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/04

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 paper presents a novel methodology to learn, starting from historical data and appropriately combining autoregressive exogenous(ARX) identification with Regression Trees and Random Forests, 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. Mininet network emulator environment has been used to validate the prediction accuracy of the calculated predictive models, as well as 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).

Contents

List of Figures

List of Tables

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). A Knowledge Plane (KP) approach [3] has been proposed to enable automation, recommendation and intelligence by applying machine learning 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 [4].

The applications of machine learning in networks is become crucial for future developments. Patcha and Park [5] have given a detailed description of machine learning techniques in the domain of intrusion detection. Nguyen and Armitage [6] focus on IP traffic classification. Bkassiny et al. [7] have studied learning problems in Cognitive Radio Networks, and surveyed existing machine learning based methods to address them. How machine learning techniques can be applied in wireless sensor networks has been investigated in [8]. Wang et al. [9] have presented the state-of-theart Artificial Intelligence based techniques applied to evolve the heterogeneous networks, and discussed future research challenges. Buczak and Guven [10] have researched on data mining methods for cyber security intrusion detection. Klaine et al. [11] have surveyed the machine learning algorithms solutions in self organizing cellular networks. How to improve network traffic control by using machine learning techniques has been studied in [12]. Similar to [5], Hodo et al. [13] also focus on machine learning based Intrusion Detection System. Zhou et al. [14] focus on using cognitive radio technology with machine learning techniques to enhance spectrum utilization and energy efficiency of wireless networks. Chen et al. [15] have studied the neural networks solutions applied in wireless networks such as communication, virtual reality and edge caching. Usama et al. [16] have applied unsupervised learning techniques in the domain of networking. Although machine learning techniques have been applied in various domains, no existing works focus on the applications of machine learning in the domain of Software Defined Network (SDN).

Thanks to the recently introduced SDN paradigm [17, 18, 19, 20, 21] 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 owns 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 [22]. 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 [23, 16]. Data is the key to the data-driven machine learning algorithms. The centralized SDN controller has a global network view, and is able to collect various network data. Based on the real-time and historical network data, machine learning techniques can bring intelligence to the SDN controller by performing data analysis, network optimization, and automated provision of network services. The programmability of SDN enables that the optimal network solutions (e.g., configuration and resource allocation) made by machine learning algorithms can be executed on the network in real time.

More in detail, a SDN controller device can configure the forwarding state of each switch by using a standard protocol called OpenFlow (OF) [24]. Thanks to the OF *counter variables* (e.g. flow statistics, port statistics, queue statistics, etc.), the controller can retrive information (feedback) from the network devices and store/process them for optimization purposes [25]. A SDN controller can supervise many aspects of traffic flow, as segment routing and queue management on switch ports. In [26] 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.

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. On this line of research, Cello *et al.* provide in [27] 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 [28] an initial effort is conducted to derive a general hybrid systems framework to model the flow of traffic in communication networks. In [29] 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 [30].

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 [1, 2, 31, 32, 33, 34, 35]. In this scenario, computing technologies such as graphic processing and tensor processing units represent a good opportunity to implement advanced control theoretic (e.g. Model Predictive Control - MPC) and machine learning algorithms (e.g. decision trees, deep neural networks, etc.) in the communication networks [23, 16, 36, 37]. 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.

The goal of this thesis is to address this challenge exploiting control theory combined with Machine Learning techniques. Queues bandwidth control must rely on an accurate model for predicting queues state: 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 and random forests algorithms [38, 39, 40], has been presented as the main contribution of this work. 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) 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.

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

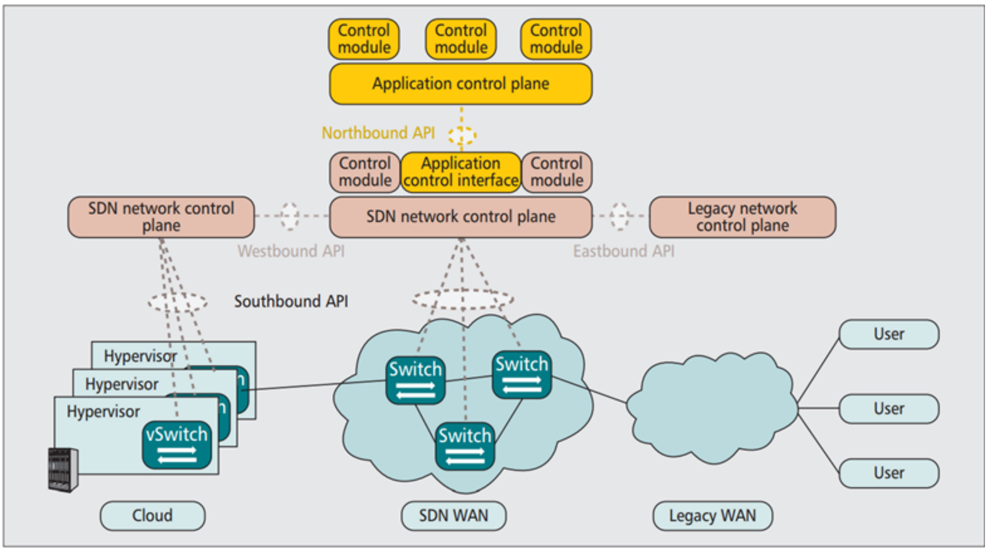


Figure 1.1: The high-level SDN architecture.

Chapter 1

Background Knowledge

1.1 Software Defined Networks Architecture

The Open Networking Foundation (ONF) [41] 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" [17]. A SDN architecture is presented 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 will give a brief description of these planes and their interactions.

Data Plane : The data plane, or infrastructure plane, is the lowest layer in SDN architecture. This plane is composed by physical switches and virtual switches and others forwarding devices. Virtual switches are software-based switches, which can run on common operating systems such. Open vSwitch [42], Indigo [43] and Pantou [44] are three implementations of virtual switches. Physical switches are hardware-based switches. They can be implemented on open network hardware (e.g., NetFPGA [45]) 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 [46] and ServerSwitch [47] 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).

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 responsable 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 [48], OpenDayLight, [49] NOX [50], POX [51], Floodlight [52] and Beacon [53]. Three communication interfaces allow the controllers to interact: southbound, northbound 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 [24] promoted by ONF is the first and the most popular open standard SBI. There exist other less popular proposals such as OVSDB [54], Protocol-Oblivious

Forwarding (POF) [55] and OpenState [56]. With NBIs, automation, innovation and management of SDN networks has been facilitatethanks 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 the large-scale networks are always partitioned into several domains and 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 [57] and HyperFlow [58] 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 [59], East-West Bridge [60] and Communication Interface for Distributed Control plane (CIDC) [61] have been proposed as eastbound/westbound interfaces to exchange network information. However, the eastbound/westbound interfaces have not yet been standardized.

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. [62] have discussed the impact of SDN on Traffic Engineering (TE) and surveyed the SDN-based TE solutions. Security in SDN has been surveyed in [63, 64, 65, 66, 67, 68]. Especially, Yan et al. [67] 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 [69], 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. [70] 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 [71], optical networks [72], wireless networks [73, 20], Internet of Things (IoT) [74], edge computing [75], Wide Area Networks (WAN) [76], cloud computing [77], Network Function Virtualization (NFV) [78, 79].

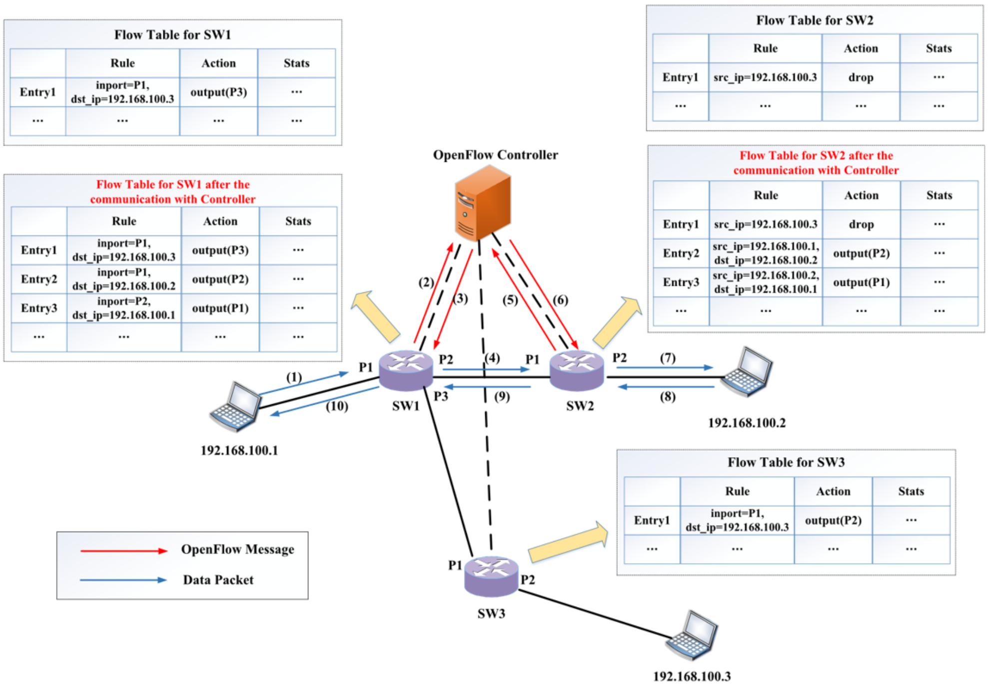


Figure 1.2: Example of OpenFlow-based SDN network.

For more informations on SDN, please refer to [80, 81, 82, 83, 84, 85, 86, 87].

1.1.1 Workflow

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 [25]. 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 [53]. 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 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. This new methods allows the system to learn useful structural patterns and models from training data. A machine learning approach consists of two main phases: training phase and decision making phase. At the training phase, after a data mining period of system input/output information, machine learning methods are applied to learn the system model using the training dataset. At the decision making phase, the system can obtain the estimated output for each new input by using the trained model. 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 and its classical concepts, please refer to [88, 89, 90].

A. 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 [91, 92]. In the following, we will give a detailed representation of supervised learning algorithms.

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 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 Chebyshev, City-block, Euclidean and Euclidean squared [93].

2) Decision Tree (DT): The DT 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

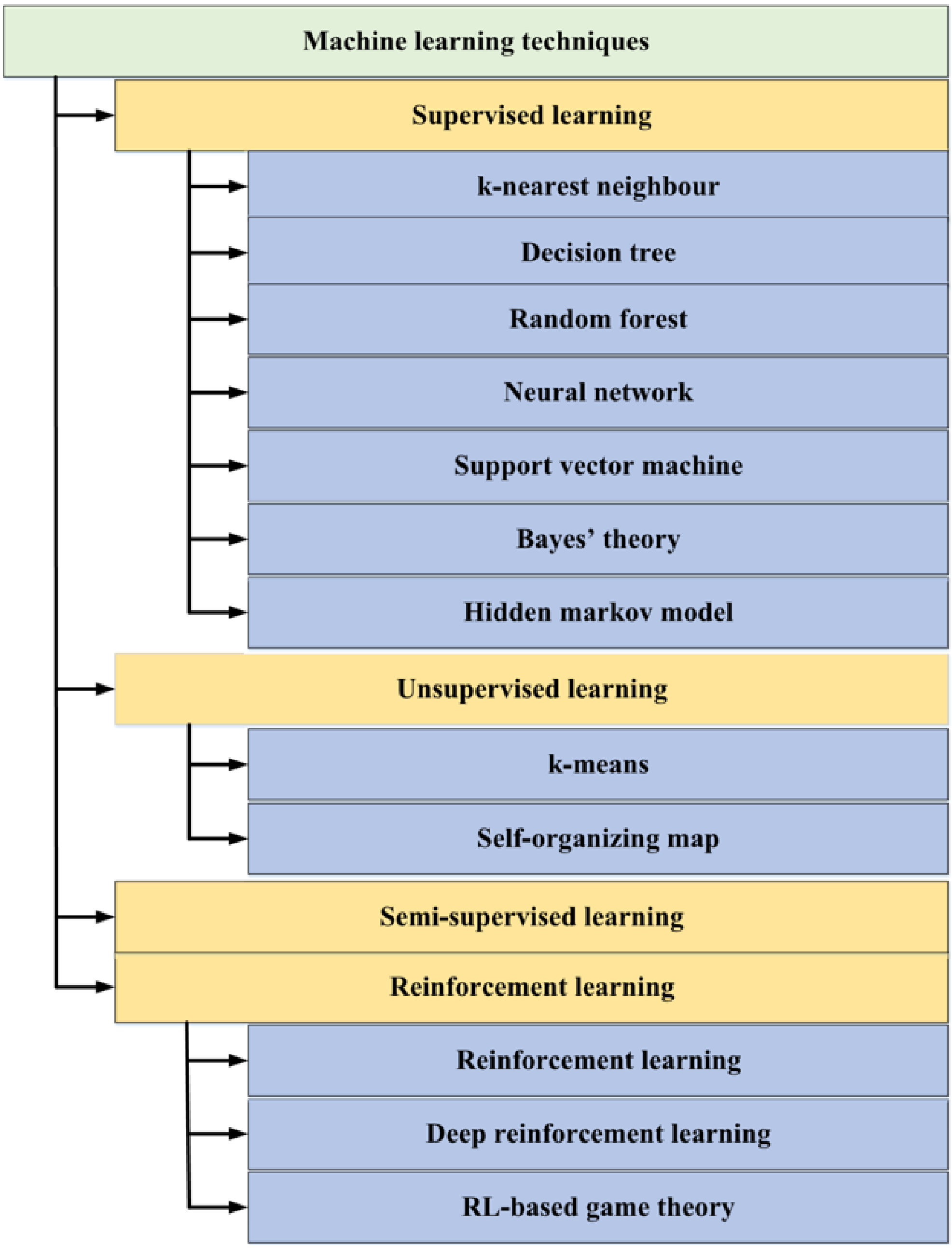


Figure 1.3: Common machine learning algorithms.

label. The unlabeled sample can be classified by comparing its feature values with the nodes of the decision tree [94]. The DT has many advantages, such as intuitive knowledge expression, simple implementation and high classification accuracy. ID3 [95], C4.5 [96] and CART [97] 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 (RF): A RF [98] consists of many DT. To mitigate overfitting of DT method and improve accuracy, the random forest method randomly chooses only a subset of the feature space to construct each DT. The steps to classify a new data sample by using random forest method are:

a) put the data sample to each tree in the forest;

(b) Each tree gives a classification result, which is the tree's "vote";

(c) The data sample will be classified into the class which has the most 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 [99]. Each neuron perform 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 and 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., $X_1, X_2, ..., X_m$) and the input can be assigned to n possible classes (i.e., $Y_1, Y_2, ..., Y_n$). Also, W_{ij}^1 denotes the variable link weight between the ith neuron of layer l and the jth neuron of layer $l + 1$, and ak^l denotes the activation function of the kth neuron in layer l. There are many types of neural networks, which can be divided in supervised or unsupervised main group [100]. 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 which exchange spiking signals. The main difference between random NN and other neural networks is that neurons in random NN exchange spiking signals probabilistically. In random NN, the internal

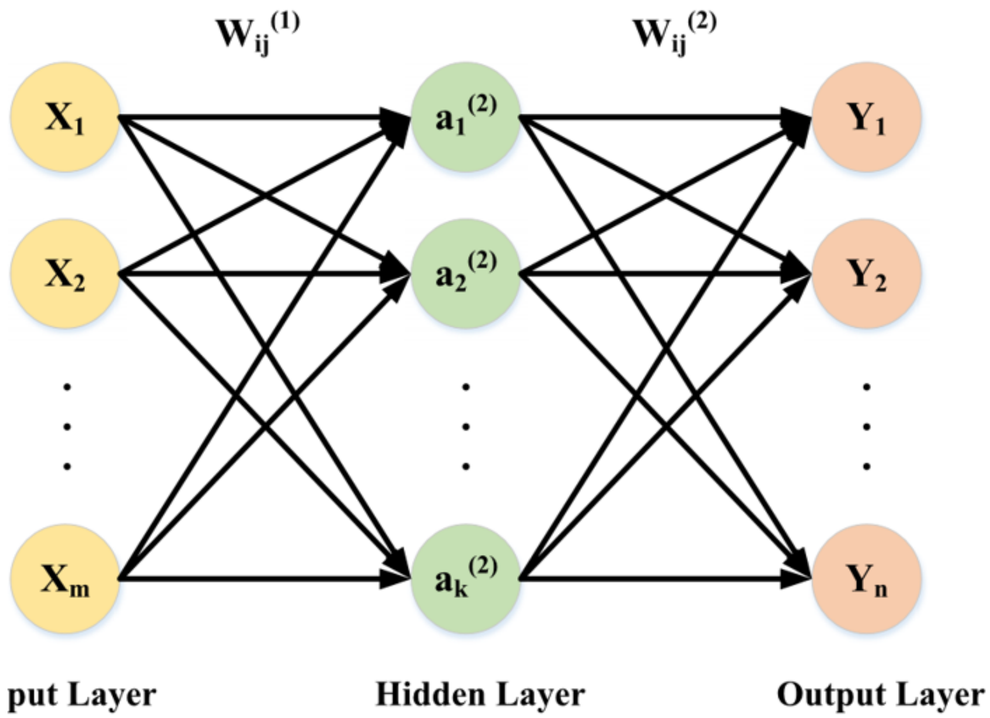


Figure 1.4: A basic neural network with three layers: an input layer, a hidden layer and an output layer.

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 neurondependent 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 [101].

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 [102, 103]. 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 [104]. In deep NNs, each layer's neurons train a feature representation based on the previous 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 [104].

c) Convolutional NN: Convolutional NN and recurrent NN are two major types of deep NNs. Convolutional NN [105, 106] 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) [107] 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 [108], 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 the Bayes' theory is that it requires a relatively small number of training dataset to learn the probability model [109]. 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 [110].

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 [111, 112].

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.

B. Unsupervised Learning In contrast to supervised learning, an unsupervised learning algorithm is given a set of inputs without labels, thus there is no output. Basically, 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 algorithm 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) [113], 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).

C. 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 hold in semisupervised learning, such as smoothness assumption, cluster assumption, low-density separation assumption, and manifold assumption. Pseudo Labeling [114] 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.

D. Reinforcement Learning

1) Reinforcement Learning (RL): RL [115, 116] 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 $st+1$. 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 $A(\pi : S \longrightarrow A)$ 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.

In summary, 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.

1.3 Switched affine modeling via RT and RF

Problem formulation. In this section it is illustrated the methodology to apply the results proposed in [117, 118] to identify, starting from a set of collected historical data $\mathcal{D} = \{y(k), u(k), d(k)\}_{k=0}^{\ell}$ as illustrated in the previous section, a switching ARX model of input-output behavior of the traffic flow in a switch of a SDN network as follows:

$$x(k+j+1) = A'_{\sigma_j(x(k),d(k))}x(k) + \sum_{\alpha=0}^{j} B'_{\sigma_j(x(k),d(k)),\alpha}u(k+\alpha) + f'_{\sigma_j(x(k),d(k))}, \qquad (1.1)$$

$j = 0, \dots, N-1$, 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 $\sigma_j : \mathbb{R}^{n_x+10} \to \mathcal{M} \subset \mathbb{N}$ 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 (1.1) to setup the following problem, which can be solved using standard Quadratic Programming (QP) solvers:

Problem 1

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

As it is well known [119], 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.

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 [120] 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) $R_1, R_2, \cdots, R_\nu$. In each partition $y(k)$ is estimated with a different constant $\hat{y}_i\, i = 1, \ldots, \nu$, given by the average of the samples of $y(k)$ falling in R_i, i.e.

$$\hat{y}_i = \frac{\sum\limits_{\{k|(x_1(k),\ldots,x_\eta(k))\in R_i\}} y(k)}{|R_i|} \tag{1.2}$$

Random Forests [121] 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 refered to [120, 121].

Switching ARX (SARX) model identification via RT. To derive a model as in (1.1), 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 $\{\mathcal{T}_{\iota,j}\},\ \iota = 1, 2, 3,\ j = 0, \ldots, N-1$, 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 (1.2). 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 (1.7). 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 [117, 118] 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

$$x_\iota(k+j+1) = A'_{\iota,i_j} x(k) + \sum_{\alpha=0}^{j} B'_{\iota,i_j,\alpha} u(k+\alpha) + f'_{\iota,i_j}, \tag{1.3}$$

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

$$
B'_{\iota,i_j,\alpha} = \begin{bmatrix} b_{1,\alpha} & b_{2,\alpha} & b_{3,\alpha} \\ 0 & 0 & 0 \\ \vdots & \vdots & \vdots \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ \vdots & \vdots & \vdots \\ 0 & 0 & 0 \end{bmatrix},\ \alpha < j,\ B'_{\iota,i_j,j} = \begin{bmatrix} b_{1,\alpha} & b_{2,\alpha} & b_{3,\alpha} \\ 0 & 0 & 0 \\ \vdots & \vdots & \vdots \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ \vdots & \vdots & \vdots \\ 0 & 0 & 0 \end{bmatrix},\ f'_{\iota,i_j} = \begin{bmatrix} f \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \tag{1.4}
$$

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

$$
\begin{aligned}
&\underset{\xi_{\iota,i_j}}{\textit{minimize}} \quad \| \Lambda_{\iota,i_j}\xi_{\iota,i_j} - \lambda_{\iota,i_j} \|_2^2 \\
&\textit{subject to } f_{\min} \le f \le f_{\max} \\
&\qquad\qquad a_{\min} \le a_\jmath \le a_{\max} \\
&\qquad\qquad b_{\min} \le b_{\iota,\alpha} \le b_{\max},
\end{aligned} \tag{1.5}
$$

where ξ_{ι,i_j}, λ_{ι,i_j}, and Λ_{ι,i_j} contain respectively the parameters of matrices in (1.4), the predictions $x_\iota(k+j+1)$, and the current measurements of $x(k)$ and $u(k+\alpha)$. Linear inequalities (1.5) are used to constrain elements in ξ_{ι,i_j} to take into account physical system constraints and improve prediction accuracy. Model (1.3) can be easily compacted in the following form taking into account all the states $\iota = 1, 2, 3$:

$$
x(k+j+1) = A'_{i_j}x(k) + \sum_{\alpha=0}^{j} B'_{i_j,\alpha}u(k+\alpha) + f'_{i_j}. \tag{1.6}
$$

In particular, with the specific choice of $\delta_u = 0$ model (1.1) can be rewritten in the following state-space formulation

$$
x(k+j+1) = A_{\sigma_j(x(k),\mathbf{u}^-(k),d(k))}x(k+j) + B_{\sigma_j(x(k),\mathbf{u}^-(k),d(k))}u(k+j) + f_{\sigma_j(x(k),\mathbf{u}^-(k),d(k))}, \tag{1.7}
$$

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 (1.6) is equivalent to model (1.7) for any initial condition, any switching signal and any control sequence.

