410                                   dpid_str)
411
412     @staticmethod
413     def unregist_ofs(dp):
414         if dp.id in QoSController._OFS_LIST:
415             del QoSController._OFS_LIST[dp.id]
416             QoSController.LOGGER.info('dpid=%s: Leave qos switch.', 
417                                       dpid_lib.dpid_to_str(dp.id))
418
419     @staticmethod
420     def set_ovsdb_addr(dpid, value):
421         ofs = QoSController._OFS_LIST.get(dpid, None)
422         if ofs is not None:
423             ofs.set_ovsdb_addr(dpid, value)
424
425     @staticmethod
426     def delete_ovsdb_addr(dpid):
427         ofs = QoSController._OFS_LIST.get(dpid, None)
428         ofs.set_ovsdb_addr(dpid, None)
429
430     @route('qos_switch', BASE_URL + '/queue/{switchid}', 
431           methods=['GET'], requirements=REQUIREMENTS)
432     def get_queue(self, req, switchid, **kwargs):
433         return self._access_switch(req, switchid, VLANID_NONE,

```

```

434             'get_queue', None)
435
436     @route('qos_switch', BASE_URL + '/queue/{switchid}',
437            methods=[POST], requirements=REQUIREMENTS)
438     def set_queue(self, req, switchid, **kwargs):
439         return self._access_switch(req, switchid, VLANID_NONE,
440                                     'set_queue', None)
441
442     @route('qos_switch', BASE_URL + '/queue/{switchid}',
443            methods=[DELETE], requirements=REQUIREMENTS)
444     def delete_queue(self, req, switchid, **kwargs):
445         return self._access_switch(req, switchid, VLANID_NONE,
446                                   'delete_queue', None)
447
448     @route('qos_switch', BASE_URL + '/queue/status/{switchid}',
449            methods=[GET], requirements=REQUIREMENTS)
450     def get_status(self, req, switchid, **kwargs):
451         return self._access_switch(req, switchid, VLANID_NONE,
452                                   'get_status', self.waiters)
453
454     @route('qos_switch', BASE_URL + '/rules/{switchid}',
455            methods=[GET], requirements=REQUIREMENTS)
456     def get_qos(self, req, switchid, **kwargs):
457         return self._access_switch(req, switchid, VLANID_NONE,
458                                   'get_qos', self.waiters)
459
460     @route('qos_switch', BASE_URL + '/rules/{switchid}/{vlanid}',
461            methods=[GET], requirements=REQUIREMENTS)
462     def get_vlan_qos(self, req, switchid, vlanid, **kwargs):
463         return self._access_switch(req, switchid, vlanid,
464                                   'get_qos', self.waiters)
465
466     @route('qos_switch', BASE_URL + '/rules/{switchid}',
467            methods=[POST], requirements=REQUIREMENTS)
468     def set_qos(self, req, switchid, **kwargs):
469         return self._access_switch(req, switchid, VLANID_NONE,
470                                   'set_qos', self.waiters)
471
472     @route('qos_switch', BASE_URL + '/rules/{switchid}/{vlanid}',
473            methods=[POST], requirements=REQUIREMENTS)
474     def set_vlan_qos(self, req, switchid, vlanid, **kwargs):
475         return self._access_switch(req, switchid, vlanid,
476                                   'set_qos', self.waiters)
476
477     @route('qos_switch', BASE_URL + '/rules/{switchid}',
478            methods=[DELETE], requirements=REQUIREMENTS)
479     def delete_qos(self, req, switchid, **kwargs):
480         return self._access_switch(req, switchid, VLANID_NONE,
481                                   'delete_qos', self.waiters)
482
483     @route('qos_switch', BASE_URL + '/rules/{switchid}/{vlanid}',
484            methods=[DELETE], requirements=REQUIREMENTS)
485     def delete_vlan_qos(self, req, switchid, vlanid, **kwargs):
486         return self._access_switch(req, switchid, vlanid,

```

```

488                               'delete_qos', self.waiters))
489
490 @route('qos_switch', BASE_URL + '/meter/{switchid}',
491         methods=['GET'], requirements=REQUIREMENTS)
492 def get_meter(self, req, switchid, **kwargs):
493     return self._access_switch(req, switchid, VLANID_NONE,
494                               'get_meter', self.waiters)
495
496 @route('qos_switch', BASE_URL + '/meter/{switchid}',
497         methods=['POST'], requirements=REQUIREMENTS)
498 def set_meter(self, req, switchid, **kwargs):
499     return self._access_switch(req, switchid, VLANID_NONE,
500                               'set_meter', self.waiters)
501
502 @route('qos_switch', BASE_URL + '/meter/{switchid}',
503         methods=['DELETE'], requirements=REQUIREMENTS)
504 def delete_meter(self, req, switchid, **kwargs):
505     return self._access_switch(req, switchid, VLANID_NONE,
506                               'delete_meter', self.waiters)
507
508 def _access_switch(self, req, switchid, vlan_id, func, waiters):
509     try:
510         rest = req.json if req.body else {}
511     except ValueError:
512         QoSController.LOGGER.debug('invalid syntax %s', req.body)
513         return Response(status=400)
514
515     try:
516         dps = self._OFS_LIST.get_ofs(switchid)
517         vid = QoSController._conv_toint_vlanid(vlan_id)
518     except ValueError as message:
519         return Response(status=400, body=str(message))
520
521     msgs = []
522     for f_ofs in dps.values():
523         function = getattr(f_ofs, func)
524         try:
525             if waiters is not None:
526                 msg = function(rest, vid, waiters)
527             else:
528                 msg = function(rest, vid)
529         except ValueError as message:
530             return Response(status=400, body=str(message))
531         msgs.append(msg)
532
533     body = json.dumps(msgs)
534     return Response(content_type='application/json', body=body)
535
536 @staticmethod
537 def _conv_toint_vlanid(vlan_id):
538     if vlan_id != REST_ALL:
539         vlan_id = int(vlan_id)
540         if (vlan_id != VLANID_NONE and
541             (vlan_id < VLANID_MIN or VLANID_MAX < vlan_id)):

```

```

542             msg = 'Invalid {vlan_id} value. Set [%d-%d]' % (
543                 VLANID_MIN,
544                 VLANID_MAX)
545             raise ValueError(msg)
546         return vlan_id
547
548     class QoS(object):
549
550         _OFCTL = {ofproto_v1_0.OFP_VERSION: ofctl_v1_0,
551                   ofproto_v1_2.OFP_VERSION: ofctl_v1_2,
552                   ofproto_v1_3.OFP_VERSION: ofctl_v1_3}
553
554     def __init__(self, dp, CONF):
555         super(QoS, self).__init__()
556         self.vlan_list = {}
557         self.vlan_list[VLANID_NONE] = 0 # for VLAN=None
558         self.dp = dp
559         self.version = dp.ofproto.OFP_VERSION
560         self.queue_list = {}
561         self.CONF = CONF
562         self.ovsdb_addr = None
563         self.ovs_bridge = None
564
565         if self.version not in self._OFCTL:
566             raise OFPUnknownVersion(version=self.version)
567
568         self.ofctl = self._OFCTL[self.version]
569
570     def set_default_flow(self):
571         if self.version == ofproto_v1_0.OFP_VERSION:
572             return
573
574         cookie = 0
575         priority = DEFAULT_FLOW_PRIORITY
576         actions = [{ 'type': 'GOTO_TABLE',
577                     'table_id': QOS_TABLE_ID + 1}]
578         flow = self._to_of_flow(cookie=cookie,
579                                priority=priority,
580                                match={},
581                                actions=actions)
582
583         cmd = self.dp.ofproto.OFPFC_ADD
584         self.ofctl.mod_flow_entry(self.dp, flow, cmd)
585
586     def set_ovsdb_addr(self, dpid, ovsdb_addr):
587         # easy check if the address format valid
588         _proto, _host, _port = ovsdb_addr.split(':')
589
590         old_address = self.ovsdb_addr
591         if old_address == ovsdb_addr:
592             return
593         if ovsdb_addr is None:

```

```

594         if self.ovs_bridge:
595             self.ovs_bridge.del_controller()
596             self.ovs_bridge = None
597     return
598     self.ovsdb_addr = ovsdb_addr
599     if self.ovs_bridge is None:
600         ovs_bridge = bridge.OVSBridge(self.CONF, dpid, ovsdb_addr)
601         self.ovs_bridge = ovs_bridge
602     try:
603         ovs_bridge.init()
604     except:
605         raise ValueError('ovsdb addr is not available.')
606
607     def _update_vlan_list(self, vlan_list):
608         for vlan_id in self.vlan_list.keys():
609             if vlan_id is not VLANID_NONE and vlan_id not in vlan_list:
610                 del self.vlan_list[vlan_id]
611
612     def _get_cookie(self, vlan_id):
613         if vlan_id == REST_ALL:
614             vlan_ids = self.vlan_list.keys()
615         else:
616             vlan_ids = [vlan_id]
617
618         cookie_list = []
619         for vlan_id in vlan_ids:
620             self.vlan_list.setdefault(vlan_id, 0)
621             self.vlan_list[vlan_id] += 1
622             self.vlan_list[vlan_id] &= ofproto_v1_3_parser.UINT32_MAX
623             cookie = (vlan_id << COOKIE_SHIFT_VLANID) + \
624                     self.vlan_list[vlan_id]
625             cookie_list.append([cookie, vlan_id])
626
627     return cookie_list
628
629     @staticmethod
630     def _cookie_to_qosid(cookie):
631         return cookie & ofproto_v1_3_parser.UINT32_MAX
632
633     # REST command template
634     def rest_command(func):
635         def _rest_command(*args, **kwargs):
636             key, value = func(*args, **kwargs)
637             switch_id = dpid_lib.dpid_to_str(args[0].dp.id)
638             return {REST_SWITCHID: switch_id,
639                     key: value}
640         return _rest_command
641
642     @rest_command
643     def get_status(self, req, vlan_id, waiters):
644         if self.version == ofproto_v1_0.OFP_VERSION:
645             raise ValueError('get_status operation is not supported')
646
647     msgs = self.ofctl.get_queue_stats(self.dp, waiters)

```