Proposition 1 *[118] 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 [118], 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 paper 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.

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 (1.4) 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 (1.9) can be obtained by averaging parameters a_*, b_* and f, $\forall \tau = 1,\dots,|\mathcal{F}_{\iota,j}|$, i.e.

$$\tilde{\xi}_{\iota,j} = \frac{\sum_{\tau=1}^{|\mathcal{F}_{\iota,j}|} \tilde{\xi}_{\iota,j,i_\tau}}{|\mathcal{F}_{\iota,j}|}, \tag{1.8}$$

over all the trees of forest $\mathcal{F}_{\iota,j}$. At this point, starting from (1.3), it can be easily construct the following model, as in (1.6) to be used in the MPC formulation by combining for $\iota = 1,2,3$ the matrices in (1.4) coming either from the RTs or from the RFs:

$$x(k+j+1) = A'_{i_j}x(k) + \sum_{\alpha=0}^{j} B'_{i_j,\alpha}u(k+\alpha) + f'_{i_j}. \tag{1.9}$$

MPC problem formulation. It is used model (1.9) to formalize Problem 1 according to the SDN priority queueing problem:

Problem 3

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

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 $k + j$; $\Delta u_\iota^{\min}$ and $\Delta u_\iota^{\max}$ 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; $u_\iota^{\min}$ and $u_\iota^{\max}$ 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 (1.9) 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. $u(k) = u_0^*$.

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 [117, 118] 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 (1.7) 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.

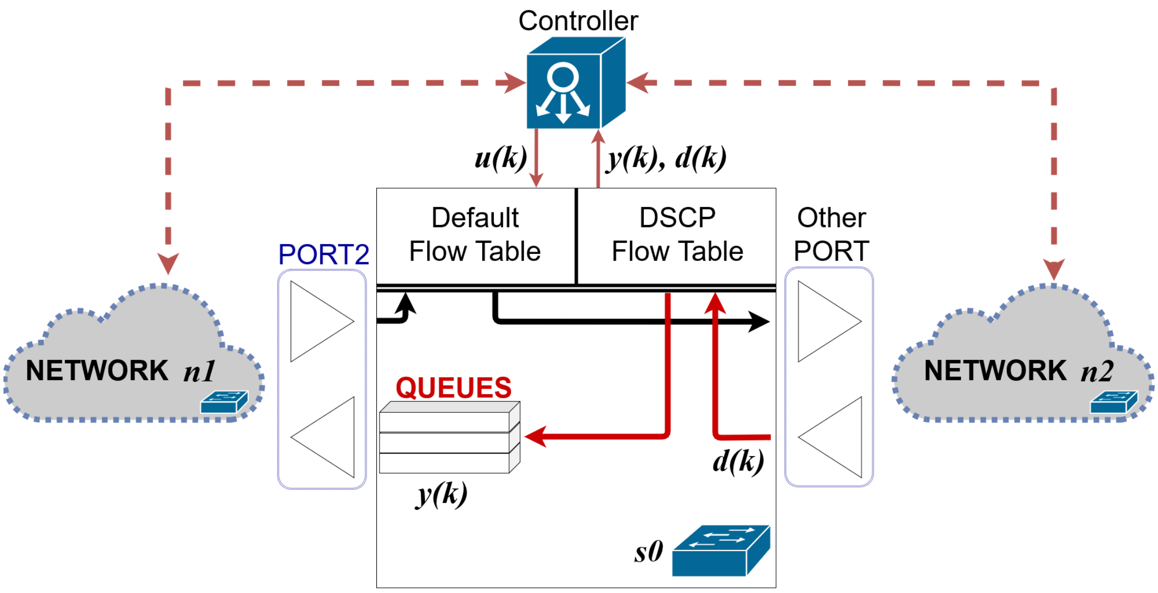


Figure 2.1: Mininet emulated network architecture.

Chapter 2

Network emulation environment

2.1 Mininet Setup

The Mininet environment [122] 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 [123, 124, 125].

We consider a network architecture as in Figure 2.1, which aims at representing a portion of a larger network where a bottleneck occurs. More precisely, we consider a switch $s0$ with one input port and one output port, and a remote controller [42, 48] that dynamically manages the configuration of the queues of $s0$. The input of $s0$ 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 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 [126, 127]. 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. [128]), 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 n1 and n2. 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 OpenFlow, Ryu supports fully 1.0, 1.2, 1.3, 1.4, 1.5 and 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 [129, 130]. 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 $d_1(k), \ldots, d_8(k)$ consist of the number of packets incoming in the switch $s0$ 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 $s0$ 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.

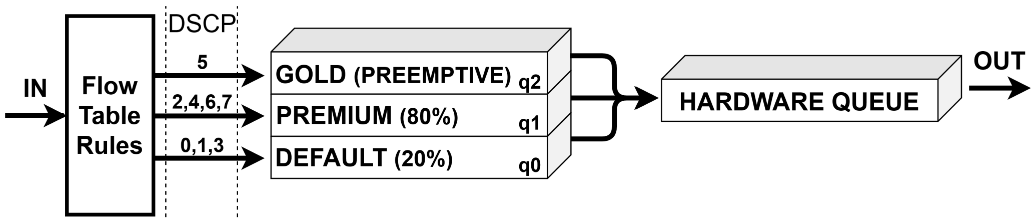


Figure 2.2: Static queues rate with routed packets relative to DSCP.

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* [128], which is depicted in Figure 2.2. To this aim we defined 3 queues in $s0$ 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 $u_1(k) = 20$*MB/s*$, \forall k$; the packets with values 2, 4, 6 and 7 (Premium queue) are routed on queue 1, with maximum rate $u_2(k) = 80$*MB/s*$, \forall k$; the packets with value 5 (Gold queue) are routed on queue 2, with minimum rate $u_3(k) = 100$*MB/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.

As we will see in Section 2.2, 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 the following section.

2.2 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 $d_1(k), \ldots, d_8(k)$, 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 1.3 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 it will be illustrated the effect of iterative dataset updates in the prediction accuracy, both with and without prediction of the future disturbances. Finally it will be used the proposed predictive models 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 thought 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 [131].

2.2.1 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.

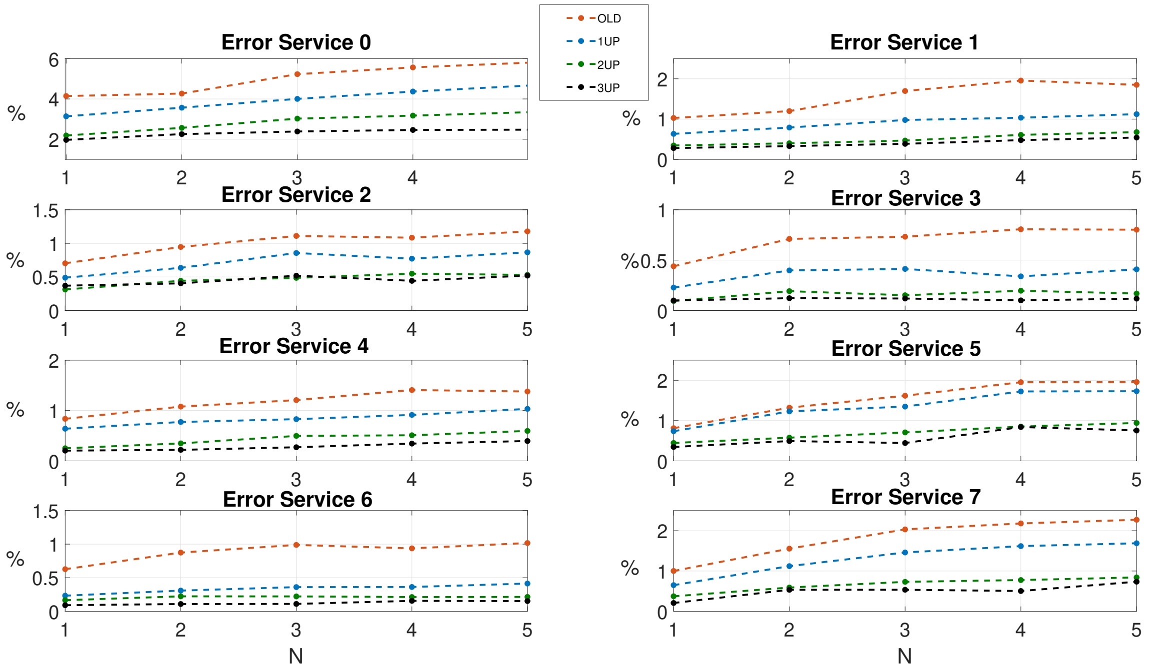


Figure 2.3: NRMSE of the disturbance predictive model over a time horizon of $N = 5$.

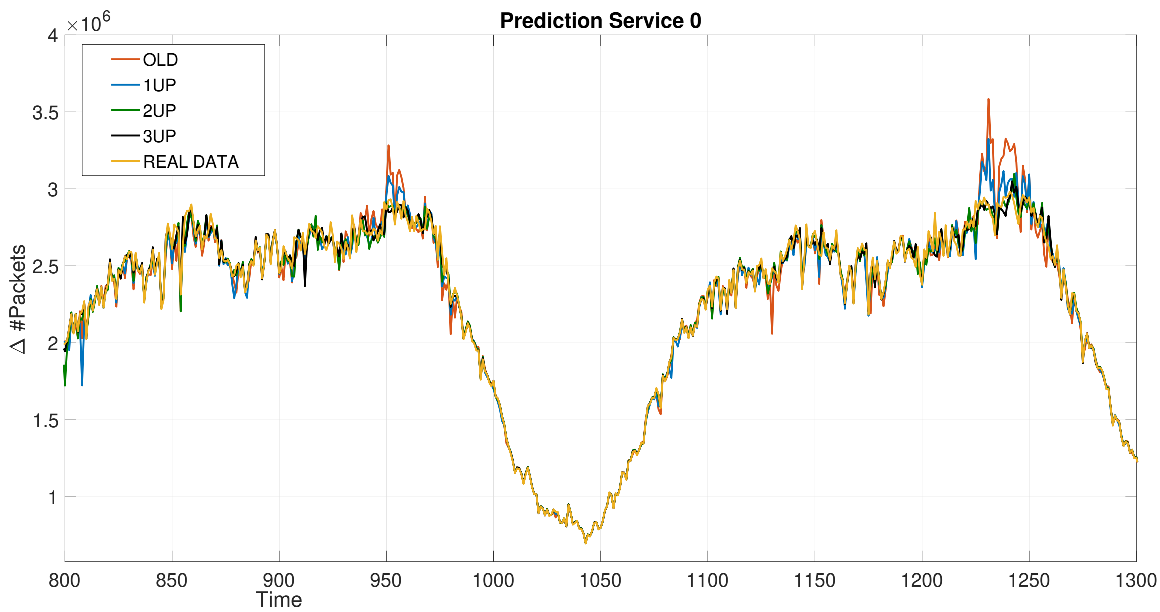


Figure 2.4: Comparison between the real traffic (YELLOW LINE) and the traffic prediction for the different models for Service 0.

2.2.2 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 [132] and references therein for more details. A wide number of tools to build Neural Networks have been developed during recent years, e.g. [133, 134, 135] 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 paper, 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 1.3 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
$\lvert\mathcal{F}_{ij}\rvert$	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 depper 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 [136], 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*).

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 pri-

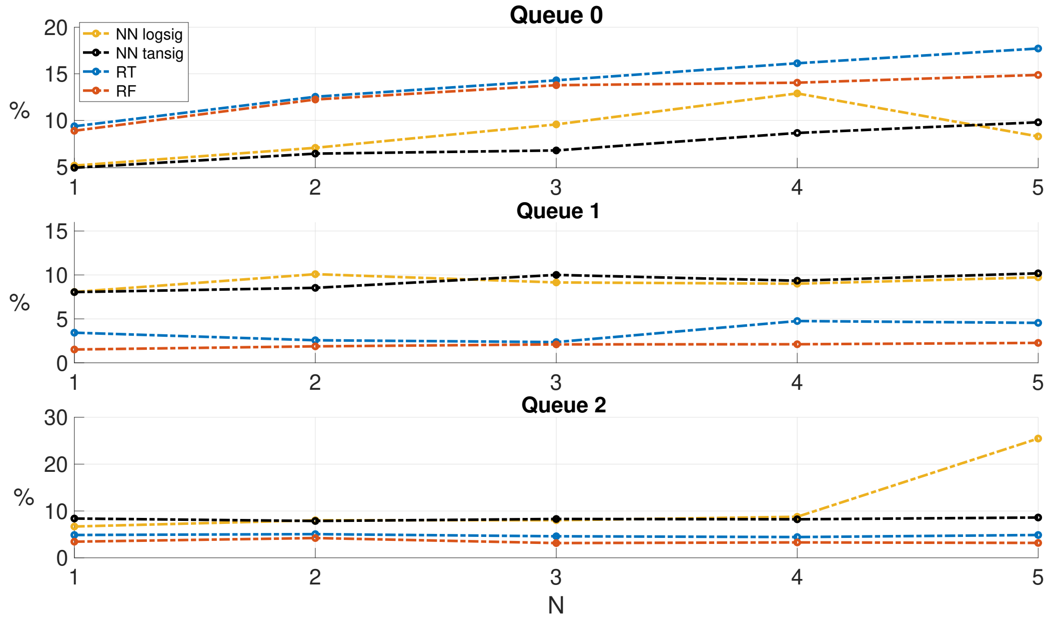


Figure 2.5: NRMSE, up to $N = 5$ and for each priority class, for RT (blue), RF (red), NN with sigmoids as activation function (yellow) and NN with hyperbolic tangent as activation function (black).

ority, 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 thought 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, we noticed a peculiar regular grid pattern that can be very well approximated 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, even thought 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 weitghts 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 2.6 and Figure 2.7 plot the NRMSEs respectively without and with knowledge of the future disturbances. The assumption of future disturbance forecast, as expected,

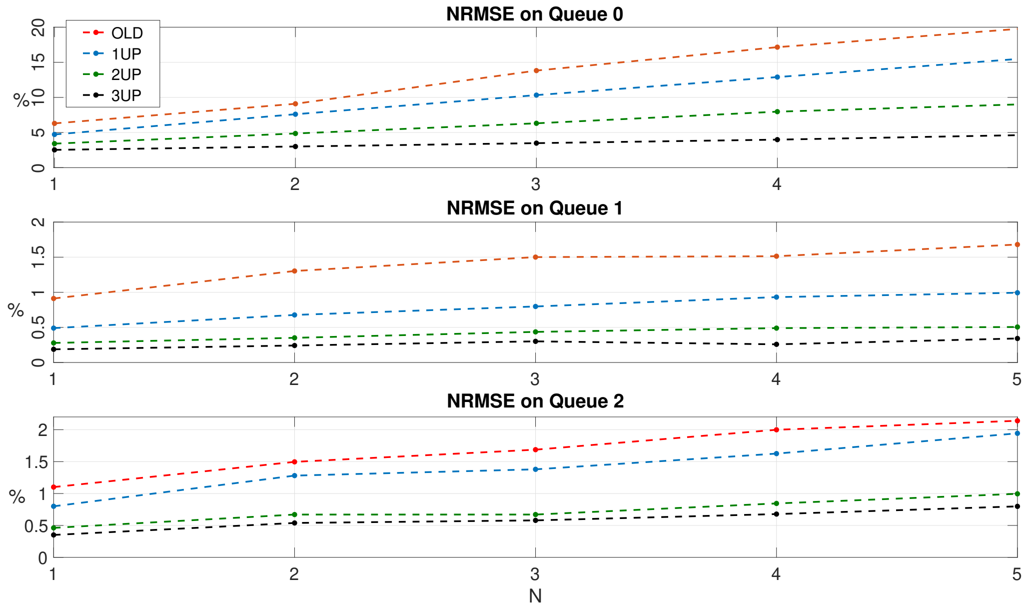


Figure 2.6: NRMSE of the queues output predictive model over a time horizon of $N = 5$, without knowledge of the future disturbances

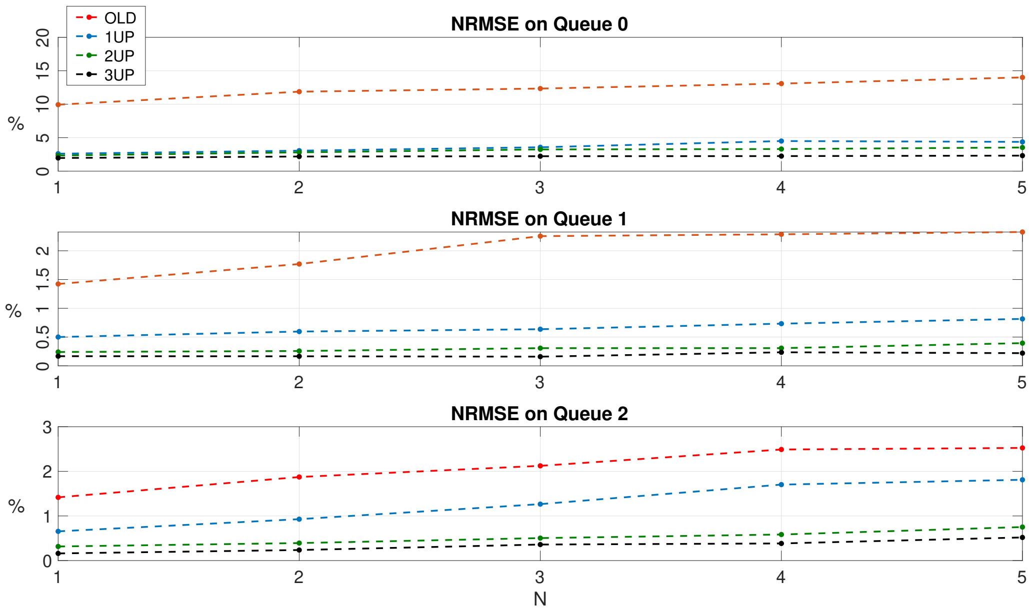


Figure 2.7: NRMSE of the queues output predictive model over a time horizon of $N = 5$, with knowledge of the 4-steps future disturbances

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 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.*

2.2.3 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 $Q = diag(1, 10^4, 10)$ 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
$\Delta u_1^{\min}$	1	$\Delta u_1^{\max}$	30
$\Delta u_2^{\min}$	20	$\Delta u_2^{\max}$	30
$\Delta u_3^{\min}$	20	$\Delta u_3^{\max}$	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.

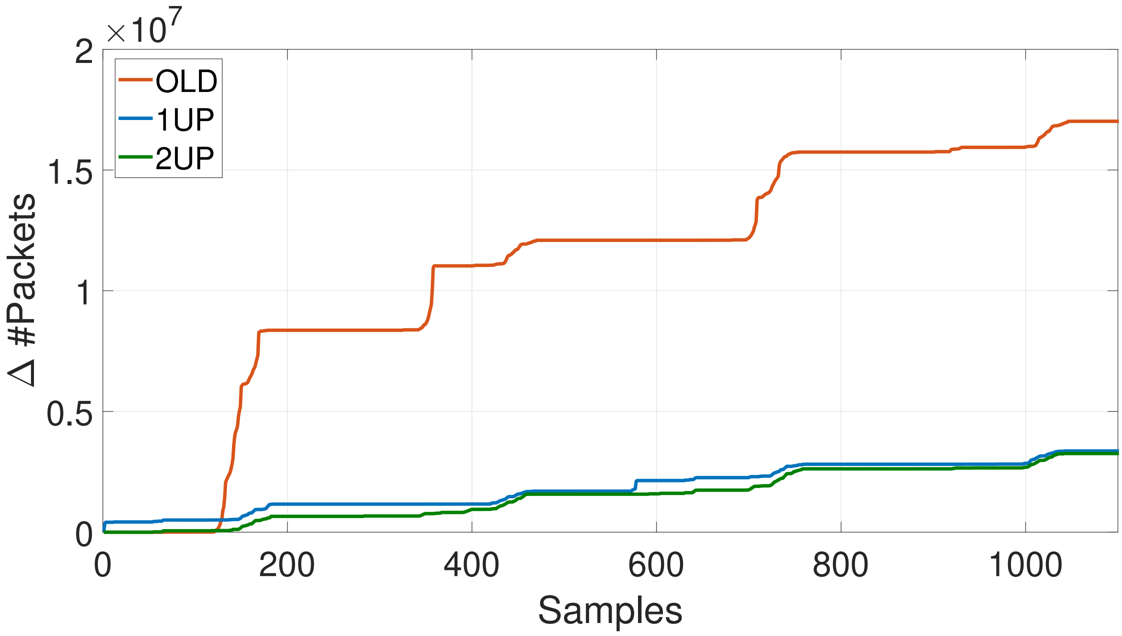


Figure 2.8: Cumulative Packet Losses without knowledge of the future disturbance.

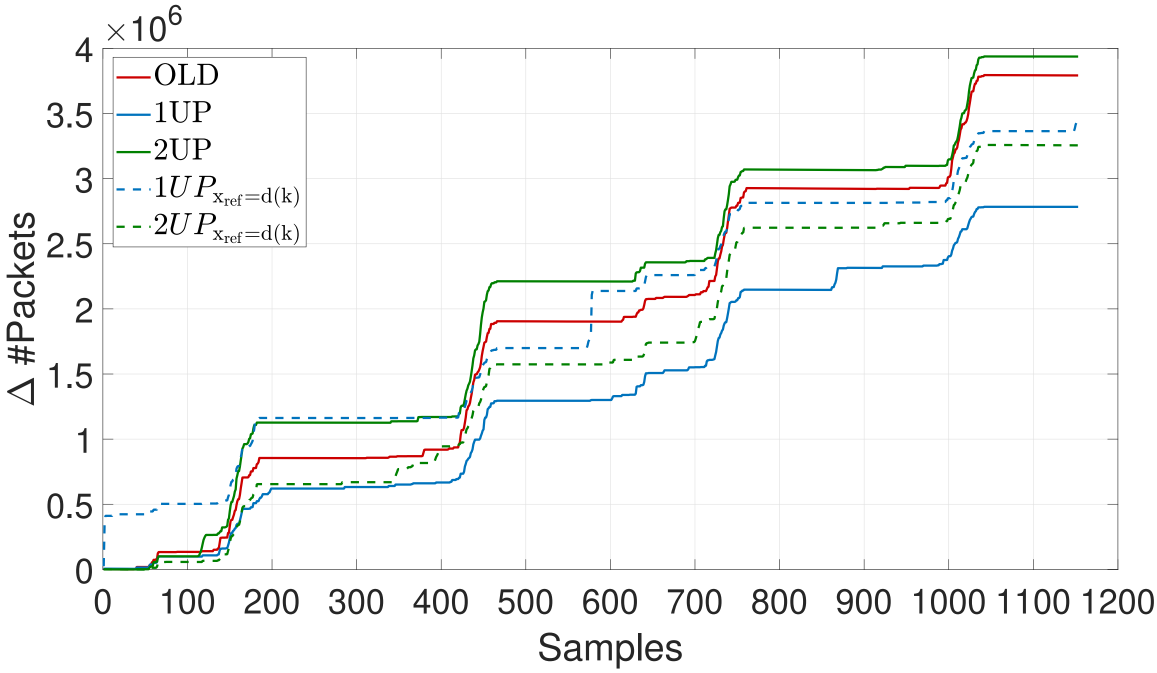
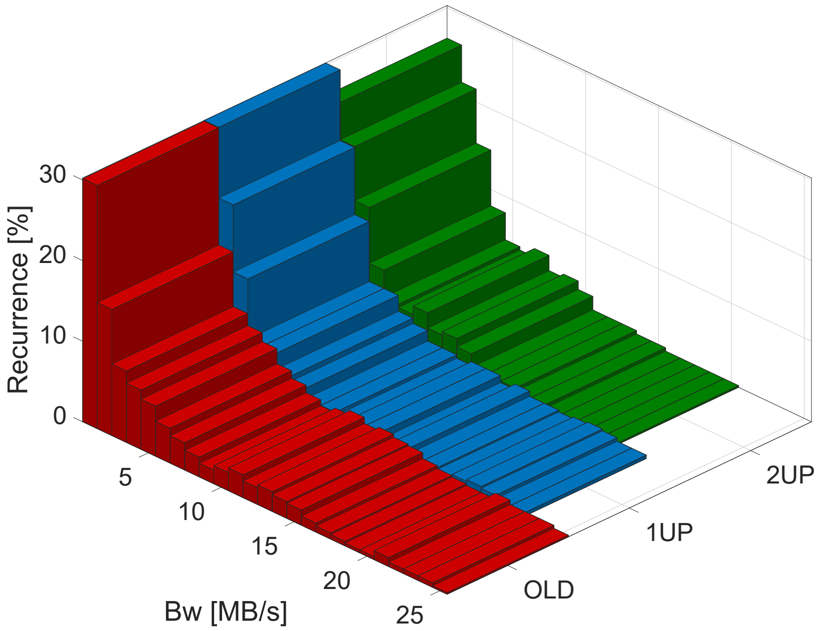
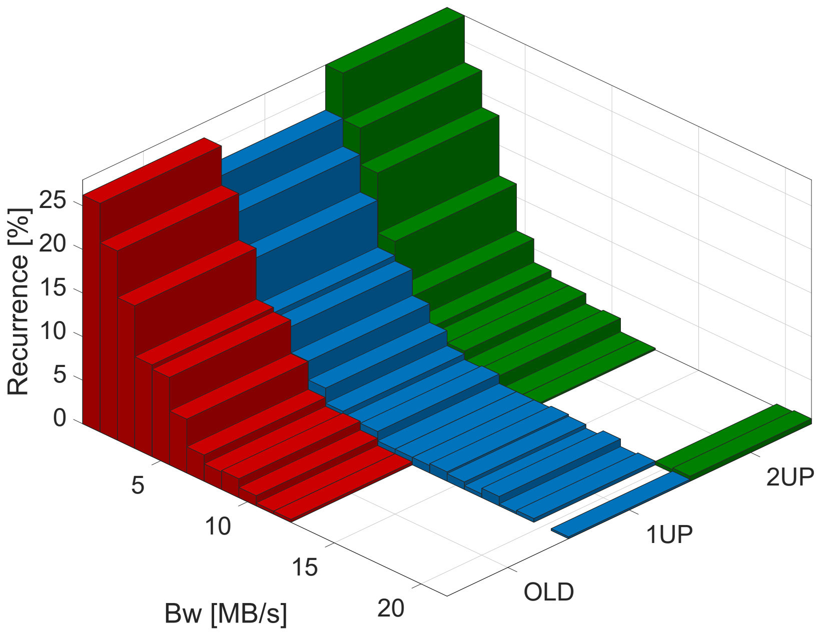


Figure 2.9: Comparison between Cumulative Packet Losses with (solid lines) and without (dashed lines) knowledge of the future disturbance.

Figures 2.8 and 2.9 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 [128]). 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.



(a)



(b)

Figure 2.10: Bandwidth saving comparison without (a) and with (b) knowledge of the future disturbances.

Figure 2.10 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.

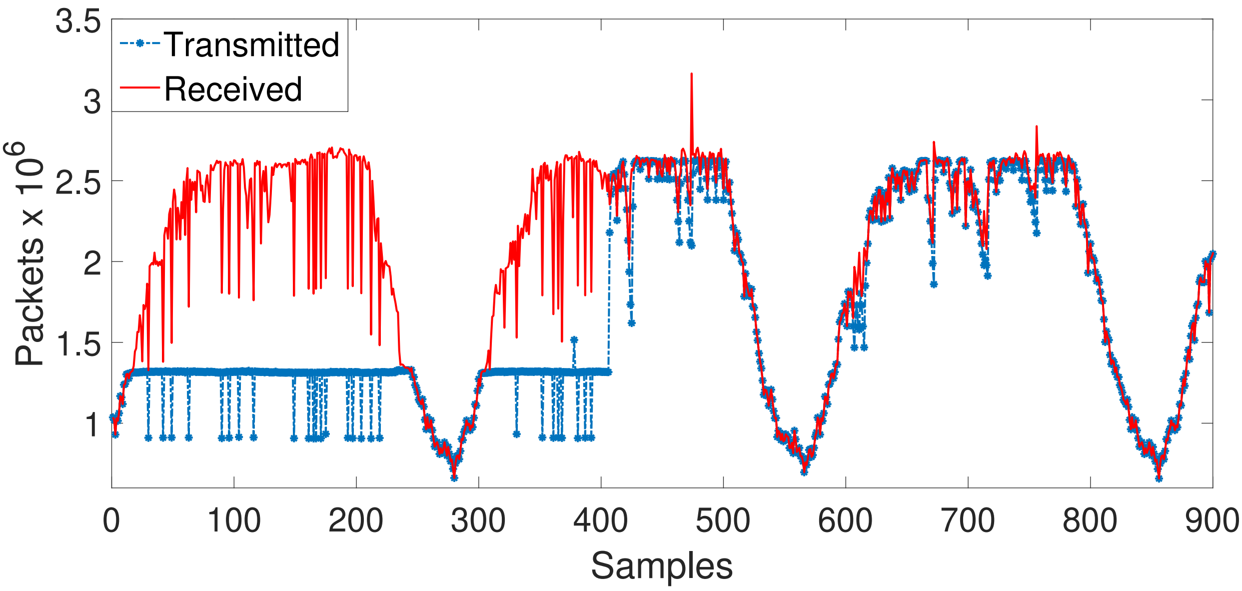


Figure 2.11: Static controller up to the $400th$, then MPC controller.

We conclude this paper 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 [128]. Figure 2.11 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 [128], 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 thought 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.

Chapter 3

Modeling Real Networks

3.1 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, the accuracy has been also tested on data measured from a real network device (Ubiquiti EP-16) of an Italian internet internet provider (Sonicatel S.r.l.). Data collection has been performed using the software Cacti [137].
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 validation in the following sections is not based on this real traffic dataset.
About data analysis, 53 days of data measurements have been used for RF training and about 3 days for model validation. Figure 3.1 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.

3.2 Control performance validation over dedicated hardware network

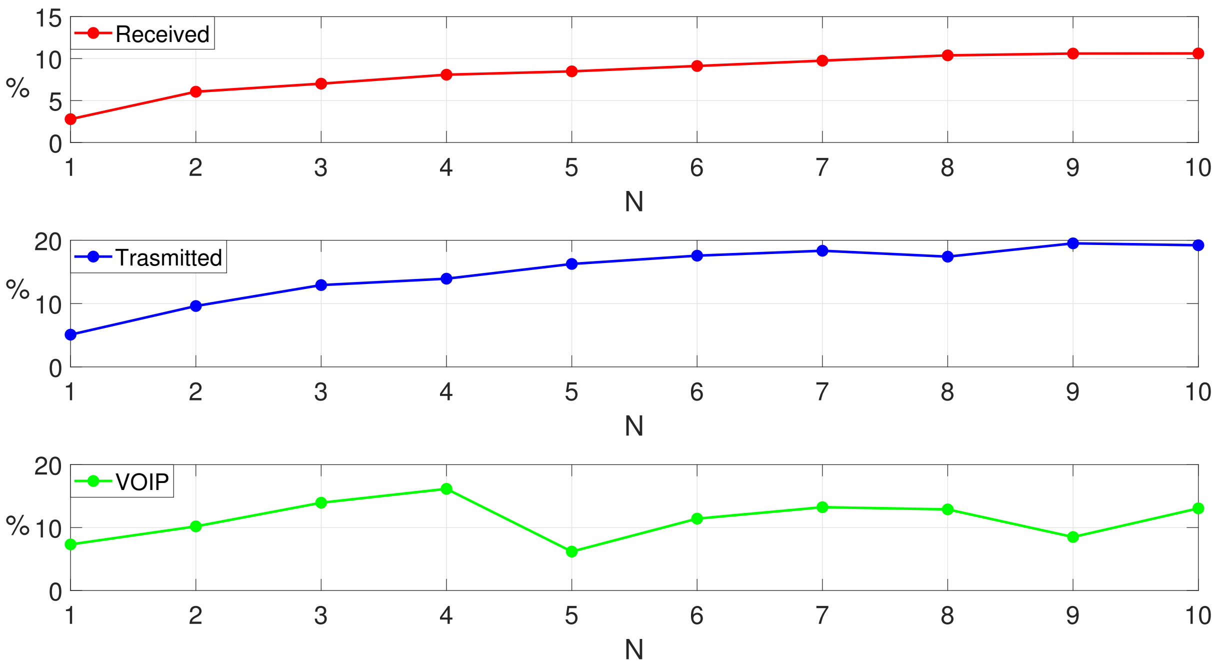


Figure 3.1: NRMSE of the packets predictive model over a time horizon of $N = 10$.

Conclusion

In this paper 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. The obtained simulative 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 in real networks. In future work it has been planned to validate these results 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. 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.

[4] 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.

[5] A. Patcha, J.-M. Park, An overview of anomaly detection techniques: Existing solutions and latest technological trends, Computer networks 51 (12) (2007) 3448–3470.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] E. Hodo, X. Bellekens, A. Hamilton, C. Tachtatzis, R. Atkinson, Shallow and deep networks intrusion detection system: A taxonomy and surveyarXiv:1701.02145v1.

[14] X. Zhou, M. Sun, G. Y. Li, B.-H. Juang, Intelligent wireless communications enabled by cognitive radio and machine learningarXiv:1710.11240v4.

[15] M. Chen, U. Challita, W. Saad, C. Yin, M. Debbah, Artificial neural networks-based machine learning for wireless networks: A tutorialarXiv:1710.02913v2.

[16] 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).

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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

[26] 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.

[27] 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.

[28] J. Lee, S. Bohacek, J. Hespanha, K. Obraczka, Modeling communication networks with hybrid systems, IEEE/ACM Transactions on Networking 15 (3) (2007) 630–643.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] 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 Radioelecrtronics, Telecommunications and Computer Engineering (TCSET), IEEE, 2018, pp. 1035–1038.

[34] 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.

[35] 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.

[36] 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.

[37] G. Xu, Y. Mu, J. Liu, Inclusion of artificial intelligence in communication networks and services (Jan. 2018).

[38] 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.

[39] 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.

[40] 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.

[41] Open networking foundation.
URL https://www.opennetworking.org/

[42] Open vSwitch, 2019.
URL https://www.openvswitch.org/

[43] Indigo: Open source openflow switches.
URL https://github.com/floodlight/indigo

[44] Pantou: Openflow 1.3 for open wrt.
URL https://github.com/CPqD/ofsoftswitch13/wiki/OpenFlow1.3-for-OpenWRT

[45] J. W. Lockwood, N. McKeown, G. Watson, G. Gibb, P. Hartke, J. Naous, R. Raghuraman, 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.

[46] 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

[47] G. Lu, C. Guo, Y. Li, Z. Zhou, T. Yuan, H. Wu, Y. Xiong, R. Gao, Y. Zhang, ServerSwitch: A Programmable and High Performance Platform for Data Center Networks (Mar. 2011).
URL https://www.microsoft.com/en-us/research/publication/serverswitch-a-programmable-and-high-performance-plat

[48] RYU Controller, 2019.
URL https://osrg.github.io/ryu/

[49] 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.

[50] 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).

[51] Pox.
URL https://github.com/noxrepo/pox

[52] Floodlight.
URL https://github.com/floodlight/floodlight

[53] 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

[54] B. Pfaff, B. Davie, The open vswitch database management protocol (2013).
URL https://rfc-editor.org/rfc/rfc7047.txt

[55] 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

[56] 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

[57] 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).

[58] A. Tootoonchian, Y. Ganjali, Hyperflow: A distributed control plane for openflow, Proc. Enterprise Netw (2010).

[59] 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

[60] 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.

[61] 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

[62] 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.

[63] 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.

[64] 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.

[65] 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.

[66] 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.

[67] 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.

[68] 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.

[69] 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.

[70] 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.

[71] 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.

[72] 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.

[73] 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.

[74] 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.

[75] 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.

[76] 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.

[77] 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.

[78] Y. Li, M. Chen, Software-Defined Network Function Virtualization: A Survey, IEEE Access 3 (2015) 2542–2553. doi:10.1109/ACCESS.2015.2499271.

[79] 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.

[80] 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.

[81] 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.

[82] 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.

[83] 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.

[84] 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

[85] 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.

[86] 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.

[87] 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.

[88] M. Mohammed, M. B. Khan, E. B. M. Bashier, Machine Learning: Algorithms and Applications, CRC Press, 2016, google-Books-ID: X8LBDAAAQBAJ.

[89] S. Marsland, Machine Learning: An Algorithmic Perspective, Second Edition, CRC Press, 2015, google-Books-ID: y_oYCwAAQBAJ.

[90] E. Alpaydin, Introduction to Machine Learning, MIT Press, 2020, google-Books-ID: tZnSDwAAQBAJ.

[91] I. Z. SB Kotsiantis, Supervised machine learning: A review of classification techniques, Emerging Artificial Intelligence Applications in Computer Engineering (2007).

[92] J. F. Trevor Hastie, Robert Tibshirani, The Elements of Statistical Learning : Data Mining, Inference, and Prediction, 2009.

[93] T. Cover, P. Hart, Nearest neighbor pattern classification, IEEE Transactions on Information Theory 13 (1) (1967) 21–27. doi:10.1109/TIT.1967.1053964.

[94] J. Han, J. Pei, M. Kamber, Data Mining: Concepts and Techniques, Elsevier, 2011, google-Books-ID: pQws07tdpjoC.

[95] 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

[96] 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.

[97] 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

[98] L. Breiman, Random forests (1999).

[99] S. Haykin, Neural Networks: A Comprehensive Foundation.

[100] 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

[101] S. Timotheou, The random neural network: a survey, The computer journal 53 (3) (2010) 251–267.

[102] G. H. Yann LeCun, Yoshua Bengio, Deep learning, Nature (2015).

[103] 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

[104] A. D. Gaurav Pandey, Learning by stretching deep networks, Proceedings of the 31st International Conference on MachineLearning, Beijing, China (2014).

[105] 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.
URL http://papers.nips.cc/paper/4824-imagenet-classification-with-deep
pdf

[106] 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

[107] 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.

[108] V. Vapnik, An overview of statistical learning theory, IEEE Transactions on Neural Networks 10 (5) (1999) 988–999. doi:10.1109/72.788640.

[109] G. E. P. Box, G. C. Tiao, Bayesian Inference in Statistical Analysis, John Wiley & Sons, 2011, google-Books-ID: T8Askeyk1k4C.

[110] J. Bakker, Intelligent Traffic Classification for Detecting DDoS Attacks using SDN/OpenFlow (2017).
URL http://researcharchive.vuw.ac.nz/handle/10063/6645

[111] 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.

[112] 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.

[113] T. Kohonen, Self-Organizing Maps, Springer Science & Business Media, 2012.

[114] 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.

[115] R. S. Sutton, A. G. Barto, Reinforcement Learning, second edition: An Introduction, MIT Press, 2018, google-Books-ID: uWV0DwAAQBAJ.

[116] 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

[117] 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.

[118] 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).

[119] F. Borrelli, et al., Predictive control for linear and hybrid systems, Cambridge University Press, 2017.

[120] L. Breiman, Classification and regression trees, Routledge, 2017.

[121] L. Breiman, Random forests, Machine learning 45 (1) (2001) 5–32.

[122] Mininet, 2019.
URL http://mininet.org/

[123] 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.

[124] 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.

[125] 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).

[126] 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).

[127] J. Babiarz, K. Chan, F. Baker, Configuration guidelines for diffserv service classes, RFC 4594 (August 2006).

[128] A. M. Langellotti, S. Mastropietro, F. T. Moretti, A. Soldati, Notiziario Tecnico Telecom Italia, Tech. rep., Telecom Italia (2004).

[129] ryu.app.ofctl rest, 2019.
URL https://ryu.readthedocs.io/en/latest/app/ofctl_rest.html

[130] QoS Ryubook 1.0 documentation, 2019.
URL https://osrg.github.io/ryu-book/en/html/rest_qos.html

[131] UDOO x86, 2019.
URL https://www.udoo.org/

[132] 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.

[133] M. Abadi, et al., TensorFlow: Large-scale machine learning on heterogeneous systems (2015).
URL https://www.tensorflow.org/

[134] F. Chollet, et al., Keras, https://keras.io (2015).

[135] A. I. Techniques, opennn, www.opennn.net (2019).

[136] M. F. Møller, A scaled conjugate gradient algorithm for fast supervised learning, Aarhus University, Computer Science Department, 1990.