```

648     return REST_COMMAND_RESULT, msgs
649
650     @rest_command
651     def get_queue(self, rest, vlan_id):
652         if len(self.queue_list):
653             msg = {'result': 'success',
654                   'details': self.queue_list}
655         else:
656             msg = {'result': 'failure',
657                   'details': 'Queue is not exists.'}
658
659     return REST_COMMAND_RESULT, msg
660
661     @rest_command
662     def set_queue(self, rest, vlan_id):
663         if self.ovs_bridge is None:
664             msg = {'result': 'failure',
665                   'details': 'ovs_bridge is not exists'}
666         return REST_COMMAND_RESULT, msg
667
668         self.queue_list.clear()
669         queue_type = rest.get(REST_QUEUE_TYPE, 'linux-htb')
670         parent_max_rate = rest.get(REST_QUEUE_MAX_RATE, None)
671         queues = rest.get(REST_QUEUES, [])
672         queue_id = 0
673         queue_config = []
674         for queue in queues:
675             max_rate = queue.get(REST_QUEUE_MAX_RATE, None)
676             min_rate = queue.get(REST_QUEUE_MIN_RATE, None)
677             if max_rate is None and min_rate is None:
678                 raise ValueError('Required to specify max_rate or
min_rate')
679             config = {}
680             if max_rate is not None:
681                 config['max-rate'] = max_rate
682             if min_rate is not None:
683                 config['min-rate'] = min_rate
684             if len(config):
685                 queue_config.append(config)
686             self.queue_list[queue_id] = {'config': config}
687             queue_id += 1
688
689         port_name = rest.get(REST_PORT_NAME, None)
690         vif_ports = self.ovs_bridge.get_port_name_list()
691
692         if port_name is not None:
693             if port_name not in vif_ports:
694                 raise ValueError('%s port is not exists' % port_name)
695             vif_ports = [port_name]
696
697         for port_name in vif_ports:
698             try:
699                 self.ovs_bridge.set_qos(port_name, type=queue_type,
700                                         max_rate=parent_max_rate,

```

```

701                                     queues=queue_config)
702     except Exception as msg:
703         raise ValueError(msg)
704
705     msg = {'result': 'success',
706            'details': self.queue_list}
707
708     return REST_COMMAND_RESULT, msg
709
710 def _delete_queue(self):
711     if self.ovs_bridge is None:
712         return False
713
714     vif_ports = self.ovs_bridge.get_external_ports()
715     for port in vif_ports:
716         self.ovs_bridge.del_qos(port.port_name)
717     return True
718
719 @rest_command
720 def delete_queue(self, rest, vlan_id):
721     self.queue_list.clear()
722     if self._delete_queue():
723         msg = 'success'
724     else:
725         msg = 'failure'
726
727     return REST_COMMAND_RESULT, msg
728
729 @rest_command
730 def set_qos(self, rest, vlan_id, waiters):
731     msgs = []
732     cookie_list = self._get_cookie(vlan_id)
733     for cookie, vid in cookie_list:
734         msg = self._set_qos(cookie, rest, waiters, vid)
735         msgs.append(msg)
736     return REST_COMMAND_RESULT, msgs
737
738 def _set_qos(self, cookie, rest, waiters, vlan_id):
739     match_value = rest[REST_MATCH]
740     if vlan_id:
741         match_value[REST_DL_VLAN] = vlan_id
742
743     priority = int(rest.get(REST_PRIORITY, QOS_PRIORITY_MIN))
744     if (QOS_PRIORITY_MAX < priority):
745         raise ValueError('Invalid priority value. Set [%d-%d]',
746                          % (QOS_PRIORITY_MIN, QOS_PRIORITY_MAX))
747
748     match = Match.to_openflow(match_value)
749
750     actions = []
751     action = rest.get(REST_ACTION, None)
752     if action is not None:
753         if REST_ACTION_MARK in action:
754             actions.append({'type': 'SET_FIELD',

```

```

755             'field': REST_DSCP,
756             'value': int(action[REST_ACTION_MARK]))})
757     if REST_ACTION_METER in action:
758         actions.append({'type': 'METER',
759                         'meter_id': action[REST_ACTION_METER]}))
760     if REST_ACTION_QUEUE in action:
761         actions.append({'type': 'SET_QUEUE',
762                         'queue_id': action[REST_ACTION_QUEUE]}))
763     else:
764         actions.append({'type': 'SET_QUEUE',
765                         'queue_id': 0})
766
767     actions.append({'type': 'GOTO_TABLE',
768                     'table_id': QOS_TABLE_ID + 1})
769     flow = self._to_of_flow(cookie=cookie, priority=priority,
770                             match=match, actions=actions)
771
772     cmd = self.dp.ofproto.OFPFC_ADD
773     try:
774         self.ofctl.mod_flow_entry(self.dp, flow, cmd)
775     except:
776         raise ValueError('Invalid rule parameter.')
777
778     qos_id = QoS._cookie_to_qosid(cookie)
779     msg = {'result': 'success',
780            'details': 'QoS added. : qos_id=%d' % qos_id}
781
782     if vlan_id != VLANID_NONE:
783         msg.setdefault(REST_VLANID, vlan_id)
784     return msg
785
786 @rest_command
787 def get_qos(self, rest, vlan_id, waiters):
788     rules = {}
789     msgs = self.ofctl.get_flow_stats(self.dp, waiters)
790     if str(self.dp.id) in msgs:
791         flow_stats = msgs[str(self.dp.id)]
792         for flow_stat in flow_stats:
793             if flow_stat['table_id'] != QOS_TABLE_ID:
794                 continue
795             priority = flow_stat[REST_PRIORITY]
796             if priority != DEFAULT_FLOW_PRIORITY:
797                 vid = flow_stat[REST_MATCH].get(REST_DL_VLAN,
VLANID_NONE)
798                 if vlan_id == REST_ALL or vlan_id == vid:
799                     rule = self._to_rest_rule(flow_stat)
800                     rules.setdefault(vid, [])
801                     rules[vid].append(rule)
802
803     get_data = []
804     for vid, rule in rules.items():
805         if vid == VLANID_NONE:
806             vid_data = {REST_QOS: rule}
807         else:

```

```

808         vid_data = {REST_VLANID: vid, REST_QOS: rule}
809         get_data.append(vid_data)
810
811     return REST_COMMAND_RESULT, get_data
812
813 @rest_command
814 def delete_qos(self, rest, vlan_id, waiters):
815     try:
816         if rest[REST_QOS_ID] == REST_ALL:
817             qos_id = REST_ALL
818         else:
819             qos_id = int(rest[REST_QOS_ID])
820     except:
821         raise ValueError('Invalid qos id.')
822
823     vlan_list = []
824     delete_list = []
825
826     msgs = self.ofctl.get_flow_stats(self.dp, waiters)
827     if str(self.dp.id) in msgs:
828         flow_stats = msgs[str(self.dp.id)]
829         for flow_stat in flow_stats:
830             cookie = flow_stat[REST_COOKIE]
831             ruleid = QoS._cookie_to_qosid(cookie)
832             priority = flow_stat[REST_PRIORITY]
833             dl_vlan = flow_stat[REST_DL_VLAN,
VLANID_NONE)
834
835             if priority != DEFAULT_FLOW_PRIORITY:
836                 if ((qos_id == REST_ALL or qos_id == ruleid) and
(vlan_id == dl_vlan or vlan_id == REST_ALL))
837                 :
838                     match = Match.to_mod_openflow(flow_stat[
REST_MATCH])
839                     delete_list.append([cookie, priority, match])
840                 else:
841                     if dl_vlan not in vlan_list:
842                         vlan_list.append(dl_vlan)
843
844     self._update_vlan_list(vlan_list)
845
846     if len(delete_list) == 0:
847         msg_details = 'QoS rule is not exist.'
848         if qos_id != REST_ALL:
849             msg_details += ': QoS ID=%d' % qos_id
850         msg = {'result': 'failure',
851               'details': msg_details}
852     else:
853         cmd = self.dp.ofproto.OFPFC_DELETE_STRICT
854         actions = []
855         delete_ids = {}
856         for cookie, priority, match in delete_list:
857             flow = self._to_of_flow(cookie=cookie, priority=priority
,

```