[137] 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 to Computer Networks, 2020

Appendix A

Python Codes

A.1 datapath monitor Code

```
from ryu.controller import ofp_event
from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER
from ryu.ofproto import ofproto_v1_3
from ryu.lib.packet import packet
from ryu.lib.packet import ethernet
from ryu.lib.packet import ether_types
from operator import attrgetter
import threading
import time
import datetime

import subprocess
import json
import sys

from Controller_commands import *
import numpy as np

from subprocess import call
import random
import os

now=datetime.datetime.now()
year=str(now.year)
month=str(now.month)
day=str(now.day)
date=year+"-"+month+"-"+day+"_"
computername="jedi"
file = open("/home/"+computername+"/Scrivania/RyuDatapathMonitor-master/
    DataLog/"+date+"flowstat.txt","a")
stringa="time"+"\t"+"datapath"+"\t"+"in-port"+"\t"+"eth-dst"+"\t"+"out-
    port"+"\t"+"packets"+"\t"+"bytes"+"\t"+"ip_dscp"+"\t"+"SET_QUEUE\n"
file.write(stringa)
file.close()

```

```
file = open("/home/"+computername+"/Scrivania/RyuDatapathMonitor-master/DataLog/"+date+"queuestat.txt","a")
stringa="time"+"\t"+"datapath"+"\t"+"port_no"+"\t"+"queue_id"+"\t"+"tx_bytes"+"\t"+"tx_packets"+"\t"+"tx_errors"+"\t"+"duration_sec"+"\t"+"duration_nsec\n"
file.write(stringa)
file.close()

file = open("/home/"+computername+"/Scrivania/RyuDatapathMonitor-master/DataLog/"+date+"portstat.txt","a")
stringa="time"+"\t"+"datapath"+"\t"+"port"+"\t"+"rx-pkts"+"\t"+"rx-bytes"+"\t"+"rx_error"+"\t"+"tx-pkts"+"\t"+"tx-bytes"+"\t"+"tx-error"+"\t"+"rx-dropped"+"\t"+"tx-dropped"+"\t"+"rx-crc-err"+"\t"+"collisions"+"\n"
file.write(stringa)
file.close()

file = open("/home/"+computername+"/Scrivania/RyuDatapathMonitor-master/DataLog/"+date+"queueconfig.txt","a")
stringa="time"+"\t"+"datapath"+"\t"+"queue_id"+"\t"+"type_of_rule"+"\t"+"rate\n"
file.write(stringa)
file.close()

def get_switchis():
    try:
        output = subprocess.check_output(
            "curl -X GET http://localhost:8080/stats/switches",
            stderr=subprocess.STDOUT,
            shell=True)
        output=output[output.find("["):]
        end_response=output.find("]")
        list1=list(output)
        list1[end_response]=','
        output=''.join(list1)+"]"
    except:
        output="No NET"
    return output

def save_flow_stat(datapath):
    datapath_in=datapath
    mom_datapath= ['0' for i in range(16-len(datapath))]
    mom_datapath=''.join(mom_datapath)
    datapath=mom_datapath+datapath
    try:
        output = subprocess.check_output(
            "curl -X GET http://localhost:8080/stats/flow/"+datapath,
            shell=True)
        i=0
        output = output[output.find("{")+1:]
        end_response = output.find("}]}")+2
        list1=list(output)
        list1[end_response-2]='}'
        list1[end_response-1]=','
```

```
        output=''.join(list1)
        while i<end_response:
            output_i=output[i:]
            i=i+output[i:].find("},")+2
            otp = output_i[output_i.find("{"):]
            otp = otp[0:otp.find("},")+1]
            otp =eval(otp)
            data = otp
            json_str = json.dumps(data)
            jsonList = json.loads(json_str)
            if jsonList['priority']!=0 and jsonList['match'].get('in_port'):
                porta=str(jsonList['actions'])
                prc = porta.find('T:')
                #Check if there is "OUTPUT:" in the string
                if prc >= 0:
                    porta=porta[prc+2:]
                    porta = int(porta[0:porta.find(']')-1])
 ##                       self.logger.info('%016x %8x %17s %8x %8d %8d',
 ##                                        1,
 ##                                        jsonList['match'].get('in_port'), jsonList['match'].get('dl_dst'),
 ##                                        porta, jsonList['packet_count'],
 ##                                        jsonList['byte_count'])
                    file = open("/home/"+computername+"/Scrivania/RyuDatapathMonitor-master/DataLog/"+date+"flowstat.txt","a")
                    now=datetime.datetime.now()
                    stringa=str(now)+"\t"+datapath_in+"\t"+str(jsonList['match'].get('in_port'))+"\t"+str(jsonList['match'].get('dl_dst'))+"\t"+str(porta)+"\t"+str(jsonList['packet_count'])+"\t"+str(jsonList['byte_count'])+"\t None"+"\t None"+"\n"
                    file.write(stringa)
                    file.close()
            if jsonList['priority']!=0 and str(jsonList['actions']).find('UE:')>=0:
                SET_QUEUE=str(str(jsonList['actions'])[str(jsonList['actions']).find('UE:')+3:str(jsonList['actions']).find(',')-1])
##                  print "SET_QUEUE: "+SET_QUEUE
                file = open("/home/"+computername+"/Scrivania/RyuDatapathMonitor-master/DataLog/"+date+"flowstat.txt","a")
                now=datetime.datetime.now()
                stringa=str(now)+"\t"+datapath_in+"\t"+str(jsonList['match'].get('in_port'))+"\t"+str(jsonList['match'].get('nw_dst'))+"\t"+"None"+"\t"+str(jsonList['packet_count'])+"\t"+str(jsonList['byte_count'])+"\t"+str(jsonList['match'].get('ip_dscp'))+"\t"+SET_QUEUE+"\n"
##                  print "ip_dscp: "+str(jsonList['match'].get('ip_dscp'))
##                  print stringa
                file.write(stringa)
                file.close()

```

```
    except:
        print "FlowStat: No NET"

def save_port_stat(datapath):
    datapath_in=datapath
    mom_datapath= ['0' for i in range(16-len(datapath))]
    mom_datapath=''.join(mom_datapath)
    datapath=mom_datapath+datapath
    try:
        output = subprocess.check_output(
            "curl -X GET http://localhost:8080/stats/port/"+datapath,
            shell=True)
        i=0
        output = output[output.find("{")+1:]
        end_response = output.find("}]}")+2
        list1=list(output)
        list1[end_response-2]='}'
        list1[end_response-1]=','
        output=''.join(list1)
        while i<end_response:
            output_i=output[i:]
            i=i+output[i:].find("},")+2
            otp = output_i[output_i.find("{"):]
            otp = otp[0:otp.find("},")+1]
            otp =eval(otp)
            data = otp
            json_str = json.dumps(data)
            jsonList = json.loads(json_str)
            if jsonList['port_no']!="LOCAL":
##                      self.logger.info('%016x %8x %17s %8x %8d %8d',
##                                       1,
##                                       jsonList['match'].get('in_port'), jsonList['match'].get('dl_dst'),
##                                       porta, jsonList['packet_count'],
##                                       jsonList['byte_count'])
                file = open("/home/"+computername+"/Scrivania/RyuDatapathMonitor-master/DataLog/"+date+"portstat.txt","a")
                now=datetime.datetime.now()
                stringa=str(now)+"\t"+datapath_in+"\t"+str(jsonList['port_no'])+"\t"+str(jsonList['rx_packets'])+"\t"+str(jsonList['rx_bytes'])+"\t"+str(jsonList['rx_errors'])+"\t"+str(jsonList['tx_packets'])+"\t"+str(jsonList['tx_bytes'])+"\t"+str(jsonList['tx_errors'])+"\t"+str(jsonList['rx_dropped'])+"\t"+str(jsonList['tx_dropped'])+"\t"+str(jsonList['rx_crc_err'])+"\t"+str(jsonList['collisions'])+"\n"
                file.write(stringa)
                file.close()

    except:
        print "PortStat: No NET"

def save_queue_stat(datapath):
    datapath_in=datapath
```

```
    mom_datapath= ['0' for i in range(16-len(datapath))]
    mom_datapath=''.join(mom_datapath)
    datapath=mom_datapath+datapath
    try:
        output = subprocess.check_output(
             "curl -X GET http://localhost:8080/qos/queue/status/"+datapath,
             shell=True)
        i=0
        output = output[output.find("ult")+1:]
        output = output[output.find("{")+1:]
        end_response = output.find("}]}")+2
        list1=list(output)
        list1[end_response-2]='}'
        list1[end_response-1]=','
        output=''.join(list1)
        if output[output.find(":")+2:output.find(":")+4]!="[]":
            while i<end_response:
                output_i=output[i:]
                i=i+output[i:].find("},")+2
                otp = output_i[output_i.find("{"):]
                otp = otp[0:otp.find("},")+1]
##                print otp
                otp = eval(otp)
                data = otp
                json_str = json.dumps(data)
                jsonList = json.loads(json_str)
                file = open("/home/"+computername+"/Scrivania/RyuDatapathMonitor-master/DataLog/"+date+"queuestat.txt","a")
                now=datetime.datetime.now()
                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"
                file.write(stringa)
                file.close()

    except:
        print "QueueStat: No NET"

def save_queue_config(datapath):
    datapath_in=datapath
    mom_datapath= ['0' for i in range(16-len(datapath))]
    mom_datapath=''.join(mom_datapath)
    datapath=mom_datapath+datapath
    try:
        output = subprocess.check_output(
             "curl -X GET http://localhost:8080/qos/queue/"+datapath,
             shell=True)
        i=0
        output=output[1:len(output)-1]
        jsonList = json.loads(output)
        config = jsonList['command_result'].get('details')
```

```
        for queue in config:
            if queue=='2':
                rate = jsonList['command_result'].get('details').get(queue).get('config').get('min-rate')
                type_of_rule='min_rate'
            else:
                rate = jsonList['command_result'].get('details').get(queue).get('config').get('max-rate')
                type_of_rule='max_rate'
            file = open("/home/"+computername+"/Scrivania/RyuDatapathMonitor-master/DataLog/"+date+"queueconfig.txt","a")
            now=datetime.datetime.now()
            stringa=str(now)+"\t"+datapath_in+"\t"+str(queue)+"\t"+str(type_of_rule)+"\t"+str(rate)+"\n"
            file.write(stringa)
            file.close()
    except:
        print "QueueConfig: No NET"

class DatapathMonitor():
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    def __init__(self, args):
        self.datapath_list = args["dp"]

        self.monitor = threading.Thread(target=self.switch_monitor_thread)
        self.delta = args["step"]
        self.logger = args["logger"]

        self.started = False

    def start(self):
        self.monitor.start()
        self.started = True

    def update(self, dplist):
        self.datapath_list = dplist

    def switch_monitor_thread(self):
        max_rate_queue=100#Mps
        max_rate_queue=max_rate_queue*1000000
        minute_wait=20
        Time_queue=minute_wait*60
        Change_flag=Time_queue/self.delta
        counter=Change_flag
        c_q2=0
        c_q1=1
        c_q0=1
        q2=np.arange(70, 101, 10)*1000000
        q1=np.arange(0, 101, 10)*1000000
        q0=np.arange(0, 101, 10)*1000000
##          time.sleep(40)
```

```
        print 'Wait for time alignment'
        wait=self.delta/60
        check_time=False
        while check_time==False:
            now=datetime.datetime.now()
            if now.minute%wait==0:
                check_time=True
            else:
                time.sleep(1)
        print 'Starting Save Data'

        while True:
            check_time=False
            while check_time==False:
                now=datetime.datetime.now()
                if now.minute%wait==0:
                    check_time=True
                    print "Save Time: "+str(now)
                else:
                    time.sleep(1)
            NET=get_switchis()
##            print "NET="+NET
            t_sleep = 0.9
            if NET != "NO NET" and NET!="[,]":
##                end_response = output.find("]")
                i=1
                while i<NET.find("]"):
                    mom_NET=NET[i:]
                    datapath=NET[i:i+mom_NET.find(",")]
##                    time.sleep(1)
                    i=i+mom_NET.find(",")+2
                    save_flow_stat(datapath)
##                    time.sleep(0.9)
                    save_port_stat(datapath)
##                    time.sleep(t_sleep)
                    save_queue_stat(datapath)
##                    time.sleep(t_sleep)
                    save_queue_config(datapath)
                print counter
                if counter >= Change_flag:
                    set_queue("1", "s1-eth2", str(max_rate_queue), "{\"max_rate\": \""+str(q0[c_q0])+"\"}, {\"max_rate\": \""+str(q1[c_q1])+"\"}, {\"min_rate\": \""+str(q2[c_q2])+"\"}")
                    set_queue("2", "s2-eth2", str(max_rate_queue), "{\"max_rate\": \""+str(q0[c_q0])+"\"}, {\"max_rate\": \""+str(q1[c_q1])+"\"}, {\"min_rate\": \""+str(q2[c_q2])+"\"}")
                    counter=0
                    c_q0=c_q0+1
                    if c_q0>len(q0)-1:
                        c_q0=1
                        c_q1=c_q1+1
                        if c_q1>len(q1)-1:
                            c_q1=1
                            c_q2=c_q2+1
```

```
                            if c_q2>len(q2)-1:
                                c_q2=0
                else:
                    counter=counter+1

            else:
                print "No Network"
```

A.2 main controller

```
from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_3
from qos_simple_switch_13 import *
from datapath_monitor_TOS import *
from ryu.lib.packet import arp
from ryu.lib.packet.arp import ARP_REQUEST, ARP_REPLY
from ryu.lib.packet import ipv4


class MainControllerMonitor(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    def __init__(self, *args, **kwargs):
        super(MainControllerMonitor, self).__init__(*args, **kwargs)
        self.device_behaviour = SimpleSwitch13(*args, **kwargs)
        self.datapath_id_list = []
        self.mac_to_port = {}
        STEP = 300
        args = {
            "step":STEP,
            "logger":self.logger,
            "dp":self.datapath_id_list
        }
        self.monitor = DatapathMonitor(args)
        self.monitor.start()

    @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
    def switch_feature_handler(self, ev):
        self.device_behaviour.switch_features_handler(ev)
        datapath = ev.msg.datapath
        if datapath not in self.datapath_id_list:
            self.datapath_id_list.append(datapath)
            self.monitor.update(self.datapath_id_list)
##          match = parser.OFPMatch()
##          actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER, ofproto.OFPCML_NO_BUFFER)]

    @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
```

```
    def _packet_in_handler(self, ev):
        self.device_behaviour._packet_in_handler(ev)
##        msg = ev.msg
##        datapath = msg.datapath
##        ofproto = datapath.ofproto
##        parser = datapath.ofproto_parser
##        in_port = msg.match['in_port']
##        ip_dscp = msg.match.get('ip_dscp')
##        print 'ip dscp',ip_dscp
##        pkt = packet.Packet(msg.data)
##        eth = pkt.get_protocols(ethernet.ethernet)[0]
##        pkt = packet.Packet(msg.data)
##        arp_pkt = pkt.get_protocol(arp.arp)
##        ip_pkt = pkt.get_protocol(ipv4.ipv4)
##        dst = eth.dst
##        src=eth.src
##        print src
##        dpid = datapath.id
##        self.mac_to_port.setdefault(dpid, {})
##        self.logger.info("packet in %s %s %s %s", dpid, src, dst, in_port)
        # learn a mac address to avoid FLOOD next time.
##        self.mac_to_port[dpid][src] = in_port

##        if dst in self.mac_to_port[dpid]:
##            out_port = self.mac_to_port[dpid][dst]
##        else:
##            out_port = ofproto.OFPP_FLOOD
##            actions = [parser.OFPActionOutput(out_port)]
##        arp_pkt = pkt.get_protocol(arp.arp)
##
##
##
##        # install a flow to avoid packet_in next time
##        if out_port != ofproto.OFPP_FLOOD:
##            if arp_pkt:
##                match = parser.OFPMatch( in_port=in_port, eth_src=src)
##            elif ip_dscp is not None and ip_dscp != 0:
##                match = parser.OFPMatch( eth_type=0x0800,ip_dscp=ip_dscp)
##                print match



##    @set_ev_cls(ofp_event.EventOFPPortStatsReply, MAIN_DISPATCHER)
##    def port_stats_reply_handler(self, ev):
##        self.monitor.port_stats_reply(ev)

##    @set_ev_cls(ofp_event.EventOFPFlowStatsReply, MAIN_DISPATCHER)
##    def flow_stats_reply_handler(self, ev):
##        self.monitor.flow_stats_reply(ev)

##    @set_ev_cls(ofp_event.EventOFPQueueStatsReply, MAIN_DISPATCHER)
##    def queue_stats_reply_handler(self, ev):
```

```
##            self.monitor.queue_stats_reply(ev)
##
##        @set_ev_cls(ofp_event.EventOFPQueueGetConfigReply, MAIN_DISPATCHER)
##        def queue_desc_reply_handler(self, ev):
##            self.monitor.queue_desc_reply(ev)




    @set_ev_cls(ofp_event.EventOFPDescStatsReply, MAIN_DISPATCHER)
    def desc_stat_reply_handler(self, ev):
        self.monitor.desc_reply(ev)

##        @set_ev_cls(ofp_event.EventOFPQueueStatsReply, MAIN_DISPATCHER)
##        def send_queue_stats_handler(self, ev):
##            self.monitor.queue_stats_reply(ev)
```

A.3 Controller commands

```
from subprocess import call
import threading
import subprocess
import random
import os
import time
import json
import sys

def ovsdb_addr(datapath):
    datapath_in=datapath
    mom_datapath= ['0' for i in range(16-len(datapath))]
    mom_datapath=''.join(mom_datapath)
    datapath=mom_datapath+datapath
    print "Set ovsdb on switch "+datapath
    try:
        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")
        print "\n"
        time.sleep(2)
    except:
        print "ovsdb: ERROR"

def ovssctl_set_bridge(switch_name):
    print "Set ovssctl on switch "+switch_name
    try:
        os.popen("sudo -S ovs-vsctl set Bridge "+switch_name+" protocols=OpenFlow13", 'w').write("Ao70pa45")
        print "\n"
##        time.sleep(2)
```

```
    except:
        print "ovssctl_set_bridge: ERROR"

def get_switchis():
    print "Get switches id"
    try:
        output = subprocess.check_output(
             "curl -X GET http://localhost:8080/stats/switches",
             stderr=subprocess.STDOUT,
             shell=True)
        output=output[output.find("["):]
        end_response=output.find("]")
        list1=list(output)
        list1[end_response]=','
        output=''.join(list1)+"]"
        print "\n"
##          time.sleep(2)
    except:
        output="NO NET"
    return output

def switch_ports_name(datapath):
    datapath_in=datapath
    mom_datapath= ['0' for i in range(16-len(datapath))]
    mom_datapath=''.join(mom_datapath)
    datapath=mom_datapath+datapath
    print "Get names on switch "+datapath
    try:
        output = subprocess.check_output(
             "curl -X GET http://localhost:8080/stats/portdesc/"+datapath ,
             stderr=subprocess.STDOUT,
             shell=True)
        i=0
        output = output[output.find("{")+1:]
        end_response = output.find("}]}")+2
        list1=list(output)
        list1[end_response -2]='}'
        list1[end_response -1]=','
        output=''.join(list1)
        names = []
        while i<end_response:
            output_i=output[i:]
            i=i+output[i:].find("},")+2
            otp = output_i[output_i.find("{"):]
            otp = otp[0:otp.find("},")+1]
            otp =eval(otp)
            data = otp
            json_str = json.dumps(data)
            jsonList = json.loads(json_str)
            if jsonList['port_no']=="LOCAL":
                names.append(str(jsonList['name']))
            else:
                names.append(str(jsonList['name']))
```

```
            print "\n"
            return names


        except:
            print "Switch port name: ERROR"


def queue_rule(datapath, port_number, ip_dscp, queue_number):
    mom_datapath= ['0' for i in range(16-len(datapath))]
    mom_datapath=''.join(mom_datapath)
    datapath=mom_datapath+datapath
    print "Set queue rule on switch "+datapath+" on port "+port_number
    try:
        os.popen("curl -X POST -d '{\"match\": {\"ip_dscp\": \""+ip_dscp+"\"}, \"actions\":{\"queue\": \""+queue_number+"\"}}' http://localhost:8080/qos/rules/"+datapath, 'w').write("Ao70pa45")
        time.sleep(0.1)
        print "\n"
    except:
        print "Set queue rule: Error"

def queue_rule_byIP(datapath, port_number, ip_dscp, queue_number, ip_dst):
    mom_datapath= ['0' for i in range(16-len(datapath))]
    mom_datapath=''.join(mom_datapath)
    datapath=mom_datapath+datapath
    print "Set queue rule on switch "+datapath+" on port "+port_number
    try:
        os.popen("curl -X POST -d '{\"match\": {\"nw_dst\": \""+ip_dst+"\", \"ip_dscp\": \""+ip_dscp+"\"}, \"actions\":{\"queue\": \""+queue_number+"\"}}' http://localhost:8080/qos/rules/"+datapath, 'w').write("Ao70pa45")
        time.sleep(0.1)
        print "\n"
    except:
        print "Set queue rule: Error"

def set_queue(datapath, port_id, max_rate, queue_rate_list):
    mom_datapath= ['0' for i in range(16-len(datapath))]
    mom_datapath=''.join(mom_datapath)
    datapath=mom_datapath+datapath
    print "Set queue on port "+port_id+" of switch "+datapath
    print queue_rate_list
    try:
##        os.popen("curl -X POST -d '{\"port_name\": \""+port_id+"\", \"type\": \"linux-htb\", \"max_rate\": \""+max_rate+"\", \"queues\": ["+queue_rate_list+"]}' http://localhost:8080/qos/queue/"+datapath, 'w').write("Ao70pa45")
        output = subprocess.check_output("curl -X POST -d '{\"port_name\": \""+port_id+"\", \"type\": \"linux-htb\", \"max_rate\": \""+max_rate+"\", \"queues\": ["+queue_rate_list+"]}' http://localhost:8080/qos/queue/"+datapath,
                stderr=subprocess.STDOUT,
```

```
                shell=True)
        time.sleep(0.1)
        print "\n"
    except:
        print "Set queue: Error"


def set_Telecom_queue(datapath, port_number, IP_flag, IP_dst):
    port=port_number
    mom_datapath= ['0' for i in range(16-len(datapath))]
    mom_datapath=''.join(mom_datapath)
    datapath=mom_datapath+datapath
    if IP_flag==True:
        #Default Queue (queue_id = 0)
    #    queue_rule(datapath, port, "0", "0")#Service 0
    #    queue_rule(datapath, port, "32", "0")#Service 1
    #    queue_rule(datapath, port, "96", "0")#Service 3
        queue_rule_byIP(datapath, port, "0", "0", IP_dst)#Service 0
        queue_rule_byIP(datapath, port, "8", "0", IP_dst)#Service 1
        queue_rule_byIP(datapath, port, "10", "0", IP_dst)#Service 1
        queue_rule_byIP(datapath, port, "12", "0", IP_dst)#Service 1
        queue_rule_byIP(datapath, port, "14", "0", IP_dst)#Service 1
        queue_rule_byIP(datapath, port, "24", "0", IP_dst)#Service 3
        queue_rule_byIP(datapath, port, "26", "0", IP_dst)#Service 3
        queue_rule_byIP(datapath, port, "28", "0", IP_dst)#Service 3
        queue_rule_byIP(datapath, port, "30", "0", IP_dst)#Service 3
        #Premium Queue (queue_id = 1)
    #    queue_rule(datapath, port, "72", "1")#Service 2
    #    queue_rule(datapath, port, "136", "1")#Service 4
    #    queue_rule(datapath, port, "192", "1")#Service 6
    #    queue_rule(datapath, port, "224", "1")#Service 7
        queue_rule_byIP(datapath, port, "16", "1", IP_dst)#Service 2
        queue_rule_byIP(datapath, port, "18", "1", IP_dst)#Service 2
        queue_rule_byIP(datapath, port, "20", "1", IP_dst)#Service 2
        queue_rule_byIP(datapath, port, "22", "1", IP_dst)#Service 2
        queue_rule_byIP(datapath, port, "32", "1", IP_dst)#Service 4
        queue_rule_byIP(datapath, port, "34", "1", IP_dst)#Service 4
        queue_rule_byIP(datapath, port, "36", "1", IP_dst)#Service 4
        queue_rule_byIP(datapath, port, "38", "1", IP_dst)#Service 4
        queue_rule_byIP(datapath, port, "48", "1", IP_dst)#Service 6
        queue_rule_byIP(datapath, port, "56", "1", IP_dst)#Service 7
        #Gold Queue (queue_id = 2)
    #    queue_rule(datapath, port, "160", "2")#Service 5
        queue_rule_byIP(datapath, port, "40", "2", IP_dst)#Service 5
        queue_rule_byIP(datapath, port, "46", "2", IP_dst)#Service 5

    if IP_flag==False:
        #Default Queue (queue_id = 0)
    #    queue_rule(datapath, port, "0", "0")#Service 0
    #    queue_rule(datapath, port, "32", "0")#Service 1
    #    queue_rule(datapath, port, "96", "0")#Service 3
        queue_rule(datapath, port, "0", "0")#Service 0
        queue_rule(datapath, port, "8", "0")#Service 1
        queue_rule(datapath, port, "10", "0")#Service 1
```

```
        queue_rule(datapath, port, "12", "0")#Service 1
        queue_rule(datapath, port, "14", "0")#Service 1
        queue_rule(datapath, port, "24", "0")#Service 3
        queue_rule(datapath, port, "26", "0")#Service 3
        queue_rule(datapath, port, "28", "0")#Service 3
        queue_rule(datapath, port, "30", "0")#Service 3
        #Premium Queue (queue_id = 1)
#       queue_rule(datapath, port, "72", "1")#Service 2
#       queue_rule(datapath, port, "136", "1")#Service 4
#       queue_rule(datapath, port, "192", "1")#Service 6
#       queue_rule(datapath, port, "224", "1")#Service 7
        queue_rule(datapath, port, "16", "1")#Service 2
        queue_rule(datapath, port, "18", "1")#Service 2
        queue_rule(datapath, port, "20", "1")#Service 2
        queue_rule(datapath, port, "22", "1")#Service 2
        queue_rule(datapath, port, "32", "1")#Service 4
        queue_rule(datapath, port, "34", "1")#Service 4
        queue_rule(datapath, port, "36", "1")#Service 4
        queue_rule(datapath, port, "38", "1")#Service 4
        queue_rule(datapath, port, "48", "1")#Service 6
        queue_rule(datapath, port, "56", "1")#Service 7
        #Gold Queue (queue_id = 2)
#       queue_rule(datapath, port, "160", "2")#Service 5
        queue_rule(datapath, port, "40", "2")#Service 5
        queue_rule(datapath, port, "46", "2")#Service 5
```

A.4 Topology

```
from mininet.net import Mininet
from mininet.node import Controller, RemoteController, OVSController
from mininet.node import CPULimitedHost, Host, Node
from mininet.node import OVSKernelSwitch, UserSwitch
from mininet.node import IVSSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink, Intf
from subprocess import call
import threading
import subprocess
import random
import os
import time
import datetime
import json
import sys
import ditg

from Controller_commands import *


def myNetwork():
```

```
    max_rate_queue=100#Mbps
    max_rate_queue=max_rate_queue*1000000
    Default=str(max_rate_queue*20/100)#20%
    Premium=str(max_rate_queue*80/100)#80%
    Gold=str(max_rate_queue*100/100)#100%

    change_values=6#change every number*5 minutes
    Q0=False
    #Stress Queue 1 (2,4,6,7)
    Q1=False
    #Stress Queue 2 (5)
    Q2=False
    #moltiplicator initialization
    F0_max=2
    F1_max=300
    F2_max=300
    F0=1
    F1=1
    F2=1

    net=Mininet( topo=None,
                 build=False,
                 ipBase='10.0.0.0/8')

    info( '***Adding controller\n')
    c0=net.addController(name='c0',
                         controller=RemoteController,
                         ip='127.0.0.1',
                         protocol='tcp',
                         port=6633)

    info( '***Adding switches\n')
    s1 = net.addSwitch('s1',dpid='0000000000000001',protocols="OpenFlow13")
    s2 = net.addSwitch('s2',dpid='0000000000000002',protocols="OpenFlow13")

    info( '***Adding Host\n')
    h0 = net.addHost('h0', ip='10.10/24', mac='00:00:00:00:00:0a')
    h1 = net.addHost('h1', ip='10.11/24', mac='00:00:00:00:00:0b')
    h2 = net.addHost('h2', ip='10.12/24', mac='00:00:00:00:00:0c')
    h3 = net.addHost('h3', ip='10.13/24', mac='00:00:00:00:00:0d')
    h4 = net.addHost('h4', ip='10.14/24', mac='00:00:00:00:00:0e')
    h5 = net.addHost('h5', ip='10.15/24', mac='00:00:00:00:00:0f')

    info( '***Adding Link\n')
##    net.addLink(s1,s2,2,2,cls=TCLink,bw=10)
    net.addLink(s1,s2,2,2)

    net.addLink(s1, h0,3,0)
    net.addLink(s1, h1,4,0)
    net.addLink(s1, h2,5,0)
    net.addLink(s2, h3,3,0)
    net.addLink(s2, h4,4,0)
```

```
    net.addLink(s2, h5,5,0)

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

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

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


    #Attivazione del manager in ascolto sulla porta 6632
    os.popen("sudo -S ovs-vsctl set-manager ptcp:6632", 'w').write("Ao70pa45")
    time.sleep(2)
    NET = get_switchis()
    if NET != "NO NET" and NET!="[,]":
        i=1
        while i<NET.find("]"):
            mom_NET=NET[i:]
            datapath=NET[i:i+mom_NET.find(",")]
            i=i+mom_NET.find(",")+2
            port_id = switch_ports_name(datapath)
            for k in range(0,len(port_id)):
                if len(port_id[k])<=2:
                    pp=port_id[k]
                    print pp
                    ovssctl_set_bridge(port_id[k])
        time.sleep(0.2)
        i=1
        while i<NET.find("]"):
            mom_NET=NET[i:]
            datapath=NET[i:i+mom_NET.find(",")]
            i=i+mom_NET.find(",")+2
            port_id = switch_ports_name(datapath)
            ovsdb_addr(datapath)
            IP_Flag=True
            for index in range(0,len(port_id)):
                if port_id[index]=="s1-eth2" or port_id[index]=="s2-eth2":
##                      print "Port_ID: "+port_id[index]
                    set_queue(datapath, port_id[index], str(max_rate_queue), "{\"max_rate\": \""+Default+"\"}, {\"max_rate\": \""+Premium+"\"}, {\"min_rate\": \""+Gold+"\"}")
                    port = port_id[index][port_id[index].find("h")+1:]
                    if port_id[index]=="s1-eth2":
                        IP_Destination="10.0.0.11"
                    if port_id[index]=="s2-eth2":
                        IP_Destination="10.0.0.13"
                    set_Telecom_queue(datapath, port, IP_Flag, IP_Destination)
```

```
            for index in range(0,len(port_id)):
                if port_id[index]=="s1-eth4" or port_id[index]=="s2-eth3":
##                  set_queue(datapath, port_id[index], str(max_rate_queue), "{\"max_rate\": \""+Default+"\"}, {\"max_rate\": \""+Premium+"\"}, {\"min_rate\": \""+Gold+"\"}")
                    port = port_id[index][port_id[index].find("h")+1:]
                    if port_id[index]=="s1-eth4":
                        IP_Destination="10.0.0.13"
                    if port_id[index]=="s2-eth3":
                        IP_Destination="10.0.0.11"
                    set_Telecom_queue(datapath, port, IP_Flag, IP_Destination)

    # Import CSV
    print 'Csv Import'
    serv = ditg.pd.read_csv(ditg.CSV, sep=';', usecols=[0], skiprows=[0]) # serv 0 tx
    time_values = serv.values
    serv = ditg.pd.read_csv(ditg.CSV, sep=';', usecols=[1], skiprows=[0]) # serv 0 tx
    serv_0_tx = serv.values
    serv = ditg.pd.read_csv(ditg.CSV, sep=';', usecols=[2], skiprows=[0]) # serv 0 rx
    serv_0_rx = serv.values
    serv = ditg.pd.read_csv(ditg.CSV, sep=';', usecols=[3], skiprows=[0]) # serv 1 tx
    serv_1_tx = serv.values
    serv = ditg.pd.read_csv(ditg.CSV, sep=';', usecols=[4], skiprows=[0]) # serv 1 rx
    serv_1_rx = serv.values
    serv = ditg.pd.read_csv(ditg.CSV, sep=';', usecols=[5], skiprows=[0]) # serv 2 tx
    serv_2_tx = serv.values
    serv = ditg.pd.read_csv(ditg.CSV, sep=';', usecols=[6], skiprows=[0]) # serv 2 rx
    serv_2_rx = serv.values
    serv = ditg.pd.read_csv(ditg.CSV, sep=';', usecols=[15], skiprows=[0]) # serv 3 tx
    serv_3_tx = serv.values
    serv = ditg.pd.read_csv(ditg.CSV, sep=';', usecols=[16], skiprows=[0]) # serv 3 rx
    serv_3_rx = serv.values
    serv = ditg.pd.read_csv(ditg.CSV, sep=';', usecols=[11], skiprows=[0]) # serv 4 tx
    serv_4_tx = serv.values
    serv = ditg.pd.read_csv(ditg.CSV, sep=';', usecols=[12], skiprows=[0]) # serv 4 rx
    serv_4_rx = serv.values
    serv = ditg.pd.read_csv(ditg.CSV, sep=';', usecols=[7], skiprows=[0]) # serv 5 tx
    serv_5_tx = serv.values
    serv = ditg.pd.read_csv(ditg.CSV, sep=';', usecols=[8], skiprows=[0]) # serv 5 rx
```

```
    serv_5_rx = serv.values
    serv = ditg.pd.read_csv(ditg.CSV, sep=';', usecols=[9], skiprows=[0]) # serv 6 tx
    serv_6_tx = serv.values
    serv = ditg.pd.read_csv(ditg.CSV, sep=';', usecols=[10], skiprows=[0]) # serv 6 rx
    serv_6_rx = serv.values
    serv = ditg.pd.read_csv(ditg.CSV, sep=';', usecols=[13], skiprows=[0]) # serv 7 tx
    serv_7_tx = serv.values
    serv = ditg.pd.read_csv(ditg.CSV, sep=';', usecols=[14], skiprows=[0]) # serv 7 rx
    serv_7_rx = serv.values

    i = 0
    j = 0
    # pkts
    n = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
    # pkts per second
    avg = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
    F0=F0_max
    F1=1
    F2=1
##     print time_values[0]
    print 'Wait for time alignment'
    wait=ditg.TIME/60
    check_time=False
    while check_time==False:
        now=datetime.datetime.now()
        time.sleep(0.1)
        if now.minute%wait==0:
            check_time=True
            if len(str(now.minute))==1:
                starting_time=str(now.hour)+':0'+str(now.minute)
            else:
                starting_time=str(now.hour)+':'+str(now.minute)
            print starting_time
            k=0
            for index in time_values:
                if index==starting_time:
                    i=k
                    break
                k=k+1
    print 'Starting Time: '+str(time_values[i])

    while j < ditg.SIM_N:
        #calculate the moltiplicators

        if Q0:
            if j%change_values==0:
                if F0==1:
                    F0=F0_max
                else:
                    F0=1
```

```
        if Q1:
            if j%change_values==0:
                if F1==1:
                    F1=F1_max
                else:
                    F1=1
        if Q2:
            if j%change_values==0:
                if F2==1:
                    F2=F2_max
                else:
                    F2=1



        # Server start
        print 'Start ITGRecv'
        h1.cmd('ITGRecv &')
        h3.cmd('ITGRecv &')
        time.sleep(2)
        # Sum of packets
        sum_in = 0
        sum_out = 0
        # Serv 0 tx
        n[0] = int(serv_0_tx[i])*F0 / ditg.SCALE + 1
        avg[0] = n[0] / (ditg.TIME-10) + 1
        if avg[0] > 0 and n[0] > 0:
            #com = ditg.createCmd(src=ditg.src,dst=ditg.dst,tos=ditg.SERV_0,nPkts=str(n[0]),avg=str(avg[0]))
            com = ditg.createCmd_2(dst=ditg.dst,port="10001",tos=ditg.SERV_0,nPkts=str(n[0]),avg=str(avg[0]))
            print(com)
            h1.cmd(com)
            sum_in = sum_in + n[0]

        # Serv 0 rx
        n[0] = int(serv_0_rx[i])*F0 / ditg.SCALE + 1
        avg[0] = n[0] / (ditg.TIME-10) + 1
        if avg[0] > 0 and n[0] > 0:
            #com = ditg.createCmd(src=ditg.dst,dst=ditg.src,tos=ditg.SERV_0,nPkts=str(n[0]),avg=str(avg[0]))
            com = ditg.createCmd_2(dst=ditg.src,port="10001",tos=ditg.SERV_0,nPkts=str(n[0]),avg=str(avg[0]))
            print(com)
            h3.cmd(com)
            sum_out = sum_out + n[0]

        # Serv 1 tx
        n[1] = int(serv_1_tx[i])*F0 / ditg.SCALE + 1
        avg[1] = n[1] / (ditg.TIME-10) + 1
        if avg[1] > 0 and n[1] > 0:
            #com = ditg.createCmd(src=ditg.src,dst=ditg.dst,tos=ditg.SERV_1,nPkts=str(n[1]),avg=str(avg[1]))
```

```
            com = ditg.createCmd_2(dst=ditg.dst,port="10002",tos=ditg.SERV_1,nPkts=str(n[1]),avg=str(avg[1]))
            print(com)
            h1.cmd(com)
            sum_in = sum_in + n[1]

        # Serv 1 rx
        n[1] = int(serv_1_rx[i])*F0 / ditg.SCALE + 1
        avg[1] = n[1] / (ditg.TIME-10) + 1
        if avg[1] > 0 and n[1] > 0:
            #com = ditg.createCmd(src=ditg.dst,dst=ditg.src,tos=ditg.SERV_1,nPkts=str(n[1]),avg=str(avg[1]))
            com = ditg.createCmd_2(dst=ditg.src,port="10002",tos=ditg.SERV_1,nPkts=str(n[1]),avg=str(avg[1]))
            print(com)
            h3.cmd(com)
            sum_out = sum_out + n[1]

        # Serv 2 tx
        n[2] = int(serv_2_tx[i])*F1 / ditg.SCALE + 1
        avg[2] = n[2] / (ditg.TIME-10) + 1
        if avg[2] > 0 and n[2] > 0:
            #com = ditg.createCmd(src=ditg.src,dst=ditg.dst,tos=ditg.SERV_2,nPkts=str(n[2]),avg=str(avg[2]))
            com = ditg.createCmd_2(dst=ditg.dst,port="10003",tos=ditg.SERV_2,nPkts=str(n[2]),avg=str(avg[2]))
            print(com)
            h1.cmd(com)
            sum_in = sum_in + n[2]

        # Serv 2 rx
        n[2] = int(serv_2_rx[i])*F1 / ditg.SCALE + 1
        avg[2] = n[2] / (ditg.TIME-10) + 1
        if avg[2] > 0 and n[2] > 0:
            #com = ditg.createCmd(src=ditg.dst,dst=ditg.src,tos=ditg.SERV_2,nPkts=str(n[2]),avg=str(avg[2]))
            com = ditg.createCmd_2(dst=ditg.src,port="10003",tos=ditg.SERV_2,nPkts=str(n[2]),avg=str(avg[2]))
            print(com)
            h3.cmd(com)
            sum_out = sum_out + n[2]

        # Serv 3 tx
        n[3] = int(serv_3_tx[i])*F0 / ditg.SCALE + 1
        avg[3] = n[3] / (ditg.TIME-10) + 1
        if avg[3] > 0 and n[3] > 0:
            #com = ditg.createCmd(src=ditg.src,dst=ditg.dst,tos=ditg.SERV_3,nPkts=str(n[3]),avg=str(avg[3]))
            com = ditg.createCmd_2(dst=ditg.dst,port="10004",tos=ditg.SERV_3,nPkts=str(n[3]),avg=str(avg[3]))
            print(com)
            h1.cmd(com)
            sum_in = sum_in + n[3]

```

```
        # Serv 3 rx
        n[3] = int(serv_3_rx[i])*F0 / ditg.SCALE + 1
        avg[3] = n[3] / (ditg.TIME-10) + 1
        if avg[3] > 0 and n[3] > 0:
            #com = ditg.createCmd(src=ditg.dst,dst=ditg.src,tos=ditg.SERV_3,nPkts=str(n[3]),avg=str(avg[3]))
            com = ditg.createCmd_2(dst=ditg.src,port="10004",tos=ditg.SERV_3,nPkts=str(n[3]),avg=str(avg[3]))
            print(com)
            h3.cmd(com)
            sum_out = sum_out + n[3]

        # Serv 4 tx
        n[4] = int(serv_4_tx[i])*F1 / ditg.SCALE + 1
        avg[4] = n[4] / (ditg.TIME-10) + 1
        if avg[4] > 0 and n[4] > 0:
            #com = ditg.createCmd(src=ditg.src,dst=ditg.dst,tos=ditg.SERV_4,nPkts=str(n[4]),avg=str(avg[4]))
            com = ditg.createCmd_2(dst=ditg.dst,port="10005",tos=ditg.SERV_4,nPkts=str(n[4]),avg=str(avg[4]))
            print(com)
            h1.cmd(com)
            sum_in = sum_in + n[4]

        # Serv 4 rx
        n[4] = int(serv_4_rx[i])*F1 / ditg.SCALE + 1
        avg[4] = n[4] / (ditg.TIME-10) + 1
        if avg[4] > 0 and n[4] > 0:
            #com = ditg.createCmd(src=ditg.dst,dst=ditg.src,tos=ditg.SERV_4,nPkts=str(n[4]),avg=str(avg[4]))
            com = ditg.createCmd_2(dst=ditg.src,port="10005",tos=ditg.SERV_4,nPkts=str(n[4]),avg=str(avg[4]))
            print(com)
            h3.cmd(com)
            sum_out = sum_out + n[4]

        # Serv 5 tx
        n[5] = int(serv_5_tx[i])*F2 / ditg.SCALE + 1
        avg[5] = n[5] / (ditg.TIME-10) + 1
        if avg[5] > 0 and n[5] > 0:
            #com = ditg.createCmd(src=ditg.src,dst=ditg.dst,tos=ditg.SERV_5,nPkts=str(n[5]),avg=str(avg[5]))
            com = ditg.createCmd_2(dst=ditg.dst,port="10006",tos=ditg.SERV_5,nPkts=str(n[5]),avg=str(avg[5]))
            print(com)
            h1.cmd(com)
            sum_in = sum_in + n[5]

        # Serv 5 rx
        n[5] = int(serv_5_rx[i])*F2 / ditg.SCALE + 1
        avg[5] = n[5] / (ditg.TIME-10) + 1
        if avg[5] > 0 and n[5] > 0:
            #com = ditg.createCmd(src=ditg.dst,dst=ditg.src,tos=ditg.SERV_5,nPkts=str(n[5]),avg=str(avg[5]))
```

```
            com = ditg.createCmd_2(dst=ditg.src,port="10006",tos=ditg.SERV_5,nPkts=str(n[5]),avg=str(avg[5]))
            print(com)
            h3.cmd(com)
            sum_out = sum_out + n[5]

        # Serv 6 tx
        n[6] = int(serv_6_tx[i])*F1 / ditg.SCALE + 1
        avg[6] = n[6] / (ditg.TIME-10) + 1
        if avg[6] > 0 and n[6] > 0:
            #com = ditg.createCmd(src=ditg.src,dst=ditg.dst,tos=ditg.SERV_6,nPkts=str(n[6]),avg=str(avg[6]))
            com = ditg.createCmd_2(dst=ditg.dst,port="10007",tos=ditg.SERV_6,nPkts=str(n[6]),avg=str(avg[6]))
            print(com)
            h1.cmd(com)
            sum_in = sum_in + n[6]

        # Serv 6 rx
        n[6] = int(serv_6_rx[i])*F1 / ditg.SCALE + 1
        avg[6] = n[6] / (ditg.TIME-10) + 1
        if avg[6] > 0 and n[6] > 0:
            #com = ditg.createCmd(src=ditg.dst,dst=ditg.src,tos=ditg.SERV_6,nPkts=str(n[6]),avg=str(avg[6]))
            com = ditg.createCmd_2(dst=ditg.src,port="10007",tos=ditg.SERV_6,nPkts=str(n[6]),avg=str(avg[6]))
            print(com)
            h3.cmd(com)
            sum_out = sum_out + n[6]


        # Serv 7 tx
        n[7] = int(serv_7_tx[i])*F1 / ditg.SCALE + 1
        avg[7] = n[7] / (ditg.TIME-10) + 1
        if avg[7] > 0 and n[7] > 0:
            #com = ditg.createCmd(src=ditg.src,dst=ditg.dst,tos=ditg.SERV_7,nPkts=str(n[7]),avg=str(avg[7]))
            com = ditg.createCmd_2(dst=ditg.dst,port="10008",tos=ditg.SERV_7,nPkts=str(n[7]),avg=str(avg[7]))
            print(com)
            h1.cmd(com)
            sum_in = sum_in + n[7]

        # Serv 7 rx
        n[7] = int(serv_7_rx[i])*F1 / ditg.SCALE + 1
        avg[7] = n[7] / (ditg.TIME-10) + 1
        if avg[7] > 0 and n[7] > 0:
            #com = ditg.createCmd(src=ditg.dst,dst=ditg.src,tos=ditg.SERV_7,nPkts=str(n[7]),avg=str(avg[7]))
            com = ditg.createCmd_2(dst=ditg.src,port="10008",tos=ditg.SERV_7,nPkts=str(n[7]),avg=str(avg[7]))
            print(com)
            h3.cmd(com)
            sum_out = sum_out + n[7]
```

```
        j = j+1
##          i = j % ditg.SIZE
        if i % ditg.SIZE==0:
            i=0
        else:
            i=i+1
        print('Sum of Packets IN: ' + str(sum_in))
        print('Sum of Packets OUT: ' + str(sum_out))

        check_time=False
        while check_time==False:
            now=datetime.datetime.now()
            if now.minute%wait==0:
                check_time=True
                print "Time: "+str(now)
                print 'Database Time: '+str(time_values[i])
            else:
                time.sleep(1)
        h1.cmd('kill %ITGSend')
        h1.cmd('kill %ITGRecv')
    CLI(net)
    net.stop()

if __name__=='__main__':
    setLogLevel( 'info' )
    myNetwork()
```