```

858                         match=match, actions=actions)
859             self.ofctl.mod_flow_entry(self.dp, flow, cmd)
860
861             vid = match.get(REST_DL_VLAN, VLANID_NONE)
862             rule_id = QoS._cookie_to_qosid(cookie)
863             delete_ids.setdefault(vid, '')
864             delete_ids[vid] += (('%d' if delete_ids[vid] == ''
865                                   else ',%d') % rule_id)
866
867             msg = []
868             for vid, rule_ids in delete_ids.items():
869                 del_msg = {'result': 'success',
870                            'details': 'deleted. : QoS ID=%s' % rule_ids}
871
872                 if vid != VLANID_NONE:
873                     del_msg.setdefault(REST_VLANID, vid)
874                 msg.append(del_msg)
875
876             return REST_COMMAND_RESULT, msg
877
878     @rest_command
879     def set_meter(self, rest, vlan_id, waiters):
880         if self.version == ofproto_v1_0.OFP_VERSION:
881             raise ValueError('set_meter operation is not supported')
882
883         msgs = []
884         msg = self._set_meter(rest, waiters)
885         msgs.append(msg)
886         return REST_COMMAND_RESULT, msgs
887
888     def _set_meter(self, rest, waiters):
889         cmd = self.dp.ofproto.OFPMC_ADD
890         try:
891             self.ofctl.mod_meter_entry(self.dp, rest, cmd)
892         except:
893             raise ValueError('Invalid meter parameter.')
894
895         msg = {'result': 'success',
896                'details': 'Meter added. : Meter ID=%s' %
897                           rest[REST_METER_ID]}
898         return msg
899
900     @rest_command
901     def get_meter(self, rest, vlan_id, waiters):
902         if (self.version == ofproto_v1_0.OFP_VERSION or
903             self.version == ofproto_v1_2.OFP_VERSION):
904             raise ValueError('get_meter operation is not supported')
905
906         msgs = self.ofctl.get_meter_stats(self.dp, waiters)
907         return REST_COMMAND_RESULT, msgs
908
909     @rest_command
910     def delete_meter(self, rest, vlan_id, waiters):
911         if (self.version == ofproto_v1_0.OFP_VERSION or

```

```

911         self.version == ofproto_v1_2.OFP_VERSION):
912             raise ValueError('delete_meter operation is not supported')
913
914     cmd = self.dp.ofproto.OFPMC_DELETE
915     try:
916         self.ofctl.mod_meter_entry(self.dp, rest, cmd)
917     except:
918         raise ValueError('Invalid meter parameter.')
919
920     msg = {'result': 'success',
921            'details': 'Meter deleted. : Meter ID=%s' %
922                       rest[REST_METER_ID]}
923     return REST_COMMAND_RESULT, msg
924
925     def _to_of_flow(self, cookie, priority, match, actions):
926         flow = {'cookie': cookie,
927                 'priority': priority,
928                 'flags': 0,
929                 'idle_timeout': 0,
930                 'hard_timeout': 0,
931                 'match': match,
932                 'actions': actions}
933         return flow
934
935     def _to_rest_rule(self, flow):
936         ruleid = QoS._cookie_to_qosid(flow[REST_COOKIE])
937         rule = {REST_QOS_ID: ruleid}
938         rule.update({REST_PRIORITY: flow[REST_PRIORITY]})  

939         rule.update(Match.to_rest(flow))
940         rule.update(Action.to_rest(flow))
941         return rule
942
943
944 class Match(object):
945
946     _CONVERT = {REST_DL_TYPE:
947                 {REST_DL_TYPE_ARP: ether.ETH_TYPE_ARP,
948                  REST_DL_TYPE_IPV4: ether.ETH_TYPE_IP,
949                  REST_DL_TYPE_IPV6: ether.ETH_TYPE_IPV6},
950                 REST_NW_PROTO:
951                 {REST_NW_PROTO_TCP: inet.IPPROTO_TCP,
952                  REST_NW_PROTO_UDP: inet.IPPROTO_UDP,
953                  REST_NW_PROTO_ICMP: inet.IPPROTO_ICMP,
954                  REST_NW_PROTO_ICMPV6: inet.IPPROTO_ICMPV6}}
955
956     @staticmethod
957     def to_openflow(rest):
958
959         def __inv_combi(msg):
960             raise ValueError('Invalid combination: [%s]' % msg)
961
962         def __inv_2and1(*args):
963             __inv_combi('%s=%s and %s' % (args[0], args[1], args[2]))
```

```

965     def __inv_2and2(*args):
966         __inv_combi('%s=%s and %s=%s' % (
967             args[0], args[1], args[2], args[3]))
968
969     def __inv_1and1(*args):
970         __inv_combi('%s and %s' % (args[0], args[1]))
971
972     def __inv_1and2(*args):
973         __inv_combi('%s and %s=%s' % (args[0], args[1], args[2]))
974
975     match = {}
976
977     # error check
978     dl_type = rest.get(REST_DL_TYPE)
979     nw_proto = rest.get(REST_NW_PROTO)
980     if dl_type is not None:
981         if dl_type == REST_DL_TYPE_ARP:
982             if REST_SRC_IPV6 in rest:
983                 __inv_2and1(
984                     REST_DL_TYPE, REST_DL_TYPE_ARP, REST_SRC_IPV6)
985             if REST_DST_IPV6 in rest:
986                 __inv_2and1(
987                     REST_DL_TYPE, REST_DL_TYPE_ARP, REST_DST_IPV6)
988             if REST_DSCP in rest:
989                 __inv_2and1(
990                     REST_DL_TYPE, REST_DL_TYPE_ARP, REST_DSCP)
991             if nw_proto:
992                 __inv_2and1(
993                     REST_DL_TYPE, REST_DL_TYPE_ARP, REST_NW_PROTO)
994         elif dl_type == REST_DL_TYPE_IPV4:
995             if REST_SRC_IPV6 in rest:
996                 __inv_2and1(
997                     REST_DL_TYPE, REST_DL_TYPE_IPV4, REST_SRC_IPV6)
998             if REST_DST_IPV6 in rest:
999                 __inv_2and1(
1000                     REST_DL_TYPE, REST_DL_TYPE_IPV4, REST_DST_IPV6)
1001             if nw_proto == REST_NW_PROTO_ICMPV6:
1002                 __inv_2and2(
1003                     REST_DL_TYPE, REST_DL_TYPE_IPV4,
1004                     REST_NW_PROTO, REST_NW_PROTO_ICMPV6)
1005         elif dl_type == REST_DL_TYPE_IPV6:
1006             if REST_SRC_IP in rest:
1007                 __inv_2and1(
1008                     REST_DL_TYPE, REST_DL_TYPE_IPV6, REST_SRC_IP)
1009             if REST_DST_IP in rest:
1010                 __inv_2and1(
1011                     REST_DL_TYPE, REST_DL_TYPE_IPV6, REST_DST_IP)
1012             if nw_proto == REST_NW_PROTO_ICMP:
1013                 __inv_2and2(
1014                     REST_DL_TYPE, REST_DL_TYPE_IPV6,
1015                     REST_NW_PROTO, REST_NW_PROTO_ICMP)
1016         else:
1017             raise ValueError('Unknown dl_type : %s' % dl_type)
1018     else:

```

```

1019     if REST_SRC_IP in rest:
1020         if REST_SRC_IPV6 in rest:
1021             __inv_1and1(REST_SRC_IP, REST_SRC_IPV6)
1022         if REST_DST_IPV6 in rest:
1023             __inv_1and1(REST_SRC_IP, REST_DST_IPV6)
1024         if nw_proto == REST_NW_PROTO_ICMPV6:
1025             __inv_1and2(
1026                 REST_SRC_IP, REST_NW_PROTO, REST_NW_PROTO_ICMPV6
1027             )
1028             rest[REST_DL_TYPE] = REST_DL_TYPE_IPV4
1029     elif REST_DST_IP in rest:
1030         if REST_SRC_IPV6 in rest:
1031             __inv_1and1(REST_DST_IP, REST_SRC_IPV6)
1032         if REST_DST_IPV6 in rest:
1033             __inv_1and1(REST_DST_IP, REST_DST_IPV6)
1034         if nw_proto == REST_NW_PROTO_ICMPV6:
1035             __inv_1and2(
1036                 REST_DST_IP, REST_NW_PROTO, REST_NW_PROTO_ICMPV6
1037             )
1038             rest[REST_DL_TYPE] = REST_DL_TYPE_IPV4
1039     elif REST_SRC_IPV6 in rest:
1040         if nw_proto == REST_NW_PROTO_ICMP:
1041             __inv_1and2(
1042                 REST_SRC_IPV6, REST_NW_PROTO, REST_NW_PROTO_ICMP
1043             )
1044             rest[REST_DL_TYPE] = REST_DL_TYPE_IPV6
1045     elif REST_DST_IPV6 in rest:
1046         if nw_proto == REST_NW_PROTO_ICMP:
1047             __inv_1and2(
1048                 REST_DST_IPV6, REST_NW_PROTO, REST_NW_PROTO_ICMP
1049             )
1050             rest[REST_DL_TYPE] = REST_DL_TYPE_IPV6
1051     elif REST_DL_TYPE in rest:
1052         # Apply dl_type ipv4, if doesn't specify dl_type
1053         rest[REST_DL_TYPE] = REST_DL_TYPE_IPV4
1054     else:
1055         if nw_proto == REST_NW_PROTO_ICMP:
1056             rest[REST_DL_TYPE] = REST_DL_TYPE_IPV4
1057         elif nw_proto == REST_NW_PROTO_ICMPV6:
1058             rest[REST_DL_TYPE] = REST_DL_TYPE_IPV6
1059         elif nw_proto == REST_NW_PROTO_TCP or \
1060             nw_proto == REST_NW_PROTO_UDP:
1061             raise ValueError('no dl_type was specified')
1062         else:
1063             raise ValueError('Unknown nw_proto: %s' % nw_proto)
1064
1065     for key, value in rest.items():
1066         if key in Match._CONVERT:
1067             if value in Match._CONVERT[key]:
1068                 match.setdefault(key, Match._CONVERT[key][value])
1069             else:
1070                 raise ValueError('Invalid rule parameter. : key=%s' %
1071                               key)
1072         else:

```

```

1068         match . setdefault (key , value )
1069
1070     return match
1071
1072     @staticmethod
1073     def to_rest (openflow):
1074         of_match = openflow [REST_MATCH]
1075
1076         mac_dontcare = mac . haddr_to_str (mac . DONTCARE)
1077         ip_dontcare = '0.0.0.0'
1078         ipv6_dontcare = '::'
1079
1080         match = {}
1081         for key , value in of_match . items ():
1082             if key == REST_SRC_MAC or key == REST_DST_MAC:
1083                 if value == mac_dontcare:
1084                     continue
1085             elif key == REST_SRC_IP or key == REST_DST_IP:
1086                 if value == ip_dontcare:
1087                     continue
1088             elif key == REST_SRC_IPV6 or key == REST_DST_IPV6:
1089                 if value == ipv6_dontcare:
1090                     continue
1091             elif value == 0:
1092                 continue
1093
1094             if key in Match . _CONVERT:
1095                 conv = Match . _CONVERT [key]
1096                 conv = dict ((value , key) for key , value in conv . items ())
1097                 match . setdefault (key , conv [value ])
1098             else :
1099                 match . setdefault (key , value )
1100
1101     return match
1102
1103     @staticmethod
1104     def to_mod_openflow (of_match):
1105         mac_dontcare = mac . haddr_to_str (mac . DONTCARE)
1106         ip_dontcare = '0.0.0.0'
1107         ipv6_dontcare = '::'
1108
1109         match = {}
1110         for key , value in of_match . items ():
1111             if key == REST_SRC_MAC or key == REST_DST_MAC:
1112                 if value == mac_dontcare:
1113                     continue
1114             elif key == REST_SRC_IP or key == REST_DST_IP:
1115                 if value == ip_dontcare:
1116                     continue
1117             elif key == REST_SRC_IPV6 or key == REST_DST_IPV6:
1118                 if value == ipv6_dontcare:
1119                     continue
1120             elif value == 0:
1121                 continue

```

```

1122         match.setdefault(key, value)
1123
1124     return match
1125
1126
1127
1128 class Action(object):
1129
1130     @staticmethod
1131     def to_rest(flow):
1132         if REST_ACTION in flow:
1133             actions = []
1134             for act in flow[REST_ACTION]:
1135                 field_value = re.search('SET_FIELD: \{ip_dscp:(\d+)', act)
1136                 if field_value:
1137                     actions.append({REST_ACTION_MARK: field_value.group(1)})
1138                 meter_value = re.search('METER:(\d+)', act)
1139                 if meter_value:
1140                     actions.append({REST_ACTION_METER: meter_value.group(1)})
1141                 queue_value = re.search('SET_QUEUE:(\d+)', act)
1142                 if queue_value:
1143                     actions.append({REST_ACTION_QUEUE: queue_value.group(1)})
1144             action = {REST_ACTION: actions}
1145         else:
1146             action = {REST_ACTION: 'Unknown action type.'}
1147
1148     return action

```

A.8 Topology_qos_TRAFFIC_GEN.py

```

1 from mininet.net import Mininet
2 from mininet.node import Controller, RemoteController, OVSCollector
3 from mininet.node import CPULimitedHost, Host, Node
4 from mininet.node import OVSKernelSwitch, UserSwitch
5 from mininet.node import IVSSwitch
6 from mininet.cli import CLI
7 from mininet.log import setLogLevel, info
8 from mininet.link import TCLink, Intf
9 from subprocess import call
10 import threading
11 import subprocess
12 import random
13 import os
14 import time
15 import datetime
16 import json
17 import sys

```

```

18 import ditg
19
20 from Controller_commands import *
21
22
23 def myNetwork():
24     max_rate_queue=100#Mbps
25     max_rate_queue=max_rate_queue*1000000
26     Default=str(max_rate_queue*20/100)#20%
27     Premium=str(max_rate_queue*80/100)#80%
28     Gold=str(max_rate_queue*100/100)#100%
29
30     change_values=6#change every number*5 minutes
31     Q0=False
32     #Stress Queue 1 (2,4,6,7)
33     Q1=False
34     #Stress Queue 2 (5)
35     Q2=False
36     #moltiplicator initialization
37     F0_max=2
38     F1_max=300
39     F2_max=300
40     F0=1
41     F1=1
42     F2=1
43
44     net=Mininet( topo=None,
45                 build=False,
46                 ipBase='10.0.0.0/8')
47
48     info('***Adding controller\n')
49     c0=net.addController(name='c0',
50                           controller=RemoteController,
51                           ip='127.0.0.1',
52                           protocol='tcp',
53                           port=6633)
54
55     info('***Adding switches\n')
56     s1 = net.addSwitch('s1',dpid='0000000000000001',protocols="OpenFlow13")
57     s2 = net.addSwitch('s2',dpid='0000000000000002',protocols="OpenFlow13")
58
59     info('***Adding Host\n')
60     h0 = net.addHost('h0', ip='10.10/24', mac='00:00:00:00:00:0a')
61     h1 = net.addHost('h1', ip='10.11/24', mac='00:00:00:00:00:0b')
62     h2 = net.addHost('h2', ip='10.12/24', mac='00:00:00:00:00:0c')
63     h3 = net.addHost('h3', ip='10.13/24', mac='00:00:00:00:00:0d')
64     h4 = net.addHost('h4', ip='10.14/24', mac='00:00:00:00:00:0e')
65     h5 = net.addHost('h5', ip='10.15/24', mac='00:00:00:00:00:0f')
66
67     info('***Adding Link\n')
68     net.addLink(s1,s2,2,2)
69

```

```

70 net.addLink(s1, h0,3,0)
71 net.addLink(s1, h1,4,0)
72 net.addLink(s1, h2,5,0)
73 net.addLink(s2, h3,3,0)
74 net.addLink(s2, h4,4,0)
75 net.addLink(s2, h5,5,0)

76
77 info( '***Starting Network\n')
78 net.build()

79
80 info( '***Starting Controllers\n')
81 for controller in net.controllers:
82     controller.start()

83
84 info( '***Starting Switches\n')
85 net.get('s1').start([c0])
86 net.get('s2').start([c0])

87
88
89 #Activation of manager in listening on port 6632
90 os.popen("sudo -S ovs-vsctl set-manager ptcp:6632", 'w').write("Ao70pa45")
91 time.sleep(2)
92 NET = get_switchis()
93 if NET != "NO NET" and NET!="[ , ]":
94     i=1
95     while i<NET.find("]"):
96         mom_NET=NET[ i :]
97         datapath=NET[ i : i+mom_NET.find(",")]
98         i=i+mom_NET.find(",") +2
99         port_id = switch_ports_name(datapath)
100        for k in range(0,len(port_id)):
101            if len(port_id[k])<=2:
102                pp=port_id[k]
103                print pp
104                ovssctl_set_bridge(port_id[k])
105        time.sleep(0.2)
106        i=1
107        while i<NET.find("]"):
108            mom_NET=NET[ i :]
109            datapath=NET[ i : i+mom_NET.find(",")]
110            i=i+mom_NET.find(",") +2
111            port_id = switch_ports_name(datapath)
112            ovssctl_set_bridge(datapath)
113            IP_Flag=True
114            for index in range(0,len(port_id)):
115                if port_id[index]=="s1-eth2" or port_id[index]=="s2-eth2":
116                    set_queue(datapath, port_id[index], str(
max_rate_queue), "{\"max_rate\": \"\""+Default+"\"}, {\"max_rate\": \""
+"Premium+"\"}, {\"min_rate\": \"\""+Gold+"\"}")
117                    port = port_id[index][port_id[index].find("h")+1:]
118                    if port_id[index]=="s1-eth2":
119                        IP_Destination="10.0.0.11"

```

```

120         if port_id[index]==”s2–eth2”:
121             IP_Destination=”10.0.0.13”
122             set_Telecom_queue(datapath , port , IP_Flag ,
123             IP_Destination)
124             for index in range(0,len(port_id)):
125                 if port_id[index]==”s1–eth4” or port_id[index]==”s2–eth3
126                 ”:
127                     port = port_id[index][port_id[index].find(”h”)+1:]
128                     if port_id[index]==”s1–eth4”:
129                         IP_Destination=”10.0.0.13”
130                         if port_id[index]==”s2–eth3”:
131                             IP_Destination=”10.0.0.11”
132                             set_Telecom_queue(datapath , port , IP_Flag ,
133                             IP_Destination)
134
# Import CSV data
135 print ’Csv Import’
136 serv = ditg .pd.read_csv(ditg .CSV, sep=’;’, usecols=[0], skiprows
137 =[0]) # serv 0 tx
138 time_values = serv .values
139 serv = ditg .pd.read_csv(ditg .CSV, sep=’;’, usecols=[1], skiprows
140 =[0]) # serv 0 rx
141 serv_0_tx = serv .values
142 serv = ditg .pd.read_csv(ditg .CSV, sep=’;’, usecols=[2], skiprows
143 =[0]) # serv 0 rx
144 serv_0_rx = serv .values
145 serv = ditg .pd.read_csv(ditg .CSV, sep=’;’, usecols=[3], skiprows
146 =[0]) # serv 1 tx
147 serv_1_tx = serv .values
148 serv = ditg .pd.read_csv(ditg .CSV, sep=’;’, usecols=[4], skiprows
149 =[0]) # serv 1 rx
150 serv_1_rx = serv .values
151 serv = ditg .pd.read_csv(ditg .CSV, sep=’;’, usecols=[5], skiprows
152 =[0]) # serv 2 tx
153 serv_2_tx = serv .values
154 serv = ditg .pd.read_csv(ditg .CSV, sep=’;’, usecols=[6], skiprows
155 =[0]) # serv 2 rx
156 serv_2_rx = serv .values
157 serv = ditg .pd.read_csv(ditg .CSV, sep=’;’, usecols=[15], skiprows
158 =[0]) # serv 3 tx
159 serv_3_tx = serv .values
160 serv = ditg .pd.read_csv(ditg .CSV, sep=’;’, usecols=[16], skiprows
161 =[0]) # serv 3 rx
162 serv_3_rx = serv .values
163 serv = ditg .pd.read_csv(ditg .CSV, sep=’;’, usecols=[11], skiprows
164 =[0]) # serv 4 tx
165 serv_4_tx = serv .values
166 serv = ditg .pd.read_csv(ditg .CSV, sep=’;’, usecols=[12], skiprows
167 =[0]) # serv 4 rx
168 serv_4_rx = serv .values
169 serv = ditg .pd.read_csv(ditg .CSV, sep=’;’, usecols=[7], skiprows
170 =[0]) # serv 5 tx
171 serv_5_tx = serv .values

```