A.5 QOS Simple Switch

```
# Copyright (C) 2011 Nippon Telegraph and Telephone Corporation.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#    http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
# implied.
# See the License for the specific language governing permissions and
# limitations under the License.

from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_3
from ryu.lib.packet import packet
```

```python
from ryu.lib.packet import ethernet
from ryu.lib.packet import ether_types

import subprocess


class SimpleSwitch13(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    def __init__(self, *args, **kwargs):
        super(SimpleSwitch13, self).__init__(*args, **kwargs)
        self.mac_to_port = {}

    @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
    def switch_features_handler(self, ev):
        datapath = ev.msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        # install table-miss flow entry
        #
        # We specify NO BUFFER to max_len of the output action due to
        # OVS bug. At this moment, if we specify a lesser number, e.g.,
        # 128, OVS will send Packet-In with invalid buffer_id and
        # truncated packet data. In that case, we cannot output packets
        # correctly.  The bug has been fixed in OVS v2.1.0.
        match = parser.OFPMatch()
        actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
                                          ofproto.OFPCML_NO_BUFFER)]
        self.add_flow(datapath, 0, match, actions)

    def add_flow(self, datapath, priority, match, actions, buffer_id=None):
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
                                             actions)]
        if buffer_id:
            mod = parser.OFPFlowMod(datapath=datapath, buffer_id=buffer_id,
                                    priority=priority, match=match,
                                    instructions=inst, table_id=1)
        else:
            mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
                                    match=match, instructions=inst, table_id=1)
        datapath.send_msg(mod)

    @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
    def _packet_in_handler(self, ev):
        # If you hit this you might want to increase
```

```
        # the "miss_send_length" of your switch
        if ev.msg.msg_len < ev.msg.total_len:
            self.logger.debug("packet truncated: only %s of %s bytes",
                              ev.msg.msg_len, ev.msg.total_len)
        msg = ev.msg
        datapath = msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
        in_port = msg.match['in_port']

        pkt = packet.Packet(msg.data)
        eth = pkt.get_protocols(ethernet.ethernet)[0]

        if eth.ethertype == ether_types.ETH_TYPE_LLDP:
            # ignore lldp packet
            return
        dst = eth.dst
        src = eth.src

        dpid = datapath.id
        self.mac_to_port.setdefault(dpid, {})

##         self.logger.info("packet in %s %s %s %s", dpid, src, dst, in_port)

        # learn a mac address to avoid FLOOD next time.
        self.mac_to_port[dpid][src] = in_port

        if dst in self.mac_to_port[dpid]:
            out_port = self.mac_to_port[dpid][dst]
        else:
            out_port = ofproto.OFPP_FLOOD

        actions = [parser.OFPActionOutput(out_port)]

        # install a flow to avoid packet_in next time
        if out_port != ofproto.OFPP_FLOOD:
            match = parser.OFPMatch(in_port=in_port, eth_dst=dst, eth_src=src)
            # verify if we have a valid buffer_id, if yes avoid to send both
            # flow_mod & packet_out
            if msg.buffer_id != ofproto.OFP_NO_BUFFER:
                self.add_flow(datapath, 1, match, actions, msg.buffer_id)
                return
            else:
                self.add_flow(datapath, 1, match, actions)
        data = None
        if msg.buffer_id == ofproto.OFP_NO_BUFFER:
            data = msg.data

        out = parser.OFPPacketOut(datapath=datapath, buffer_id=msg.buffer_id,
```

```
                                   in_port=in_port, actions=actions, data
    =data)
        datapath.send_msg(out)
```

A.6 ofctl rest API

```
# Copyright (C) 2012 Nippon Telegraph and Telephone Corporation.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#    http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
# implied.
# See the License for the specific language governing permissions and
# limitations under the License.

import logging
import json
import ast

from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller import dpset
from ryu.controller.handler import MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.exception import RyuException
from ryu.ofproto import ofproto_v1_0
from ryu.ofproto import ofproto_v1_2
from ryu.ofproto import ofproto_v1_3
from ryu.ofproto import ofproto_v1_4
from ryu.ofproto import ofproto_v1_5
from ryu.lib import ofctl_v1_0
from ryu.lib import ofctl_v1_2
from ryu.lib import ofctl_v1_3
from ryu.lib import ofctl_v1_4
from ryu.lib import ofctl_v1_5
from ryu.app.wsgi import ControllerBase
from ryu.app.wsgi import Response
from ryu.app.wsgi import WSGIApplication

LOG = logging.getLogger('ryu.app.ofctl_rest')

# supported ofctl versions in this restful app
supported_ofctl = {
    ofproto_v1_0.OFP_VERSION: ofctl_v1_0,
    ofproto_v1_2.OFP_VERSION: ofctl_v1_2,
```

```
    ofproto_v1_3.OFP_VERSION: ofctl_v1_3,
    ofproto_v1_4.OFP_VERSION: ofctl_v1_4,
    ofproto_v1_5.OFP_VERSION: ofctl_v1_5,
}

# REST API
#

# Retrieve the switch stats
#
# get the list of all switches
# GET /stats/switches
#
# get the desc stats of the switch
# GET /stats/desc/<dpid>
#
# get flows desc stats of the switch
# GET /stats/flowdesc/<dpid>
#
# get flows desc stats of the switch filtered by the fields
# POST /stats/flowdesc/<dpid>
#
# get flows stats of the switch
# GET /stats/flow/<dpid>
#
# get flows stats of the switch filtered by the fields
# POST /stats/flow/<dpid>
#
# get aggregate flows stats of the switch
# GET /stats/aggregateflow/<dpid>
#
# get aggregate flows stats of the switch filtered by the fields
# POST /stats/aggregateflow/<dpid>
#
# get table stats of the switch
# GET /stats/table/<dpid>
#
# get table features stats of the switch
# GET /stats/tablefeatures/<dpid>
#
# get ports stats of the switch
# GET /stats/port/<dpid>[/<port>]
# Note: Specification of port number is optional
#
# get queues stats of the switch
# GET /stats/queue/<dpid>[/<port>[/<queue_id>]]
# Note: Specification of port number and queue id are optional
#       If you want to omitting the port number and setting the queue id
  ,
#       please specify the keyword "ALL" to the port number
#       e.g. GET /stats/queue/1/ALL/1
#
# get queues config stats of the switch
# GET /stats/queueconfig/<dpid>[/<port>]
```

```
# Note: Specification of port number is optional
#
# get queues desc stats of the switch
# GET /stats/queuedesc/<dpid>[/<port>[/<queue_id>]]
# Note: Specification of port number and queue id are optional
#       If you want to omitting the port number and setting the queue id
    ,
#       please specify the keyword "ALL" to the port number
#       e.g. GET /stats/queuedesc/1/ALL/1
#
# get meter features stats of the switch
# GET /stats/meterfeatures/<dpid>
#
# get meter config stats of the switch
# GET /stats/meterconfig/<dpid>[/<meter_id>]
# Note: Specification of meter id is optional
#
# get meter desc stats of the switch
# GET /stats/meterdesc/<dpid>[/<meter_id>]
# Note: Specification of meter id is optional
#
# get meters stats of the switch
# GET /stats/meter/<dpid>[/<meter_id>]
# Note: Specification of meter id is optional
#
# get group features stats of the switch
# GET /stats/groupfeatures/<dpid>
#
# get groups desc stats of the switch
# GET /stats/groupdesc/<dpid>[/<group_id>]
# Note: Specification of group id is optional (OpenFlow 1.5 or later)
#
# get groups stats of the switch
# GET /stats/group/<dpid>[/<group_id>]
# Note: Specification of group id is optional
#
# get ports description of the switch
# GET /stats/portdesc/<dpid>[/<port_no>]
# Note: Specification of port number is optional (OpenFlow 1.5 or later)

# Update the switch stats
#
# add a flow entry
# POST /stats/flowentry/add
#
# modify all matching flow entries
# POST /stats/flowentry/modify
#
# modify flow entry strictly matching wildcards and priority
# POST /stats/flowentry/modify_strict
#
# delete all matching flow entries
# POST /stats/flowentry/delete
#
```

```
# delete flow entry strictly matching wildcards and priority
# POST /stats/flowentry/delete_strict
#
# delete all flow entries of the switch
# DELETE /stats/flowentry/clear/<dpid>
#
# add a meter entry
# POST /stats/meterentry/add
#
# modify a meter entry
# POST /stats/meterentry/modify
#
# delete a meter entry
# POST /stats/meterentry/delete
#
# add a group entry
# POST /stats/groupentry/add
#
# modify a group entry
# POST /stats/groupentry/modify
#
# delete a group entry
# POST /stats/groupentry/delete
#
# modify behavior of the physical port
# POST /stats/portdesc/modify
#
# modify role of controller
# POST /stats/role
#
#
# send a experimeter message
# POST /stats/experimenter/<dpid>


class CommandNotFoundError(RyuException):
    message = 'No such command : %(cmd)s'


class PortNotFoundError(RyuException):
    message = 'No such port info: %(port_no)s'


def stats_method(method):
    def wrapper(self, req, dpid, *args, **kwargs):
        # Get datapath instance from DPSet
        try:
            dp = self.dpset.get(int(str(dpid), 0))
        except ValueError:
            LOG.exception('Invalid dpid: %s', dpid)
            return Response(status=400)
        if dp is None:
            LOG.error('No such Datapath: %s', dpid)
            return Response(status=404)
```

```

        # Get lib/ofctl_* module
        try:
            ofctl = supported_ofctl.get(dp.ofproto.OFP_VERSION)
        except KeyError:
            LOG.exception('Unsupported OF version: %s',
                          dp.ofproto.OFP_VERSION)
            return Response(status=501)

        # Invoke StatsController method
        try:
            ret = method(self, req, dp, ofctl, *args, **kwargs)
            return Response(content_type='application/json',
                            body=json.dumps(ret))
        except ValueError:
            LOG.exception('Invalid syntax: %s', req.body)
            return Response(status=400)
        except AttributeError:
            LOG.exception('Unsupported OF request in this version: %s',
                          dp.ofproto.OFP_VERSION)
            return Response(status=501)

    return wrapper


def command_method(method):
    def wrapper(self, req, *args, **kwargs):
        # Parse request json body
        try:
            if req.body:
                # We use ast.literal_eval() to parse request json body
                # instead of json.loads().
                # Because we need to parse binary format body
                # in send_experimenter().
                body = ast.literal_eval(req.body.decode('utf-8'))
            else:
                body = {}
        except SyntaxError:
            LOG.exception('Invalid syntax: %s', req.body)
            return Response(status=400)

        # Get datapath_id from request parameters
        dpid = body.get('dpid', None)
        if not dpid:
            try:
                dpid = kwargs.pop('dpid')
            except KeyError:
                LOG.exception('Cannot get dpid from request parameters')
                return Response(status=400)

        # Get datapath instance from DPSet
        try:
            dp = self.dpset.get(int(str(dpid), 0))
        except ValueError:
```

```
            LOG.exception('Invalid dpid: %s', dpid)
            return Response(status=400)
        if dp is None:
            LOG.error('No such Datapath: %s', dpid)
            return Response(status=404)

        # Get lib/ofctl_* module
        try:
            ofctl = supported_ofctl.get(dp.ofproto.OFP_VERSION)
        except KeyError:
            LOG.exception('Unsupported OF version: version=%s',
                          dp.ofproto.OFP_VERSION)
            return Response(status=501)

        # Invoke StatsController method
        try:
            method(self, req, dp, ofctl, body, *args, **kwargs)
            return Response(status=200)
        except ValueError:
            LOG.exception('Invalid syntax: %s', req.body)
            return Response(status=400)
        except AttributeError:
            LOG.exception('Unsupported OF request in this version: %s',
                          dp.ofproto.OFP_VERSION)
            return Response(status=501)
        except CommandNotFoundError as e:
            LOG.exception(e.message)
            return Response(status=404)
        except PortNotFoundError as e:
            LOG.exception(e.message)
            return Response(status=404)

    return wrapper


class StatsController(ControllerBase):
    def __init__(self, req, link, data, **config):
        super(StatsController, self).__init__(req, link, data, **config)
        self.dpset = data['dpset']
        self.waiters = data['waiters']

    def get_dpids(self, req, **_kwargs):
        dps = list(self.dpset.dps.keys())
        body = json.dumps(dps)
        return Response(content_type='application/json', body=body)

    @stats_method
    def get_desc_stats(self, req, dp, ofctl, **kwargs):
        return ofctl.get_desc_stats(dp, self.waiters)

    @stats_method
    def get_flow_desc(self, req, dp, ofctl, **kwargs):
        flow = req.json if req.body else {}
        return ofctl.get_flow_desc(dp, self.waiters, flow)
```

```python

    @stats_method
    def get_flow_stats(self, req, dp, ofctl, **kwargs):
        flow = req.json if req.body else {}
        return ofctl.get_flow_stats(dp, self.waiters, flow)

    @stats_method
    def get_aggregate_flow_stats(self, req, dp, ofctl, **kwargs):
        flow = req.json if req.body else {}
        return ofctl.get_aggregate_flow_stats(dp, self.waiters, flow)

    @stats_method
    def get_table_stats(self, req, dp, ofctl, **kwargs):
        return ofctl.get_table_stats(dp, self.waiters)

    @stats_method
    def get_table_features(self, req, dp, ofctl, **kwargs):
        return ofctl.get_table_features(dp, self.waiters)

    @stats_method
    def get_port_stats(self, req, dp, ofctl, port=None, **kwargs):
        if port == "ALL":
            port = None

        return ofctl.get_port_stats(dp, self.waiters, port)

    @stats_method
    def get_queue_stats(self, req, dp, ofctl,
                        port=None, queue_id=None, **kwargs):
        if port == "ALL":
            port = None

        if queue_id == "ALL":
            queue_id = None

        return ofctl.get_queue_stats(dp, self.waiters, port, queue_id)

    @stats_method
    def get_queue_config(self, req, dp, ofctl, port=None, **kwargs):
        if port == "ALL":
            port = None

        return ofctl.get_queue_config(dp, self.waiters, port)

    @stats_method
    def get_queue_desc(self, req, dp, ofctl,
                       port=None, queue=None, **_kwargs):
        if port == "ALL":
            port = None

        if queue == "ALL":
            queue = None

        return ofctl.get_queue_desc(dp, self.waiters, port, queue)
```

```python

    @stats_method
    def get_meter_features(self, req, dp, ofctl, **kwargs):
        return ofctl.get_meter_features(dp, self.waiters)

    @stats_method
    def get_meter_config(self, req, dp, ofctl, meter_id=None, **kwargs):
        if meter_id == "ALL":
            meter_id = None

        return ofctl.get_meter_config(dp, self.waiters, meter_id)

    @stats_method
    def get_meter_desc(self, req, dp, ofctl, meter_id=None, **kwargs):
        if meter_id == "ALL":
            meter_id = None

        return ofctl.get_meter_desc(dp, self.waiters, meter_id)

    @stats_method
    def get_meter_stats(self, req, dp, ofctl, meter_id=None, **kwargs):
        if meter_id == "ALL":
            meter_id = None

        return ofctl.get_meter_stats(dp, self.waiters, meter_id)

    @stats_method
    def get_group_features(self, req, dp, ofctl, **kwargs):
        return ofctl.get_group_features(dp, self.waiters)

    @stats_method
    def get_group_desc(self, req, dp, ofctl, group_id=None, **kwargs):
        if dp.ofproto.OFP_VERSION < ofproto_v1_5.OFP_VERSION:
            return ofctl.get_group_desc(dp, self.waiters)
        else:
            return ofctl.get_group_desc(dp, self.waiters, group_id)

    @stats_method
    def get_group_stats(self, req, dp, ofctl, group_id=None, **kwargs):
        if group_id == "ALL":
            group_id = None

        return ofctl.get_group_stats(dp, self.waiters, group_id)

    @stats_method
    def get_port_desc(self, req, dp, ofctl, port_no=None, **kwargs):
        if dp.ofproto.OFP_VERSION < ofproto_v1_5.OFP_VERSION:
            return ofctl.get_port_desc(dp, self.waiters)
        else:
            return ofctl.get_port_desc(dp, self.waiters, port_no)

    @stats_method
    def get_role(self, req, dp, ofctl, **kwargs):
        return ofctl.get_role(dp, self.waiters)
```

```
    @command_method
    def mod_flow_entry(self, req, dp, ofctl, flow, cmd, **kwargs):
        cmd_convert = {
            'add': dp.ofproto.OFPFC_ADD,
            'modify': dp.ofproto.OFPFC_MODIFY,
            'modify_strict': dp.ofproto.OFPFC_MODIFY_STRICT,
            'delete': dp.ofproto.OFPFC_DELETE,
            'delete_strict': dp.ofproto.OFPFC_DELETE_STRICT,
        }
        mod_cmd = cmd_convert.get(cmd, None)
        if mod_cmd is None:
            raise CommandNotFoundError(cmd=cmd)

        ofctl.mod_flow_entry(dp, flow, mod_cmd)

    @command_method
    def delete_flow_entry(self, req, dp, ofctl, flow, **kwargs):
        if ofproto_v1_0.OFP_VERSION == dp.ofproto.OFP_VERSION:
            flow = {}
        else:
            flow = {'table_id': dp.ofproto.OFPTT_ALL}

        ofctl.mod_flow_entry(dp, flow, dp.ofproto.OFPFC_DELETE)

    @command_method
    def mod_meter_entry(self, req, dp, ofctl, meter, cmd, **kwargs):
        cmd_convert = {
            'add': dp.ofproto.OFPMC_ADD,
            'modify': dp.ofproto.OFPMC_MODIFY,
            'delete': dp.ofproto.OFPMC_DELETE,
        }
        mod_cmd = cmd_convert.get(cmd, None)
        if mod_cmd is None:
            raise CommandNotFoundError(cmd=cmd)

        ofctl.mod_meter_entry(dp, meter, mod_cmd)

    @command_method
    def mod_group_entry(self, req, dp, ofctl, group, cmd, **kwargs):
        cmd_convert = {
            'add': dp.ofproto.OFPGC_ADD,
            'modify': dp.ofproto.OFPGC_MODIFY,
            'delete': dp.ofproto.OFPGC_DELETE,
        }
        mod_cmd = cmd_convert.get(cmd, None)
        if mod_cmd is None:
            raise CommandNotFoundError(cmd=cmd)

        ofctl.mod_group_entry(dp, group, mod_cmd)

    @command_method
    def mod_port_behavior(self, req, dp, ofctl, port_config, cmd, **
kwargs):
```

```
        port_no = port_config.get('port_no', None)
        port_no = int(str(port_no), 0)

        port_info = self.dpset.port_state[int(dp.id)].get(port_no)
        if port_info:
            port_config.setdefault('hw_addr', port_info.hw_addr)
            if dp.ofproto.OFP_VERSION < ofproto_v1_4.OFP_VERSION:
                port_config.setdefault('advertise', port_info.advertised)
            else:
                port_config.setdefault('properties', port_info.properties)
        else:
            raise PortNotFoundError(port_no=port_no)

        if cmd != 'modify':
            raise CommandNotFoundError(cmd=cmd)

        ofctl.mod_port_behavior(dp, port_config)

    @command_method
    def send_experimenter(self, req, dp, ofctl, exp, **kwargs):
        ofctl.send_experimenter(dp, exp)

    @command_method
    def set_role(self, req, dp, ofctl, role, **kwargs):
        ofctl.set_role(dp, role)


class RestStatsApi(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_0.OFP_VERSION,
                    ofproto_v1_2.OFP_VERSION,
                    ofproto_v1_3.OFP_VERSION,
                    ofproto_v1_4.OFP_VERSION,
                    ofproto_v1_5.OFP_VERSION]
    _CONTEXTS = {
        'dpset': dpset.DPSet,
        'wsgi': WSGIApplication
    }

    def __init__(self, *args, **kwargs):
        super(RestStatsApi, self).__init__(*args, **kwargs)
        self.dpset = kwargs['dpset']
        wsgi = kwargs['wsgi']
        self.waiters = {}
        self.data = {}
        self.data['dpset'] = self.dpset
        self.data['waiters'] = self.waiters
        mapper = wsgi.mapper

        wsgi.registory['StatsController'] = self.data
        path = '/stats'
        uri = path + '/switches'
        mapper.connect('stats', uri,
```

```
                        controller=StatsController, action='get_dpids',
                        conditions=dict(method=['GET']))

        uri = path + '/desc/{dpid}'
        mapper.connect('stats', uri,
                        controller=StatsController, action='get_desc_stats',
                        conditions=dict(method=['GET']))

        uri = path + '/flowdesc/{dpid}'
        mapper.connect('stats', uri,
                        controller=StatsController, action='get_flow_stats',
                        conditions=dict(method=['GET', 'POST']))

        uri = path + '/flow/{dpid}'
        mapper.connect('stats', uri,
                        controller=StatsController, action='get_flow_stats',
                        conditions=dict(method=['GET', 'POST']))

        uri = path + '/aggregateflow/{dpid}'
        mapper.connect('stats', uri,
                        controller=StatsController,
                        action='get_aggregate_flow_stats',
                        conditions=dict(method=['GET', 'POST']))

        uri = path + '/table/{dpid}'
        mapper.connect('stats', uri,
                        controller=StatsController, action='get_table_stats',
                        conditions=dict(method=['GET']))

        uri = path + '/tablefeatures/{dpid}'
        mapper.connect('stats', uri,
                        controller=StatsController, action='get_table_features',
                        conditions=dict(method=['GET']))

        uri = path + '/port/{dpid}'
        mapper.connect('stats', uri,
                        controller=StatsController, action='get_port_stats',
                        conditions=dict(method=['GET']))

        uri = path + '/port/{dpid}/{port}'
        mapper.connect('stats', uri,
                        controller=StatsController, action='get_port_stats',
                        conditions=dict(method=['GET']))

        uri = path + '/queue/{dpid}'
        mapper.connect('stats', uri,
```

```
                             controller=StatsController, action='get_queue_stats',
                             conditions=dict(method=['GET']))

        uri = path + '/queue/{dpid}/{port}'
        mapper.connect('stats', uri,
                             controller=StatsController, action='get_queue_stats',
                             conditions=dict(method=['GET']))

        uri = path + '/queue/{dpid}/{port}/{queue_id}'
        mapper.connect('stats', uri,
                             controller=StatsController, action='get_queue_stats',
                             conditions=dict(method=['GET']))

        uri = path + '/queueconfig/{dpid}'
        mapper.connect('stats', uri,
                             controller=StatsController, action='get_queue_config',
                             conditions=dict(method=['GET']))

        uri = path + '/queueconfig/{dpid}/{port}'
        mapper.connect('stats', uri,
                             controller=StatsController, action='get_queue_config',
                             conditions=dict(method=['GET']))

        uri = path + '/queuedesc/{dpid}'
        mapper.connect('stats', uri,
                             controller=StatsController, action='get_queue_desc',
                             conditions=dict(method=['GET']))

        uri = path + '/queuedesc/{dpid}/{port}'
        mapper.connect('stats', uri,
                             controller=StatsController, action='get_queue_desc',
                             conditions=dict(method=['GET']))

        uri = path + '/queuedesc/{dpid}/{port}/{queue}'
        mapper.connect('stats', uri,
                             controller=StatsController, action='get_queue_desc',
                             conditions=dict(method=['GET']))

        uri = path + '/meterfeatures/{dpid}'
        mapper.connect('stats', uri,
                             controller=StatsController, action='get_meter_features',
                             conditions=dict(method=['GET']))

        uri = path + '/meterconfig/{dpid}'
        mapper.connect('stats', uri,
```

```
                            controller=StatsController, action='get_meter_config',
                            conditions=dict(method=['GET']))

        uri = path + '/meterconfig/{dpid}/{meter_id}'
        mapper.connect('stats', uri,
                            controller=StatsController, action='get_meter_config',
                            conditions=dict(method=['GET']))

        uri = path + '/meterdesc/{dpid}'
        mapper.connect('stats', uri,
                            controller=StatsController, action='get_meter_desc',
                            conditions=dict(method=['GET']))

        uri = path + '/meterdesc/{dpid}/{meter_id}'
        mapper.connect('stats', uri,
                            controller=StatsController, action='get_meter_desc',
                            conditions=dict(method=['GET']))

        uri = path + '/meter/{dpid}'
        mapper.connect('stats', uri,
                            controller=StatsController, action='get_meter_stats',
                            conditions=dict(method=['GET']))

        uri = path + '/meter/{dpid}/{meter_id}'
        mapper.connect('stats', uri,
                            controller=StatsController, action='get_meter_stats',
                            conditions=dict(method=['GET']))

        uri = path + '/groupfeatures/{dpid}'
        mapper.connect('stats', uri,
                            controller=StatsController, action='get_group_features',
                            conditions=dict(method=['GET']))

        uri = path + '/groupdesc/{dpid}'
        mapper.connect('stats', uri,
                            controller=StatsController, action='get_group_desc',
                            conditions=dict(method=['GET']))

        uri = path + '/groupdesc/{dpid}/{group_id}'
        mapper.connect('stats', uri,
                            controller=StatsController, action='get_group_desc',
                            conditions=dict(method=['GET']))

        uri = path + '/group/{dpid}'
        mapper.connect('stats', uri,
```

```
                                controller=StatsController, action='get_group_stats',
                                conditions=dict(method=['GET']))

        uri = path + '/group/{dpid}/{group_id}'
        mapper.connect('stats', uri,
                                controller=StatsController, action='get_group_stats',
                                conditions=dict(method=['GET']))

        uri = path + '/portdesc/{dpid}'
        mapper.connect('stats', uri,
                                controller=StatsController, action='get_port_desc',
                                conditions=dict(method=['GET']))

        uri = path + '/portdesc/{dpid}/{port_no}'
        mapper.connect('stats', uri,
                                controller=StatsController, action='get_port_desc',
                                conditions=dict(method=['GET']))

        uri = path + '/role/{dpid}'
        mapper.connect('stats', uri,
                                controller=StatsController, action='get_role',
                                conditions=dict(method=['GET']))

        uri = path + '/flowentry/{cmd}'
        mapper.connect('stats', uri,
                                controller=StatsController, action='mod_flow_entry',
                                conditions=dict(method=['POST']))

        uri = path + '/flowentry/clear/{dpid}'
        mapper.connect('stats', uri,
                                controller=StatsController, action='delete_flow_entry',
                                conditions=dict(method=['DELETE']))

        uri = path + '/meterentry/{cmd}'
        mapper.connect('stats', uri,
                                controller=StatsController, action='mod_meter_entry',
                                conditions=dict(method=['POST']))

        uri = path + '/groupentry/{cmd}'
        mapper.connect('stats', uri,
                                controller=StatsController, action='mod_group_entry',
                                conditions=dict(method=['POST']))

        uri = path + '/portdesc/{cmd}'
        mapper.connect('stats', uri,
```

```
                                controller=StatsController, action='
    mod_port_behavior',
                                conditions=dict(method=['POST']))

        uri = path + '/experimenter/{dpid}'
        mapper.connect('stats', uri,
                                controller=StatsController, action='
    send_experimenter',
                                conditions=dict(method=['POST']))

        uri = path + '/role'
        mapper.connect('stats', uri,
                                controller=StatsController, action='set_role',
                                conditions=dict(method=['POST']))

    @set_ev_cls([ofp_event.EventOFPStatsReply,
                 ofp_event.EventOFPDescStatsReply,
                 ofp_event.EventOFPFlowStatsReply,
                 ofp_event.EventOFPAggregateStatsReply,
                 ofp_event.EventOFPTableStatsReply,
                 ofp_event.EventOFPTableFeaturesStatsReply,
                 ofp_event.EventOFPPortStatsReply,
                 ofp_event.EventOFPQueueStatsReply,
                 ofp_event.EventOFPQueueDescStatsReply,
                 ofp_event.EventOFPMeterStatsReply,
                 ofp_event.EventOFPMeterFeaturesStatsReply,
                 ofp_event.EventOFPMeterConfigStatsReply,
                 ofp_event.EventOFPGroupStatsReply,
                 ofp_event.EventOFPGroupFeaturesStatsReply,
                 ofp_event.EventOFPGroupDescStatsReply,
                 ofp_event.EventOFPPortDescStatsReply
                 ], MAIN_DISPATCHER)
    def stats_reply_handler(self, ev):
        msg = ev.msg
        dp = msg.datapath

        if dp.id not in self.waiters:
            return
        if msg.xid not in self.waiters[dp.id]:
            return
        lock, msgs = self.waiters[dp.id][msg.xid]
        msgs.append(msg)

        flags = 0
        if dp.ofproto.OFP_VERSION == ofproto_v1_0.OFP_VERSION:
            flags = dp.ofproto.OFPSF_REPLY_MORE
        elif dp.ofproto.OFP_VERSION == ofproto_v1_2.OFP_VERSION:
            flags = dp.ofproto.OFPSF_REPLY_MORE
        elif dp.ofproto.OFP_VERSION >= ofproto_v1_3.OFP_VERSION:
            flags = dp.ofproto.OFPMPF_REPLY_MORE

        if msg.flags & flags:
            return
        del self.waiters[dp.id][msg.xid]
```

```
        lock.set()

    @set_ev_cls([ofp_event.EventOFPSwitchFeatures,
                 ofp_event.EventOFPQueueGetConfigReply,
                 ofp_event.EventOFPRoleReply,
                 ], MAIN_DISPATCHER)
    def features_reply_handler(self, ev):
        msg = ev.msg
        dp = msg.datapath

        if dp.id not in self.waiters:
            return
        if msg.xid not in self.waiters[dp.id]:
            return
        lock, msgs = self.waiters[dp.id][msg.xid]
        msgs.append(msg)

        del self.waiters[dp.id][msg.xid]
        lock.set()
```

A.7 rest conf switch

```
# Copyright (C) 2012 Nippon Telegraph and Telephone Corporation.
# Copyright (C) 2012 Isaku Yamahata <yamahata at private email ne jp>
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#    http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
# implied.
# See the License for the specific language governing permissions and
# limitations under the License.

"""
This module provides a set of REST API for switch configuration.
- Per-switch Key-Value store

Used by OpenStack Ryu agent.
"""

import json

from six.moves import http_client

from ryu.app.wsgi import ControllerBase
from ryu.app.wsgi import Response
```

```python
from ryu.base import app_manager
from ryu.controller import conf_switch
from ryu.lib import dpid as dpid_lib


# REST API for switch configuration
#
# get all the switches
# GET /v1.0/conf/switches
#
# get all the configuration keys of a switch
# GET /v1.0/conf/switches/<dpid>
#
# delete all the configuration of a switch
# DELETE /v1.0/conf/switches/<dpid>
#
# set the <key> configuration of a switch
# PUT /v1.0/conf/switches/<dpid>/<key>
#
# get the <key> configuration of a switch
# GET /v1.0/conf/switches/<dpid>/<key>
#
# delete the <key> configuration of a switch
# DELETE /v1.0/conf/switches/<dpid>/<key>
#
# where
# <dpid>: datapath id in 16 hex


class ConfSwitchController(ControllerBase):
    def __init__(self, req, link, data, **config):
        super(ConfSwitchController, self).__init__(req, link, data, **
    config)
        self.conf_switch = data

    def list_switches(self, _req, **_kwargs):
        dpids = self.conf_switch.dpids()
        body = json.dumps([dpid_lib.dpid_to_str(dpid) for dpid in dpids
    ])
        return Response(content_type='application/json', body=body)

    @staticmethod
    def _do_switch(dpid, func, ret_func):
        dpid = dpid_lib.str_to_dpid(dpid)
        try:
            ret = func(dpid)
        except KeyError:
            return Response(status=http_client.NOT_FOUND,
                            body='no dpid is found %s' %
                            dpid_lib.dpid_to_str(dpid))

        return ret_func(ret)

    def delete_switch(self, _req, dpid, **_kwargs):
```

```
        def _delete_switch(dpid):
            self.conf_switch.del_dpid(dpid)
            return None

        def _ret(_ret):
            return Response(status=http_client.ACCEPTED)

        return self._do_switch(dpid, _delete_switch, _ret)

    def list_keys(self, _req, dpid, **_kwargs):
        def _list_keys(dpid):
            return self.conf_switch.keys(dpid)

        def _ret(keys):
            body = json.dumps(keys)
            return Response(content_type='application/json', body=body)

        return self._do_switch(dpid, _list_keys, _ret)

    @staticmethod
    def _do_key(dpid, key, func, ret_func):
        dpid = dpid_lib.str_to_dpid(dpid)
        try:
            ret = func(dpid, key)
        except KeyError:
            return Response(status=http_client.NOT_FOUND,
                            body='no dpid/key is found %s %s' %
                            (dpid_lib.dpid_to_str(dpid), key))
        return ret_func(ret)

    def set_key(self, req, dpid, key, **_kwargs):
        def _set_val(dpid, key):
            try:
                val = req.json if req.body else {}
            except ValueError:
                return Response(status=http_client.BAD_REQUEST,
                                body='invalid syntax %s' % req.body)
            self.conf_switch.set_key(dpid, key, val)
            return None

        def _ret(_ret):
            return Response(status=http_client.CREATED)

        return self._do_key(dpid, key, _set_val, _ret)

    def get_key(self, _req, dpid, key, **_kwargs):
        def _get_key(dpid, key):
            return self.conf_switch.get_key(dpid, key)

        def _ret(val):
            return Response(content_type='application/json',
                            body=json.dumps(val))

        return self._do_key(dpid, key, _get_key, _ret)
```

```python

    def delete_key(self, _req, dpid, key, **_kwargs):
        def _delete_key(dpid, key):
            self.conf_switch.del_key(dpid, key)
            return None

        def _ret(_ret):
            return Response()

        return self._do_key(dpid, key, _delete_key, _ret)


class ConfSwitchAPI(app_manager.RyuApp):
    _CONTEXTS = {
        'conf_switch': conf_switch.ConfSwitchSet,
    }

    def __init__(self, *args, **kwargs):
        super(ConfSwitchAPI, self).__init__(*args, **kwargs)
        self.conf_switch = kwargs['conf_switch']
        wsgi = kwargs['wsgi']
        mapper = wsgi.mapper

        controller = ConfSwitchController
        wsgi.registory[controller.__name__] = self.conf_switch
        route_name = 'conf_switch'
        uri = '/v1.0/conf/switches'
        mapper.connect(route_name, uri, controller=controller,
                       action='list_switches',
                       conditions=dict(method=['GET']))

        uri += '/{dpid}'
        requirements = {'dpid': dpid_lib.DPID_PATTERN}
        s = mapper.submapper(controller=controller, requirements=
    requirements)
        s.connect(route_name, uri, action='delete_switch',
                  conditions=dict(method=['DELETE']))
        s.connect(route_name, uri, action='list_keys',
                  conditions=dict(method=['GET']))

        uri += '/{key}'
        s.connect(route_name, uri, action='set_key',
                  conditions=dict(method=['PUT']))
        s.connect(route_name, uri, action='get_key',
                  conditions=dict(method=['GET']))
        s.connect(route_name, uri, action='delete_key',
                  conditions=dict(method=['DELETE']))
```

A.8 rest qos

```python
# Copyright (C) 2014 Kiyonari Harigae <lakshmi at cloudysunny14 org>
```

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


import logging
import json
import re

from ryu.app import conf_switch_key as cs_key
from ryu.app.wsgi import ControllerBase
from ryu.app.wsgi import Response
from ryu.app.wsgi import route
from ryu.app.wsgi import WSGIApplication
from ryu.base import app_manager
from ryu.controller import conf_switch
from ryu.controller import ofp_event
from ryu.controller import dpset
from ryu.controller.handler import set_ev_cls
from ryu.controller.handler import MAIN_DISPATCHER
from ryu.exception import OFPUnknownVersion
from ryu.lib import dpid as dpid_lib
from ryu.lib import mac
from ryu.lib import ofctl_v1_0
from ryu.lib import ofctl_v1_2
from ryu.lib import ofctl_v1_3
from ryu.lib.ovs import bridge
from ryu.ofproto import ofproto_v1_0
from ryu.ofproto import ofproto_v1_2
from ryu.ofproto import ofproto_v1_3
from ryu.ofproto import ofproto_v1_3_parser
from ryu.ofproto import ether
from ryu.ofproto import inet


# =============================
#          REST API
# =============================
#
#  Note: specify switch and vlan group, as follows.
#   {switch-id} : 'all' or switchID
#   {vlan-id}   : 'all' or vlanID
#
# about queue status
```

```
#
# get status of queue
# GET /qos/queue/status/{switch-id}
#
# about queues
# get a queue configurations
# GET /qos/queue/{switch-id}
#
# set a queue to the switches
# POST /qos/queue/{switch-id}
#
# request body format:
#  {"port_name":"<name of port>",
#   "type": "<linux-htb or linux-other>",
#   "max-rate": "<int>",
#   "queues":[{"max_rate": "<int>", "min_rate": "<int>"},...]}
#
#   Note: This operation override
#         previous configurations.
#   Note: Queue configurations are available for
#         OpenvSwitch.
#   Note: port_name is optional argument.
#         If does not pass the port_name argument,
#         all ports are target for configuration.
#
# delete queue
# DELETE /qos/queue/{swtich-id}
#
#   Note: This operation delete relation of qos record from
#         qos colum in Port table. Therefore,
#         QoS records and Queue records will remain.
#
# about qos rules
#
# get rules of qos
# * for no vlan
# GET /qos/rules/{switch-id}
#
# * for specific vlan group
# GET /qos/rules/{switch-id}/{vlan-id}
#
# set a qos rules
#
#   QoS rules will do the processing pipeline,
#   which entries are register the first table (by default table id 0)
#   and process will apply and go to next table.
#
# * for no vlan
# POST /qos/{switch-id}
#
# * for specific vlan group
# POST /qos/{switch-id}/{vlan-id}
#
#  request body format:
```

```
#    {"priority": "<value>",
#     "match": {"<field1>": "<value1>", "<field2>": "<value2>",...},
#     "actions": {"<action1>": "<value1>", "<action2>": "<value2>",...}
#    }
#
#   Description
#     * priority field
#      <value>
#     "0 to 65533"
#
#    Note: When "priority" has not been set up,
#          "priority: 1" is set to "priority".
#
#     * match field
#      <field> : <value>
#     "in_port" : "<int>"
#     "dl_src"  : "<xx:xx:xx:xx:xx:xx>"
#     "dl_dst"  : "<xx:xx:xx:xx:xx:xx>"
#     "dl_type" : "<ARP or IPv4 or IPv6>"
#     "nw_src"  : "<A.B.C.D/M>"
#     "nw_dst"  : "<A.B.C.D/M>"
#     "ipv6_src": "<xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx/M>"
#     "ipv6_dst": "<xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx/M>"
#     "nw_proto": "<TCP or UDP or ICMP or ICMPv6>"
#     "tp_src"  : "<int>"
#     "tp_dst"  : "<int>"
#     "ip_dscp" : "<int>"
#
#     * actions field
#      <field> : <value>
#     "mark": <dscp-value>
#     sets the IPv4 ToS/DSCP field to tos.
#     "meter": <meter-id>
#     apply meter entry
#     "queue": <queue-id>
#     register queue specified by queue-id
#
#    Note: When "actions" has not been set up,
#          "queue: 0" is set to "actions".
#
# delete a qos rules
# * for no vlan
# DELETE /qos/rule/{switch-id}
#
# * for specific vlan group
# DELETE /qos/{switch-id}/{vlan-id}
#
#   request body format:
#    {"<field>":"<value>"}
#
#      <field>  : <value>
#     "qos_id" : "<int>" or "all"
#
# about meter entries
```

```
#
# set a meter entry
# POST /qos/meter/{switch-id}
#
#  request body format:
#   {"meter_id": <int>,
#    "bands":[{"action": "<DROP or DSCP_REMARK>",
#              "flag": "<KBPS or PKTPS or BURST or STATS"
#              "burst_size": <int>,
#              "rate": <int>,
#              "prec_level": <int>},...]}
#
# delete a meter entry
# DELETE /qos/meter/{switch-id}
#
#  request body format:
#   {"<field>":"<value>"}
#
#     <field>  : <value>
#    "meter_id" : "<int>"
#


SWITCHID_PATTERN = dpid_lib.DPID_PATTERN + r'|all'
VLANID_PATTERN = r'[0-9]{1,4}|all'

QOS_TABLE_ID = 0

REST_ALL = 'all'
REST_SWITCHID = 'switch_id'
REST_COMMAND_RESULT = 'command_result'
REST_PRIORITY = 'priority'
REST_VLANID = 'vlan_id'
REST_PORT_NAME = 'port_name'
REST_QUEUE_TYPE = 'type'
REST_QUEUE_MAX_RATE = 'max_rate'
REST_QUEUE_MIN_RATE = 'min_rate'
REST_QUEUES = 'queues'
REST_QOS = 'qos'
REST_QOS_ID = 'qos_id'
REST_COOKIE = 'cookie'

REST_MATCH = 'match'
REST_IN_PORT = 'in_port'
REST_SRC_MAC = 'dl_src'
REST_DST_MAC = 'dl_dst'
REST_DL_TYPE = 'dl_type'
REST_DL_TYPE_ARP = 'ARP'
REST_DL_TYPE_IPV4 = 'IPv4'
REST_DL_TYPE_IPV6 = 'IPv6'
REST_DL_VLAN = 'dl_vlan'
REST_SRC_IP = 'nw_src'
REST_DST_IP = 'nw_dst'
REST_SRC_IPV6 = 'ipv6_src'
```

```
REST_DST_IPV6 = 'ipv6_dst'
REST_NW_PROTO = 'nw_proto'
REST_NW_PROTO_TCP = 'TCP'
REST_NW_PROTO_UDP = 'UDP'
REST_NW_PROTO_ICMP = 'ICMP'
REST_NW_PROTO_ICMPV6 = 'ICMPv6'
REST_TP_SRC = 'tp_src'
REST_TP_DST = 'tp_dst'
REST_DSCP = 'ip_dscp'

REST_ACTION = 'actions'
REST_ACTION_QUEUE = 'queue'
REST_ACTION_MARK = 'mark'
REST_ACTION_METER = 'meter'

REST_METER_ID = 'meter_id'
REST_METER_BURST_SIZE = 'burst_size'
REST_METER_RATE = 'rate'
REST_METER_PREC_LEVEL = 'prec_level'
REST_METER_BANDS = 'bands'
REST_METER_ACTION_DROP = 'drop'
REST_METER_ACTION_REMARK = 'remark'

DEFAULT_FLOW_PRIORITY = 0
QOS_PRIORITY_MAX = ofproto_v1_3_parser.UINT16_MAX - 1
QOS_PRIORITY_MIN = 1

VLANID_NONE = 0
VLANID_MIN = 2
VLANID_MAX = 4094
COOKIE_SHIFT_VLANID = 32

BASE_URL = '/qos'
REQUIREMENTS = {'switchid': SWITCHID_PATTERN,
                'vlanid': VLANID_PATTERN}

LOG = logging.getLogger(__name__)


class RestQoSAPI(app_manager.RyuApp):

    OFP_VERSIONS = [ofproto_v1_0.OFP_VERSION,
                    ofproto_v1_2.OFP_VERSION,
                    ofproto_v1_3.OFP_VERSION]

    _CONTEXTS = {
        'dpset': dpset.DPSet,
        'conf_switch': conf_switch.ConfSwitchSet,
        'wsgi': WSGIApplication}

    def __init__(self, *args, **kwargs):
        super(RestQoSAPI, self).__init__(*args, **kwargs)