```

158 serv = ditg.pd.read_csv(ditg.CSV, sep=';', usecols=[8], skiprows
159 =[0]) # serv 5 rx
160 serv_5_rx = serv.values
161 serv = ditg.pd.read_csv(ditg.CSV, sep=';', usecols=[9], skiprows
162 =[0]) # serv 6 tx
163 serv_6_tx = serv.values
164 serv = ditg.pd.read_csv(ditg.CSV, sep=';', usecols=[10], skiprows
165 =[0]) # serv 6 rx
166 serv_6_rx = serv.values
167 serv = ditg.pd.read_csv(ditg.CSV, sep=';', usecols=[13], skiprows
168 =[0]) # serv 7 tx
169 serv_7_tx = serv.values
170 serv = ditg.pd.read_csv(ditg.CSV, sep=';', usecols=[14], skiprows
171 =[0]) # serv 7 rx
172 serv_7_rx = serv.values
173
174 i = 0
175 j = 0
176 # pkts
177 n = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
178 # pkts per second
179 avg = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
180 F0=F0_max
181 F1=1
182 F2=1
183 print 'Wait for time alignment'
184 wait=ditg.TIME/60
185 check_time=False
186 while check_time==False:
187     now=datetime.datetime.now()
188     time.sleep(0.1)
189     if now.minute%wait==0:
190         check_time=True
191         if len(str(now.minute))==1:
192             starting_time=str(now.hour)+':0'+str(now.minute)
193         else:
194             starting_time=str(now.hour)+':'+str(now.minute)
195         print starting_time
196         k=0
197         for index in time_values:
198             if index==starting_time:
199                 i=k
200                 break
201             k=k+1
202         print 'Starting Time: '+str(time_values[i])
203         while j < ditg.SIM_N:
204             #calculate the multiplicators
205
206             if Q0:
207                 if j%change_values==0:
208                     if F0==1:
209                         F0=F0_max
210                     else:
211                         F0=1

```

```

207     if Q1:
208         if j%change_values==0:
209             if F1==1:
210                 F1=F1_max
211             else:
212                 F1=1
213     if Q2:
214         if j%change_values==0:
215             if F2==1:
216                 F2=F2_max
217             else:
218                 F2=1
219
220
221
222 # Server start
223 print 'Start ITGRecv'
224 h1.cmd('ITGRecv &')
225 h3.cmd('ITGRecv &')
226 time.sleep(2)
227 # Sum of packets
228 sum_in = 0
229 sum_out = 0
230 # Serv 0 tx
231 n[0] = int(serv_0_tx[i])*F0 / ditg.SCALE + 1
232 avg[0] = n[0] / (ditg.TIME-10) + 1
233 if avg[0] > 0 and n[0] > 0:
234     com = ditg.createCmd_2(dst=ditg.dst, port="10001", tos=ditg.
SERV_0, nPkts=str(n[0]), avg=str(avg[0]))
235     print(com)
236     h1.cmd(com)
237     sum_in = sum_in + n[0]
238
239 # Serv 0 rx
240 n[0] = int(serv_0_rx[i])*F0 / ditg.SCALE + 1
241 avg[0] = n[0] / (ditg.TIME-10) + 1
242 if avg[0] > 0 and n[0] > 0:
243     com = ditg.createCmd_2(dst=ditg.src, port="10001", tos=ditg.
SERV_0, nPkts=str(n[0]), avg=str(avg[0]))
244     print(com)
245     h3.cmd(com)
246     sum_out = sum_out + n[0]
247
248 # Serv 1 tx
249 n[1] = int(serv_1_tx[i])*F0 / ditg.SCALE + 1
250 avg[1] = n[1] / (ditg.TIME-10) + 1
251 if avg[1] > 0 and n[1] > 0:
252     com = ditg.createCmd_2(dst=ditg.dst, port="10002", tos=ditg.
SERV_1, nPkts=str(n[1]), avg=str(avg[1]))
253     print(com)
254     h1.cmd(com)
255     sum_in = sum_in + n[1]
256
257 # Serv 1 rx

```

```

258     n[1] = int(serv_1_rx[i])*F0 / ditg.SCALE + 1
259     avg[1] = n[1] / (ditg.TIME-10) + 1
260     if avg[1] > 0 and n[1] > 0:
261         com = ditg.createCmd_2(dst=ditg.src, port="10002", tos=ditg.
SERV_1, nPkts=str(n[1]), avg=str(avg[1]))
262         print(com)
263         h3.cmd(com)
264         sum_out = sum_out + n[1]
265
266     # Serv 2 tx
267     n[2] = int(serv_2_tx[i])*F1 / ditg.SCALE + 1
268     avg[2] = n[2] / (ditg.TIME-10) + 1
269     if avg[2] > 0 and n[2] > 0:
270         com = ditg.createCmd_2(dst=ditg.dst, port="10003", tos=ditg.
SERV_2, nPkts=str(n[2]), avg=str(avg[2]))
271         print(com)
272         h1.cmd(com)
273         sum_in = sum_in + n[2]
274
275     # Serv 2 rx
276     n[2] = int(serv_2_rx[i])*F1 / ditg.SCALE + 1
277     avg[2] = n[2] / (ditg.TIME-10) + 1
278     if avg[2] > 0 and n[2] > 0:
279         com = ditg.createCmd_2(dst=ditg.src, port="10003", tos=ditg.
SERV_2, nPkts=str(n[2]), avg=str(avg[2]))
280         print(com)
281         h3.cmd(com)
282         sum_out = sum_out + n[2]
283
284     # Serv 3 tx
285     n[3] = int(serv_3_tx[i])*F0 / ditg.SCALE + 1
286     avg[3] = n[3] / (ditg.TIME-10) + 1
287     if avg[3] > 0 and n[3] > 0:
288         com = ditg.createCmd_2(dst=ditg.dst, port="10004", tos=ditg.
SERV_3, nPkts=str(n[3]), avg=str(avg[3]))
289         print(com)
290         h1.cmd(com)
291         sum_in = sum_in + n[3]
292
293     # Serv 3 rx
294     n[3] = int(serv_3_rx[i])*F0 / ditg.SCALE + 1
295     avg[3] = n[3] / (ditg.TIME-10) + 1
296     if avg[3] > 0 and n[3] > 0:
297         com = ditg.createCmd_2(dst=ditg.src, port="10004", tos=ditg.
SERV_3, nPkts=str(n[3]), avg=str(avg[3]))
298         print(com)
299         h3.cmd(com)
300         sum_out = sum_out + n[3]
301
302     # Serv 4 tx
303     n[4] = int(serv_4_tx[i])*F1 / ditg.SCALE + 1
304     avg[4] = n[4] / (ditg.TIME-10) + 1
305     if avg[4] > 0 and n[4] > 0:

```

```

306     com = ditg.createCmd_2(dst=ditg.dst, port="10005", tos=ditg.
307 SERV_4, nPkts=str(n[4]), avg=str(avg[4]))
308     print(com)
309     h1.cmd(com)
310     sum_in = sum_in + n[4]
311
312     # Serv 4 rx
313     n[4] = int(serv_4_rx[i])*F1 / ditg.SCALE + 1
314     avg[4] = n[4] / (ditg.TIME-10) + 1
315     if avg[4] > 0 and n[4] > 0:
316         com = ditg.createCmd_2(dst=ditg.src, port="10005", tos=ditg.
317 SERV_4, nPkts=str(n[4]), avg=str(avg[4]))
318         print(com)
319         h3.cmd(com)
320         sum_out = sum_out + n[4]
321
322     # Serv 5 tx
323     n[5] = int(serv_5_tx[i])*F2 / ditg.SCALE + 1
324     avg[5] = n[5] / (ditg.TIME-10) + 1
325     if avg[5] > 0 and n[5] > 0:
326         com = ditg.createCmd_2(dst=ditg.dst, port="10006", tos=ditg.
327 SERV_5, nPkts=str(n[5]), avg=str(avg[5]))
328         print(com)
329         h1.cmd(com)
330         sum_in = sum_in + n[5]
331
332     # Serv 5 rx
333     n[5] = int(serv_5_rx[i])*F2 / ditg.SCALE + 1
334     avg[5] = n[5] / (ditg.TIME-10) + 1
335     if avg[5] > 0 and n[5] > 0:
336         com = ditg.createCmd_2(dst=ditg.src, port="10006", tos=ditg.
337 SERV_5, nPkts=str(n[5]), avg=str(avg[5]))
338         print(com)
339         h3.cmd(com)
340         sum_out = sum_out + n[5]
341
342     # Serv 6 tx
343     n[6] = int(serv_6_tx[i])*F1 / ditg.SCALE + 1
344     avg[6] = n[6] / (ditg.TIME-10) + 1
345     if avg[6] > 0 and n[6] > 0:
346         com = ditg.createCmd_2(dst=ditg.dst, port="10007", tos=ditg.
347 SERV_6, nPkts=str(n[6]), avg=str(avg[6]))
348         print(com)
349         h1.cmd(com)
350         sum_in = sum_in + n[6]
351
352     # Serv 6 rx
353     n[6] = int(serv_6_rx[i])*F1 / ditg.SCALE + 1
354     avg[6] = n[6] / (ditg.TIME-10) + 1
355     if avg[6] > 0 and n[6] > 0:
356         com = ditg.createCmd_2(dst=ditg.src, port="10007", tos=ditg.
357 SERV_6, nPkts=str(n[6]), avg=str(avg[6]))
358         print(com)
359         h3.cmd(com)

```

```

354         sum_out = sum_out + n[6]
355
356
357     # Serv 7 tx
358     n[7] = int(serv_7_tx[i])*F1 / ditg.SCALE + 1
359     avg[7] = n[7] / (ditg.TIME-10) + 1
360     if avg[7] > 0 and n[7] > 0:
361         com = ditg.createCmd_2(dst=ditg.dst, port="10008", tos=ditg.
362 SERV_7, nPkts=str(n[7]), avg=str(avg[7]))
363         print(com)
364         h1.cmd(com)
365         sum_in = sum_in + n[7]
366
367     # Serv 7 rx
368     n[7] = int(serv_7_rx[i])*F1 / ditg.SCALE + 1
369     avg[7] = n[7] / (ditg.TIME-10) + 1
370     if avg[7] > 0 and n[7] > 0:
371         com = ditg.createCmd_2(dst=ditg.src, port="10008", tos=ditg.
372 SERV_7, nPkts=str(n[7]), avg=str(avg[7]))
373         print(com)
374         h3.cmd(com)
375         sum_out = sum_out + n[7]
376
377     j = j+1
378     i=i+1
379     if i % ditg.SIZE==0 and i!=0:
380         i=0
381     print i
382     print('Sum of Packets IN: ' + str(sum_in))
383     print('Sum of Packets OUT: ' + str(sum_out))
384
385     time.sleep(61)
386     check_time=False
387     while check_time==False:
388         now=datetime.datetime.now()
389         if now.minute%wait==0:
390             check_time=True
391             print "Time: "+str(now)
392             print 'Database Time: '+str(time_values[i])
393         else:
394             time.sleep(1)
395     h1.cmd('kill %ITGSend')
396     h1.cmd('kill %ITGRecv')
397     CLI(net)
398     net.stop()
399
400 if __name__=='__main__':
401     setLogLevel('info')
402     myNetwork()

```

A.9 ditg.py

```
1 #!/usr/bin/python
2 import pandas as pd
3 import time
4
5 # Constants
6 # Value of Interval Time in second
7 TIME = 300
8 TIME_MS = TIME * 1000
9 # Scale factor for the packet rate. It will be equal to [ pkts / scale ]
10 # per second.
11 SCALE = 400
12 # CSV Entries
13 SIZE = 576
14 # Simulation Time in CSV hours
15 SIM = 720
16 # Simulation in number of intervals
17 SIM_N = SIM * 12
18 computername="jedi"
19 CSV = '/home/' + computername + '/Scrivania/RyuDatapathMonitor-master/CSV/
    vlan_interfaccia1_DOS.csv'
20 # ToS
21 SERV_0 = "0"
22 SERV_1 = "32"
23 SERV_2 = "72"
24 SERV_3 = "96"
25 SERV_4 = "136"
26 SERV_5 = "160"
27 SERV_6 = "192"
28 SERV_7 = "224"
29
30 # IDT_OPT
31 # constant
32 c_idt = "-C"
33 #poisson
34 p_idt = "-O"
35 #esponential
36 e_idt ="-E"
37
38 # PS_OPT
39 # constant
40 c_ps = "-c"
41 # poisson
42 p_ps = "-o"
43 # esponential
44 e_ps ="-e"
45
46 # default value for protocol and packet size
47 DEFAULT_P = "UDP"
48 DEFAULT_PS = "512"
```

```

50 # Host Ip
51 src = "10.0.0.11"
52 dst = "10.0.0.13"
53
54 # Type of distribution
55 choice_i = c_idt
56 choice_s = c_ps
57
58
59 # Cmd creation
60 def createCmd(src , dst , tos , nPkts , avg , protocol=DEFAULT_P, ps_dim=
61 DEFAULT_PS):
62     com = 'ITGManager ' + src + ' -a ' + dst + ' -b ' + tos + ' ' +
63     choice_i + ' ' + str(avg) + ' ' + choice_s + ' ' + ps_dim + ' -t ' +
64     str(TIME_MS-8000)
65     return com
66
67
68 # Cmd creation
69 def createCmd_2(dst , port , tos , nPkts , avg , protocol=DEFAULT_P, ps_dim=
70 DEFAULT_PS):
71     com = 'ITGSend -a ' + dst + ' -rp ' + str(port) + ' -b ' + tos + ' ' +
72     + choice_i + ' ' + str(avg) + ' ' + choice_s + ' ' + ps_dim + ' -t ' +
73     ' + str(TIME_MS-10000) + '&
74     return com
75
76
77 # Cmd creation Iperf
78 def createCmd_3(dst , port , tos , nPkts , avg , protocol=DEFAULT_P, ps_dim=
79 DEFAULT_PS):
80     com = 'iperf3 -c ' + dst + ' -p ' + str(port) + ' -S ' + tos + ' ' +
81     ' -k ' + str(nPkts) + '&
82     return com

```


Appendix B

Python Codes For Dedicated Hardware Network Devices

B.1 Traffic_Real_Hwd.py

```
2
3 from subprocess import call
4 import threading
5 import subprocess
6 import random
7 import os
8 import time
9 import datetime
10 import json
11 import sys
12 import ditg
13 import psutil
14 import SDL_DS1307
15
16
17 def f1():
18     IP_SRC='169.254.207.222'
19     IP_DST = "169.254.181.98"
20     os.system('sudo ifconfig eth0 '+IP_SRC+' netmask 255.255.0.0')
21     ds1307 = SDL_DS1307.SDL_DS1307(1, 0x68)
22     time.sleep(2)
23     now=ds1307.read_datetime()
24     str_time='''+str(now.year)+ '-' +str(now.month)+ '-' +str(now.day)+ ' '
25     str_time+=str(now.hour)+ ':' +str(now.minute)+ ':' +str(now.second)+ '''
26     os.system('sudo date --set '+str_time)
27
28     print ('Csv Import')
29     serv = ditg.pd.read_csv(ditg.CSV, sep=';', usecols=[0], skiprows
30     =[0]) # serv 0 tx
31     time_values = serv.values
32     serv = ditg.pd.read_csv(ditg.CSV, sep=';', usecols=[1], skiprows
33     =[0]) # serv 0 tx
34     serv_0_tx = serv.values
```

```

32 serv = ditg .pd.read_csv(ditg .CSV, sep=';', usecols=[2], skiprows
33 =[0]) # serv 0 rx
34 serv_0_rx = serv .values
35 serv = ditg .pd.read_csv(ditg .CSV, sep=';', usecols=[3], skiprows
36 =[0]) # serv 1 tx
37 serv_1_tx = serv .values
38 serv = ditg .pd.read_csv(ditg .CSV, sep=';', usecols=[4], skiprows
39 =[0]) # serv 1 rx
40 serv_1_rx = serv .values
41 serv = ditg .pd.read_csv(ditg .CSV, sep=';', usecols=[5], skiprows
42 =[0]) # serv 2 tx
43 serv_2_tx = serv .values
44 serv = ditg .pd.read_csv(ditg .CSV, sep=';', usecols=[6], skiprows
45 =[0]) # serv 2 rx
46 serv_2_rx = serv .values
47 serv = ditg .pd.read_csv(ditg .CSV, sep=';', usecols=[15], skiprows
48 =[0]) # serv 3 tx
49 serv_3_tx = serv .values
50 serv = ditg .pd.read_csv(ditg .CSV, sep=';', usecols=[16], skiprows
51 =[0]) # serv 3 rx
52 serv_3_rx = serv .values
53 serv = ditg .pd.read_csv(ditg .CSV, sep=';', usecols=[11], skiprows
54 =[0]) # serv 4 tx
55 serv_4_tx = serv .values
56 serv = ditg .pd.read_csv(ditg .CSV, sep=';', usecols=[12], skiprows
57 =[0]) # serv 4 rx
58 serv_4_rx = serv .values
59 serv = ditg .pd.read_csv(ditg .CSV, sep=';', usecols=[7], skiprows
60 =[0]) # serv 5 tx
61 serv_5_tx = serv .values
62 serv = ditg .pd.read_csv(ditg .CSV, sep=';', usecols=[8], skiprows
63 =[0]) # serv 5 rx
64 serv_5_rx = serv .values
65 serv = ditg .pd.read_csv(ditg .CSV, sep=';', usecols=[9], skiprows
66 =[0]) # serv 6 tx
67 serv_6_tx = serv .values
68 serv = ditg .pd.read_csv(ditg .CSV, sep=';', usecols=[10], skiprows
69 =[0]) # serv 6 rx
70 serv_6_rx = serv .values
71
72 i = 0
73 j = 0
74 ### pkts
75 n = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
76 # pkts per second
77 avg = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
78 F0_max=2
79 F0=F0_max

```

```

71 F1=1
72 F2=1
73 print ('Wait for time alignment')
74 wait=ditg.TIME/60
75 check_time=False
76 while check_time==False:
77     now=datetime.datetime.now()
78     time.sleep(0.1)
79     try:
80         if now.minute%wait==0:
81             check_time=True
82             if len(str(now.minute))==1:
83                 starting_time=str(now.hour)+':0'+str(now.minute)
84             else:
85                 starting_time=str(now.hour)+':'+' '+str(now.minute)
86             print (starting_time)
87             k=0
88             for index in time_values:
89                 if index==starting_time:
90                     i=k
91                     break
92                 k=k+1
93             else:
94                 time.sleep(0.5)
95         except:
96             time.sleep(0.1)
97     print ('Starting Time: '+str(time_values[i]))
98     while j < ditg.SIM_N:
99         try:
100            now=datetime.datetime.now()
101            if len(str(now.minute))==1:
102                starting_time=str(now.hour)+':0'+str(now.minute)
103            else:
104                starting_time=str(now.hour)+':'+' '+str(now.minute)

105            if starting_time==time_values[1]:
106                print (starting_time)
107            else:
108                k=0
109                for index in time_values:
110                    if index==starting_time:
111                        i=k
112                        break
113                    k=k+1
114            except:
115                time.sleep(0.1)
116            time.sleep(4)
117            # Sum of packets
118            sum_in = 0
119            sum_out = 0
120
121
122            # Serv 0 rx
123            n[0] = int(serv_0_rx[i])*F0 / ditg.SCALE + 1
124

```

```

125     avg[0] = n[0] / (ditg.TIME-10) + 1
126     if avg[0] > 0 and n[0] > 0:
127         com = ditg.createCmd_2(dst=IP_DST, port="10001", tos=ditg.
SERV_0, nPkts=str(n[0]), avg=str(avg[0]))
128         print(com)
129         os.popen(com)
130         sum_out = sum_out + n[0]
131
132     # Serv 1 rx
133     n[1] = int(serv_1_rx[i])*F0 / ditg.SCALE + 1
134     avg[1] = n[1] / (ditg.TIME-10) + 1
135     if avg[1] > 0 and n[1] > 0:
136         com = ditg.createCmd_2(dst=IP_DST, port="10002", tos=ditg.
SERV_1, nPkts=str(n[1]), avg=str(avg[1]))
137         print(com)
138         os.popen(com)
139         sum_out = sum_out + n[1]
140
141     # Serv 2 rx
142     n[2] = int(serv_2_rx[i])*F1 / ditg.SCALE + 1
143     avg[2] = n[2] / (ditg.TIME-10) + 1
144     if avg[2] > 0 and n[2] > 0:
145         com = ditg.createCmd_2(dst=IP_DST, port="10003", tos=ditg.
SERV_2, nPkts=str(n[2]), avg=str(avg[2]))
146         print(com)
147         os.popen(com)
148         sum_out = sum_out + n[2]
149
150     # Serv 3 rx
151     n[3] = int(serv_3_rx[i])*F0 / ditg.SCALE + 1
152     avg[3] = n[3] / (ditg.TIME-10) + 1
153     if avg[3] > 0 and n[3] > 0:
154         com = ditg.createCmd_2(dst=IP_DST, port="10004", tos=ditg.
SERV_3, nPkts=str(n[3]), avg=str(avg[3]))
155         print(com)
156         os.popen(com)
157         sum_out = sum_out + n[3]
158
159     # Serv 4 rx
160     n[4] = int(serv_4_rx[i])*F1 / ditg.SCALE + 1
161     avg[4] = n[4] / (ditg.TIME-10) + 1
162     if avg[4] > 0 and n[4] > 0:
163         com = ditg.createCmd_2(dst=IP_DST, port="10005", tos=ditg.
SERV_4, nPkts=str(n[4]), avg=str(avg[4]))
164         print(com)
165         os.popen(com)
166         sum_out = sum_out + n[4]
167
168     # Serv 5 rx
169     n[5] = int(serv_5_rx[i])*F2 / ditg.SCALE + 1
170     avg[5] = n[5] / (ditg.TIME-10) + 1
171     if avg[5] > 0 and n[5] > 0:
172         com = ditg.createCmd_2(dst=IP_DST, port="10006", tos=ditg.
SERV_5, nPkts=str(n[5]), avg=str(avg[5]))

```

```

173     print(com)
174     os .popen (com)
175     sum_out = sum_out + n[5]
176
177 # Serv 6 rx
178 n[6] = int(serv_6_rx[i])*F1 / ditg .SCALE + 1
179 avg[6] = n[6] / (ditg .TIME-10) + 1
180 if avg[6] > 0 and n[6] > 0:
181     com = ditg .createCmd_2(dst=IP_DST , port="10007" , tos=ditg .
SERV_6 , nPkts=str(n[6]) , avg=str(avg[6]))
182     print(com)
183     os .popen (com)
184     sum_out = sum_out + n[6]
185
186 # Serv 7 rx
187 n[7] = int(serv_7_rx[i])*F1 / ditg .SCALE + 1
188 avg[7] = n[7] / (ditg .TIME-10) + 1
189 if avg[7] > 0 and n[7] > 0:
190     com = ditg .createCmd_2(dst=IP_DST , port="10008" , tos=ditg .
SERV_7 , nPkts=str(n[7]) , avg=str(avg[7]))
191     print(com)
192     os .popen (com)
193     sum_out = sum_out + n[7]
194
195 j = j+1
196 i=i+1
197 if i % ditg .SIZE==0 and i!=0:
198     i=0
199 print (i)
200 print('Sum of Packets OUT: ' + str(sum_out))
201
202 ##Wait next minute to avoid another detection
203 check_time=False
204 while check_time==False:
205     now=datetime.datetime.now()
206     try:
207         if (now.minute== 4 or now.minute==17 or now.minute==31
208 or now.minute==41 or now.minute==54) and now.second==0:
209             now=ds1307.read_datetime()
210             str_time='''+str(now.year)+'''+str(now.month)+'''+
211             str(now.day)+'''+str(now.hour)+':'+str(now.minute)+':'+str(now.
second)+'''
212             os .system( 'sudo date --set '+str_time)
213             now=datetime.datetime.now()
214             if now.minute%wait==1:
215                 check_time=True
216             else:
217                 time .sleep(1)
218     except:
219         time .sleep(0.1)
220
221 check_time=False
222 while check_time==False:
223     now=datetime.datetime.now()

```

```

222     try:
223         if (now.minute== 4 or now.minute==17 or now.minute==31
224         or now.minute==41 or now.minute==54) and now.second==0:
225             now=ds1307.read_datetime()
226             str_time='''+str(now.year)+’-’+str(now.month)+’-’+
227             str(now.day)+ ’+str(now.hour)+’:’+str(now.minute)+’:’+str(now.
228             second)+''''
229             os.system('sudo date --set '+str_time)
230             now=datetime.datetime.now()
231             if now.minute%wait==0:
232                 check_time=True
233                 print ("Time: "+str(now))
234                 print ('Database Time: '+str(time_values[i]))
235             else:
236                 time.sleep(1)
237             except:
238                 time.sleep(0.1)
239             try:
240                 cmd = "pidof ITGRecv"
241                 PID = subprocess.check_output(cmd, shell=True).decode("utf-8")
242             ")
243                 print ("pidof ITGRecv: "+str(PID))
244                 os.popen('sudo kill -9 '+str(PID))
245                 cmd = "pidof ITGSend"
246                 PID = subprocess.check_output(cmd, shell=True).decode("utf-8")
247             ")
248                 print ("pidof ITGSend: "+str(PID))
249                 os.popen('sudo kill -9 '+str(PID))
250             except:
251                 time.sleep(0.1)
252
253 def f2():
254     while True:
255         try:
256             print("Start ITGRecv")
257             os.popen('ITGRecv')
258         except:
259             time.sleep(1)
260 t1 = threading.Thread(target=f1, args=())
261 t2 = threading.Thread(target=f2, args=())
262
263 #Started the threads
264 t1.start()
265 time.sleep(4)
266 t2.start()
267
268 #Joined the threads
269 t1.join()
270 t2.join()

```

B.2 Start_ITGRecv.py

```
2 from subprocess import call
3 import threading
4 import subprocess
5 import random
6 import os
7 import time
8 import datetime
9 import json
10 import sys
11 import ditg
12
13 while True:
14     try:
15         print("Start ITGRecv")
16         os.popen('ITGRecv')
17     except:
18         time.sleep(1)
```

B.3 ditg.py

```
2 #!/usr/bin/python
3
4 import pandas as pd
5 import time
6
7 # Constants
8 # Value of Interval Time in second
9 TIME = 300
10 TIME_MS = TIME * 1000
11 # Scale factor for the packet rate. It will be equal to [ pkts / scale ]
12 # per second.
12 SCALE = 400
13 # CSV Entries
14 SIZE = 576
15 # Simulation Time in CSV hours
16 SIM = 720
17 # Simulation in number of intervals
18 SIM_N = SIM * 12
19 computername="pi"
20 # CSV File
21 CSV = '/home/' + computername + '/Desktop/Hardware_Interface/CSV/
22     wlan_interfaccia1_DOS.csv'
22 # ToS
23 SERV_0 = "0"
24 SERV_1 = "32"
25 SERV_2 = "72"
26 SERV_3 = "96"
27 SERV_4 = "136"
28 SERV_5 = "160"
```

```

29 SERV_6 = "192"
30 SERV_7 = "224"
31
32 # IDT_OPT
33 # constant
34 c_idt = "-C"
35 #poisson
36 p_idt = "-O"
37 #esponential
38 e_idt ="-E"
39
40 # PS_OPT
41 # constant
42 c_ps = "-c"
43 # poisson
44 p_ps = "-o"
45 # esponential
46 e_ps ="-e"
47
48 # default value for protocol and packet size
49 DEFAULT_P = "UDP"
50 DEFAULT_PS = "512"
51
52 # Host Ip
53 dst = "169.254.207.100"
54
55 # Type of distribution
56 choice_i = c_idt
57 choice_s = c_ps
58
59
60 # Cmd creation
61 def createCmd(src , dst , tos , nPkts , avg , protocol=DEFAULT_P, ps_dim=
62 DEFAULT_PS):
63     com = 'ITGManager ' + src + ' -a ' + dst + ' -b ' + tos + ' ' +
64     choice_i + ' ' + str(avg) + ' ' + choice_s + ' ' + ps_dim + ' -t ' +
65     str(TIME_MS-8000)
66     return com
67
68 # Cmd creation
69 def createCmd_2(dst , port , tos , nPkts , avg , protocol=DEFAULT_P, ps_dim=
70 DEFAULT_PS):
71     com = 'ITGSend -a ' + dst + ' -rp ' + str(port) + ' -b ' + tos + ' ' +
72     + choice_i + ' ' + str(avg) + ' ' + choice_s + ' ' + ps_dim + ' -t ' +
73     ' + str(TIME_MS-10000)+ '&
74     return com
75
76 # Cmd creation Iperf
77 def createCmd_3(dst , port , tos , nPkts , avg , protocol=DEFAULT_P, ps_dim=
78 DEFAULT_PS):
79     com = 'iperf3 -c ' + dst + ' -p ' + str(port) + ' -S ' + tos + ' ' +
80     ' -k ' + str(nPkts) + '&
81     return com

```

B.4 Set_Queue.py

```
2 from subprocess import call
3 import threading
4 import subprocess
5 import random
6 import os
7 import time
8 import datetime
9 import json
10 import sys
11 import ditg
12
13 from Controller_commands import *
14
15 max_rate_queue=100#Mbps
16 max_rate_queue=max_rate_queue*1000000
17 Default=str(max_rate_queue *20/100)#20%
18 Premium=str(max_rate_queue *80/100)#80%
19 Gold=str(max_rate_queue *100/100)#100%
20
21 def set_queue_ether2():
22     NET = get_switchis()
23     if NET != "NO NET" and NET!="[ , ]":
24         i=1
25         while i<NET.find("]"):
26             mom_NET=NET[ i :]
27             datapath=NET[ i : i+mom_NET.find(",")]
28             i=i+mom_NET.find(",") +2
29             port_id = switch_ports_name(datapath)
30             time.sleep(0.2)
31             i=1
32             while i<NET.find("]"):
33                 mom_NET=NET[ i :]
34                 datapath=NET[ i : i+mom_NET.find(",")]
35                 i=i+mom_NET.find(",") +2
36                 port_id = switch_ports_name(datapath)
37                 ovsdb_addr(datapath)
38
39                 print(port_id)
40                 IP_Flag=True
41                 for index in range(0,len(port_id)):
42                     if port_id[index]=="eth2" or port_id[index]=="eth1":
43                         print "Port_ID: "+port_id[index]
44                         port = port_id[index][port_id[index].find("h") +1:]
45                         if port_id[index]=="eth2":
46                             set_queue(datapath, port_id[index], str(
47                             max_rate_queue), {"\\"max_rate\": \\""+Default+"\"}, {"\\"max_rate\": \
48                             "+Premium+"\"}, {"\\"min_rate\": \\""+Gold+"\""})
49                             IP_Destination="169.254.181.98"
50                             set_Telecom_queue(datapath, port, IP_Flag,
51                             IP_Destination)
```

```

50 try:
51     os.popen("sudo -S curl -X DELETE http://localhost:8080/qos/queue
52 /0000000000000002", 'w').write("Ao70pa45")
53     print "\n"
54     set_queue_eth2()
55 except:
56     print "ERROR"

```

B.5 SendTime.py

```

2 #!/usr/bin/python
3 import serial
4 import time
5 import datetime
6 import os
7
8 os.popen("sudo chmod a+rw /dev/ttyACM0", 'w').write("Ao70pa45")
9
10
11 def read_all(port, chunk_size=200):
12     """Read all characters on the serial port and return them."""
13     if not port.timeout:
14         raise TypeError('Port needs to have a timeout set!')
15
16     read_buffer = b''
17
18     while True:
19         # Read in chunks. Each chunk will wait as long as specified by
20         # timeout. Increase chunk_size to fail quicker
21         byte_chunk = port.read(size=chunk_size)
22         read_buffer += byte_chunk
23         if not len(byte_chunk) == chunk_size:
24             break
25
26     return read_buffer
27
28 #'COM3'
29 ser = serial.Serial(
30     port = '/dev/ttyACM0',
31     baudrate = 115200,
32     parity = serial.PARITY_NONE,
33     stopbits = serial.STOPBITS_ONE,
34     bytesize = serial.EIGHTBITS,
35     timeout=0.5, # IMPORTANT, can be lower or higher
36     inter_byte_timeout=0.1 # Alternative
37 )
38 time.sleep(5)
39 flag = 0;
40 first_loop=0;
41 bufsize=0;
42 while True:

```

```

43     ts = datetime.datetime.now();
44
45     sec_mom=ts.second;
46     minute_mom=ts.minute;
47     hour_mom=ts.hour;
48     DAY_mom=ts.day;
49     MONTH_mom=ts.month;
50     YEAR=str(ts.year);
51
52     if sec_mom<10:
53         sec='0'+str(ts.second);
54     else:
55         sec=str(ts.second);
56     if minute_mom<10:
57         minute='0'+str(ts.minute);
58     else:
59         minute=str(ts.minute);
60     if hour_mom<10:
61         hour='0'+str(ts.hour);
62     else:
63         hour=str(ts.hour);
64     if DAY_mom<10:
65         DAY='0'+str(ts.day);
66     else:
67         DAY=str(ts.day);
68     if MONTH_mom<10:
69         MONTH='0'+str(ts.month);
70     else:
71         MONTH=str(ts.month);
72
73     DATA=YEAR+' '+MONTH+' '+DAY+' '+hour+' '+minute+' '+sec;
74
75     if minute_mom%13==0 and sec_mom==0:
76         flag=0;
77     if flag==0:
78         if first_loop==0:
79             print("Time aligned at:")
80             print(DATA)
81             first_loop=1
82
83             ser.write(DATA.encode()) #Send data to arduino. Activate
84             arduino read pin and write to serial
85             time.sleep(2)
86             byteData = read_all(ser)
87             if byteData.decode("utf-8")=="1":
88                 flag=1;
89             else:
90                 print("ERROR-->resend data")
91                 flag=0;
92             time.sleep(1)
93         else:
94             time.sleep(0.2)

```

B.6 ReadTime.ino

```
2 #include <Wire.h>
3 #include <TimeLib.h>
4 #include <DS1307RTC.h>
5
6 const char *monthName[12] = {
7     "Jan", "Feb", "Mar", "Apr", "May", "Jun",
8     "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"
9 };
10
11 tmElements_t tm;
12
13 char inByte=0;
14 int lenbuffer=19;
15 byte bufferDATA[19];
16
17 String myString;
18 String Hour;
19 String Min;
20 String Sec;
21 String Day;
22 String Month;
23 String Year;
24
25 void setup() {
26     Serial.begin(115200);
27     pinMode(LED_BUILTIN, OUTPUT);
28     digitalWrite(LED_BUILTIN, LOW);
29     while (!Serial) {
30         ; // wait for serial port to connect. Needed for
native USB
31     }
32 }
33
34 void loop() {
35
36     String DATA;
37     if (Serial.available() > 0) //Waiting for request
38     {
39         Serial.readBytes(bufferDATA, lenbuffer);
40         myString = String((char *)bufferDATA);
41         Hour=myString.substring(11, 13);
42         Min=myString.substring(14, 16);
```

```
43     Sec=myString.substring(17, 19);
44     Day=myString.substring(8, 10);
45     Month=myString.substring(5, 7);
46     Year=myString.substring(0, 4);
47
48     tm.Hour = Hour.toInt();
49     tm.Minute = Min.toInt();
50     tm.Second = Sec.toInt();
51     tm.Day = Day.toInt();
52     tm.Month = Month.toInt();
53     tm.Year = CalendarYrToTm(Year.toInt());
54     RTC.write(tm);
55     Serial.print("1");
56     delay(1000);
57 }
58
59 }
```