        # logger configure
```

```
        QoSController.set_logger(self.logger)
        self.cs = kwargs['conf_switch']
        self.dpset = kwargs['dpset']
        wsgi = kwargs['wsgi']
        self.waiters = {}
        self.data = {}
        self.data['dpset'] = self.dpset
        self.data['waiters'] = self.waiters
        wsgi.registory['QoSController'] = self.data
        wsgi.register(QoSController, self.data)

    def stats_reply_handler(self, ev):
        msg = ev.msg
        dp = msg.datapath

        if dp.id not in self.waiters:
            return
        if msg.xid not in self.waiters[dp.id]:
            return
        lock, msgs = self.waiters[dp.id][msg.xid]
        msgs.append(msg)

        flags = 0
        if dp.ofproto.OFP_VERSION == ofproto_v1_0.OFP_VERSION or \
                dp.ofproto.OFP_VERSION == ofproto_v1_2.OFP_VERSION:
            flags = dp.ofproto.OFPSF_REPLY_MORE
        elif dp.ofproto.OFP_VERSION == ofproto_v1_3.OFP_VERSION:
            flags = dp.ofproto.OFPMPF_REPLY_MORE

        if msg.flags & flags:
            return
        del self.waiters[dp.id][msg.xid]
        lock.set()

    @set_ev_cls(conf_switch.EventConfSwitchSet)
    def conf_switch_set_handler(self, ev):
        if ev.key == cs_key.OVSDB_ADDR:
            QoSController.set_ovsdb_addr(ev.dpid, ev.value)
        else:
            QoSController._LOGGER.debug("unknown event: %s", ev)

    @set_ev_cls(conf_switch.EventConfSwitchDel)
    def conf_switch_del_handler(self, ev):
        if ev.key == cs_key.OVSDB_ADDR:
            QoSController.delete_ovsdb_addr(ev.dpid)
        else:
            QoSController._LOGGER.debug("unknown event: %s", ev)

    @set_ev_cls(dpset.EventDP, dpset.DPSET_EV_DISPATCHER)
    def handler_datapath(self, ev):
        if ev.enter:
            QoSController.regist_ofs(ev.dp, self.CONF)
        else:
            QoSController.unregist_ofs(ev.dp)
```

```

    # for OpenFlow version1.0
    @set_ev_cls(ofp_event.EventOFPFlowStatsReply, MAIN_DISPATCHER)
    def stats_reply_handler_v1_0(self, ev):
        self.stats_reply_handler(ev)

    # for OpenFlow version1.2 or later
    @set_ev_cls(ofp_event.EventOFPStatsReply, MAIN_DISPATCHER)
    def stats_reply_handler_v1_2(self, ev):
        self.stats_reply_handler(ev)

    # for OpenFlow version1.2 or later
    @set_ev_cls(ofp_event.EventOFPQueueStatsReply, MAIN_DISPATCHER)
    def queue_stats_reply_handler_v1_2(self, ev):
        self.stats_reply_handler(ev)

    # for OpenFlow version1.2 or later
    @set_ev_cls(ofp_event.EventOFPMeterStatsReply, MAIN_DISPATCHER)
    def meter_stats_reply_handler_v1_2(self, ev):
        self.stats_reply_handler(ev)


class QoSOfsList(dict):

    def __init__(self):
        super(QoSOfsList, self).__init__()

    def get_ofs(self, dp_id):
        if len(self) == 0:
            raise ValueError('qos sw is not connected.')

        dps = {}
        if dp_id == REST_ALL:
            dps = self
        else:
            try:
                dpid = dpid_lib.str_to_dpid(dp_id)
            except:
                raise ValueError('Invalid switchID.')

            if dpid in self:
                dps = {dpid: self[dpid]}
            else:
                msg = 'qos sw is not connected. : switchID=%s' % dp_id
                raise ValueError(msg)

        return dps


class QoSController(ControllerBase):

    _OFS_LIST = QoSOfsList()
    _LOGGER = None

```

```
    def __init__(self, req, link, data, **config):
        super(QoSController, self).__init__(req, link, data, **config)
        self.dpset = data['dpset']
        self.waiters = data['waiters']

    @classmethod
    def set_logger(cls, logger):
        cls._LOGGER = logger
        cls._LOGGER.propagate = False
        hdlr = logging.StreamHandler()
        fmt_str = '[QoS][%(levelname)s] %(message)s'
        hdlr.setFormatter(logging.Formatter(fmt_str))
        cls._LOGGER.addHandler(hdlr)

    @staticmethod
    def regist_ofs(dp, CONF):
        if dp.id in QoSController._OFS_LIST:
            return

        dpid_str = dpid_lib.dpid_to_str(dp.id)
        try:
            f_ofs = QoS(dp, CONF)
            f_ofs.set_default_flow()
        except OFPUnknownVersion as message:
            QoSController._LOGGER.info('dpid=%s: %s',
                                       dpid_str, message)
            return

        QoSController._OFS_LIST.setdefault(dp.id, f_ofs)
        QoSController._LOGGER.info('dpid=%s: Join qos switch.',
                                   dpid_str)

    @staticmethod
    def unregist_ofs(dp):
        if dp.id in QoSController._OFS_LIST:
            del QoSController._OFS_LIST[dp.id]
            QoSController._LOGGER.info('dpid=%s: Leave qos switch.',
                                       dpid_lib.dpid_to_str(dp.id))

    @staticmethod
    def set_ovsdb_addr(dpid, value):
        ofs = QoSController._OFS_LIST.get(dpid, None)
        if ofs is not None:
            ofs.set_ovsdb_addr(dpid, value)

    @staticmethod
    def delete_ovsdb_addr(dpid):
        ofs = QoSController._OFS_LIST.get(dpid, None)
        ofs.set_ovsdb_addr(dpid, None)

    @route('qos_switch', BASE_URL + '/queue/{switchid}',
           methods=['GET'], requirements=REQUIREMENTS)
    def get_queue(self, req, switchid, **_kwargs):
        return self._access_switch(req, switchid, VLANID_NONE,
```

```
                                        'get_queue', None)

    @route('qos_switch', BASE_URL + '/queue/{switchid}',
           methods=['POST'], requirements=REQUIREMENTS)
    def set_queue(self, req, switchid, **_kwargs):
        return self._access_switch(req, switchid, VLANID_NONE,
                                        'set_queue', None)

    @route('qos_switch', BASE_URL + '/queue/{switchid}',
           methods=['DELETE'], requirements=REQUIREMENTS)
    def delete_queue(self, req, switchid, **_kwargs):
        return self._access_switch(req, switchid, VLANID_NONE,
                                        'delete_queue', None)

    @route('qos_switch', BASE_URL + '/queue/status/{switchid}',
           methods=['GET'], requirements=REQUIREMENTS)
    def get_status(self, req, switchid, **_kwargs):
        return self._access_switch(req, switchid, VLANID_NONE,
                                        'get_status', self.waiters)

    @route('qos_switch', BASE_URL + '/rules/{switchid}',
           methods=['GET'], requirements=REQUIREMENTS)
    def get_qos(self, req, switchid, **_kwargs):
        return self._access_switch(req, switchid, VLANID_NONE,
                                        'get_qos', self.waiters)

    @route('qos_switch', BASE_URL + '/rules/{switchid}/{vlanid}',
           methods=['GET'], requirements=REQUIREMENTS)
    def get_vlan_qos(self, req, switchid, vlanid, **_kwargs):
        return self._access_switch(req, switchid, vlanid,
                                        'get_qos', self.waiters)

    @route('qos_switch', BASE_URL + '/rules/{switchid}',
           methods=['POST'], requirements=REQUIREMENTS)
    def set_qos(self, req, switchid, **_kwargs):
        return self._access_switch(req, switchid, VLANID_NONE,
                                        'set_qos', self.waiters)

    @route('qos_switch', BASE_URL + '/rules/{switchid}/{vlanid}',
           methods=['POST'], requirements=REQUIREMENTS)
    def set_vlan_qos(self, req, switchid, vlanid, **_kwargs):
        return self._access_switch(req, switchid, vlanid,
                                        'set_qos', self.waiters)

    @route('qos_switch', BASE_URL + '/rules/{switchid}',
           methods=['DELETE'], requirements=REQUIREMENTS)
    def delete_qos(self, req, switchid, **_kwargs):
        return self._access_switch(req, switchid, VLANID_NONE,
                                        'delete_qos', self.waiters)

    @route('qos_switch', BASE_URL + '/rules/{switchid}/{vlanid}',
           methods=['DELETE'], requirements=REQUIREMENTS)
    def delete_vlan_qos(self, req, switchid, vlanid, **_kwargs):
        return self._access_switch(req, switchid, vlanid,
```

```
                                   'delete_qos', self.waiters)

    @route('qos_switch', BASE_URL + '/meter/{switchid}',
           methods=['GET'], requirements=REQUIREMENTS)
    def get_meter(self, req, switchid, **_kwargs):
        return self._access_switch(req, switchid, VLANID_NONE,
                                   'get_meter', self.waiters)

    @route('qos_switch', BASE_URL + '/meter/{switchid}',
           methods=['POST'], requirements=REQUIREMENTS)
    def set_meter(self, req, switchid, **_kwargs):
        return self._access_switch(req, switchid, VLANID_NONE,
                                   'set_meter', self.waiters)

    @route('qos_switch', BASE_URL + '/meter/{switchid}',
           methods=['DELETE'], requirements=REQUIREMENTS)
    def delete_meter(self, req, switchid, **_kwargs):
        return self._access_switch(req, switchid, VLANID_NONE,
                                   'delete_meter', self.waiters)

    def _access_switch(self, req, switchid, vlan_id, func, waiters):
        try:
            rest = req.json if req.body else {}
        except ValueError:
            QoSController._LOGGER.debug('invalid syntax %s', req.body)
            return Response(status=400)

        try:
            dps = self._OFS_LIST.get_ofs(switchid)
            vid = QoSController._conv_toint_vlanid(vlan_id)
        except ValueError as message:
            return Response(status=400, body=str(message))

        msgs = []
        for f_ofs in dps.values():
            function = getattr(f_ofs, func)
            try:
                if waiters is not None:
                    msg = function(rest, vid, waiters)
                else:
                    msg = function(rest, vid)
            except ValueError as message:
                return Response(status=400, body=str(message))
            msgs.append(msg)

        body = json.dumps(msgs)
        return Response(content_type='application/json', body=body)

    @staticmethod
    def _conv_toint_vlanid(vlan_id):
        if vlan_id != REST_ALL:
            vlan_id = int(vlan_id)
            if (vlan_id != VLANID_NONE and
                    (vlan_id < VLANID_MIN or VLANID_MAX < vlan_id)):
```

```python
                msg = 'Invalid {vlan_id} value. Set [%d-%d]' % (
VLANID_MIN,

VLANID_MAX)
                raise ValueError(msg)
        return vlan_id


class QoS(object):

    _OFCTL = {ofproto_v1_0.OFP_VERSION: ofctl_v1_0,
              ofproto_v1_2.OFP_VERSION: ofctl_v1_2,
              ofproto_v1_3.OFP_VERSION: ofctl_v1_3}

    def __init__(self, dp, CONF):
        super(QoS, self).__init__()
        self.vlan_list = {}
        self.vlan_list[VLANID_NONE] = 0  # for VLAN=None
        self.dp = dp
        self.version = dp.ofproto.OFP_VERSION
        self.queue_list = {}
        self.CONF = CONF
        self.ovsdb_addr = None
        self.ovs_bridge = None

        if self.version not in self._OFCTL:
            raise OFPUnknownVersion(version=self.version)

        self.ofctl = self._OFCTL[self.version]

    def set_default_flow(self):
        if self.version == ofproto_v1_0.OFP_VERSION:
            return

        cookie = 0
        priority = DEFAULT_FLOW_PRIORITY
        actions = [{'type': 'GOTO_TABLE',
                    'table_id': QOS_TABLE_ID + 1}]
        flow = self._to_of_flow(cookie=cookie,
                                priority=priority,
                                match={},
                                actions=actions)

        cmd = self.dp.ofproto.OFPFC_ADD
        self.ofctl.mod_flow_entry(self.dp, flow, cmd)

    def set_ovsdb_addr(self, dpid, ovsdb_addr):
        # easy check if the address format valid
        _proto, _host, _port = ovsdb_addr.split(':')

        old_address = self.ovsdb_addr
        if old_address == ovsdb_addr:
            return
        if ovsdb_addr is None:
```

```
            if self.ovs_bridge:
                self.ovs_bridge.del_controller()
                self.ovs_bridge = None
            return
        self.ovsdb_addr = ovsdb_addr
        if self.ovs_bridge is None:
            ovs_bridge = bridge.OVSBridge(self.CONF, dpid, ovsdb_addr)
            self.ovs_bridge = ovs_bridge
            try:
                ovs_bridge.init()
            except:
                raise ValueError('ovsdb addr is not available.')

    def _update_vlan_list(self, vlan_list):
        for vlan_id in self.vlan_list.keys():
            if vlan_id is not VLANID_NONE and vlan_id not in vlan_list:
                del self.vlan_list[vlan_id]

    def _get_cookie(self, vlan_id):
        if vlan_id == REST_ALL:
            vlan_ids = self.vlan_list.keys()
        else:
            vlan_ids = [vlan_id]

        cookie_list = []
        for vlan_id in vlan_ids:
            self.vlan_list.setdefault(vlan_id, 0)
            self.vlan_list[vlan_id] += 1
            self.vlan_list[vlan_id] &= ofproto_v1_3_parser.UINT32_MAX
            cookie = (vlan_id << COOKIE_SHIFT_VLANID) + \
                self.vlan_list[vlan_id]
            cookie_list.append([cookie, vlan_id])

        return cookie_list

    @staticmethod
    def _cookie_to_qosid(cookie):
        return cookie & ofproto_v1_3_parser.UINT32_MAX

    # REST command template
    def rest_command(func):
        def _rest_command(*args, **kwargs):
            key, value = func(*args, **kwargs)
            switch_id = dpid_lib.dpid_to_str(args[0].dp.id)
            return {REST_SWITCHID: switch_id,
                    key: value}
        return _rest_command

    @rest_command
    def get_status(self, req, vlan_id, waiters):
        if self.version == ofproto_v1_0.OFP_VERSION:
            raise ValueError('get_status operation is not supported')

        msgs = self.ofctl.get_queue_stats(self.dp, waiters)
```

```
        return REST_COMMAND_RESULT, msgs

    @rest_command
    def get_queue(self, rest, vlan_id):
        if len(self.queue_list):
            msg = {'result': 'success',
                   'details': self.queue_list}
        else:
            msg = {'result': 'failure',
                   'details': 'Queue is not exists.'}

        return REST_COMMAND_RESULT, msg

    @rest_command
    def set_queue(self, rest, vlan_id):
        if self.ovs_bridge is None:
            msg = {'result': 'failure',
                   'details': 'ovs_bridge is not exists'}
            return REST_COMMAND_RESULT, msg

        self.queue_list.clear()
        queue_type = rest.get(REST_QUEUE_TYPE, 'linux-htb')
        parent_max_rate = rest.get(REST_QUEUE_MAX_RATE, None)
        queues = rest.get(REST_QUEUES, [])
        queue_id = 0
        queue_config = []
        for queue in queues:
            max_rate = queue.get(REST_QUEUE_MAX_RATE, None)
            min_rate = queue.get(REST_QUEUE_MIN_RATE, None)
            if max_rate is None and min_rate is None:
                raise ValueError('Required to specify max_rate or min_rate')
            config = {}
            if max_rate is not None:
                config['max-rate'] = max_rate
            if min_rate is not None:
                config['min-rate'] = min_rate
            if len(config):
                queue_config.append(config)
            self.queue_list[queue_id] = {'config': config}
            queue_id += 1

        port_name = rest.get(REST_PORT_NAME, None)
        vif_ports = self.ovs_bridge.get_port_name_list()

        if port_name is not None:
            if port_name not in vif_ports:
                raise ValueError('%s port is not exists' % port_name)
            vif_ports = [port_name]

        for port_name in vif_ports:
            try:
                self.ovs_bridge.set_qos(port_name, type=queue_type,
                                        max_rate=parent_max_rate,
```

```
                                        queues=queue_config)
            except Exception as msg:
                raise ValueError(msg)

        msg = {'result': 'success',
               'details': self.queue_list}

        return REST_COMMAND_RESULT, msg

    def _delete_queue(self):
        if self.ovs_bridge is None:
            return False

        vif_ports = self.ovs_bridge.get_external_ports()
        for port in vif_ports:
            self.ovs_bridge.del_qos(port.port_name)
        return True

    @rest_command
    def delete_queue(self, rest, vlan_id):
        self.queue_list.clear()
        if self._delete_queue():
            msg = 'success'
        else:
            msg = 'failure'

        return REST_COMMAND_RESULT, msg

    @rest_command
    def set_qos(self, rest, vlan_id, waiters):
        msgs = []
        cookie_list = self._get_cookie(vlan_id)
        for cookie, vid in cookie_list:
            msg = self._set_qos(cookie, rest, waiters, vid)
            msgs.append(msg)
        return REST_COMMAND_RESULT, msgs

    def _set_qos(self, cookie, rest, waiters, vlan_id):
        match_value = rest[REST_MATCH]
        if vlan_id:
            match_value[REST_DL_VLAN] = vlan_id

        priority = int(rest.get(REST_PRIORITY, QOS_PRIORITY_MIN))
        if (QOS_PRIORITY_MAX < priority):
            raise ValueError('Invalid priority value. Set [%d-%d]'
                             % (QOS_PRIORITY_MIN, QOS_PRIORITY_MAX))

        match = Match.to_openflow(match_value)

        actions = []
        action = rest.get(REST_ACTION, None)
        if action is not None:
            if REST_ACTION_MARK in action:
                actions.append({'type': 'SET_FIELD',
```

```
                                'field': REST_DSCP,
                                'value': int(action[REST_ACTION_MARK])})
            if REST_ACTION_METER in action:
                actions.append({'type': 'METER',
                                'meter_id': action[REST_ACTION_METER]})
            if REST_ACTION_QUEUE in action:
                actions.append({'type': 'SET_QUEUE',
                                'queue_id': action[REST_ACTION_QUEUE]})
        else:
            actions.append({'type': 'SET_QUEUE',
                            'queue_id': 0})

        actions.append({'type': 'GOTO_TABLE',
                        'table_id': QOS_TABLE_ID + 1})
        flow = self._to_of_flow(cookie=cookie, priority=priority,
                                match=match, actions=actions)

        cmd = self.dp.ofproto.OFPFC_ADD
        try:
            self.ofctl.mod_flow_entry(self.dp, flow, cmd)
        except:
            raise ValueError('Invalid rule parameter.')

        qos_id = QoS._cookie_to_qosid(cookie)
        msg = {'result': 'success',
               'details': 'QoS added. : qos_id=%d' % qos_id}

        if vlan_id != VLANID_NONE:
            msg.setdefault(REST_VLANID, vlan_id)
        return msg

    @rest_command
    def get_qos(self, rest, vlan_id, waiters):
        rules = {}
        msgs = self.ofctl.get_flow_stats(self.dp, waiters)
        if str(self.dp.id) in msgs:
            flow_stats = msgs[str(self.dp.id)]
            for flow_stat in flow_stats:
                if flow_stat['table_id'] != QOS_TABLE_ID:
                    continue
                priority = flow_stat[REST_PRIORITY]
                if priority != DEFAULT_FLOW_PRIORITY:
                    vid = flow_stat[REST_MATCH].get(REST_DL_VLAN,
VLANID_NONE)
                    if vlan_id == REST_ALL or vlan_id == vid:
                        rule = self._to_rest_rule(flow_stat)
                        rules.setdefault(vid, [])
                        rules[vid].append(rule)

        get_data = []
        for vid, rule in rules.items():
            if vid == VLANID_NONE:
                vid_data = {REST_QOS: rule}
            else:
```

```
                vid_data = {REST_VLANID: vid, REST_QOS: rule}
            get_data.append(vid_data)

        return REST_COMMAND_RESULT, get_data

    @rest_command
    def delete_qos(self, rest, vlan_id, waiters):
        try:
            if rest[REST_QOS_ID] == REST_ALL:
                qos_id = REST_ALL
            else:
                qos_id = int(rest[REST_QOS_ID])
        except:
            raise ValueError('Invalid qos id.')

        vlan_list = []
        delete_list = []

        msgs = self.ofctl.get_flow_stats(self.dp, waiters)
        if str(self.dp.id) in msgs:
            flow_stats = msgs[str(self.dp.id)]
            for flow_stat in flow_stats:
                cookie = flow_stat[REST_COOKIE]
                ruleid = QoS._cookie_to_qosid(cookie)
                priority = flow_stat[REST_PRIORITY]
                dl_vlan = flow_stat[REST_MATCH].get(REST_DL_VLAN, VLANID_NONE)

                if priority != DEFAULT_FLOW_PRIORITY:
                    if ((qos_id == REST_ALL or qos_id == ruleid) and
                            (vlan_id == dl_vlan or vlan_id == REST_ALL)):
                        match = Match.to_mod_openflow(flow_stat[REST_MATCH])
                        delete_list.append([cookie, priority, match])
                    else:
                        if dl_vlan not in vlan_list:
                            vlan_list.append(dl_vlan)

        self._update_vlan_list(vlan_list)

        if len(delete_list) == 0:
            msg_details = 'QoS rule is not exist.'
            if qos_id != REST_ALL:
                msg_details += ' : QoS ID=%d' % qos_id
            msg = {'result': 'failure',
                   'details': msg_details}
        else:
            cmd = self.dp.ofproto.OFPFC_DELETE_STRICT
            actions = []
            delete_ids = {}
            for cookie, priority, match in delete_list:
                flow = self._to_of_flow(cookie=cookie, priority=priority,
```

```
                                        match=match, actions=actions)
                self.ofctl.mod_flow_entry(self.dp, flow, cmd)

                vid = match.get(REST_DL_VLAN, VLANID_NONE)
                rule_id = QoS._cookie_to_qosid(cookie)
                delete_ids.setdefault(vid, '')
                delete_ids[vid] += (('%d' if delete_ids[vid] == ''
                                     else ',%d') % rule_id)

            msg = []
            for vid, rule_ids in delete_ids.items():
                del_msg = {'result': 'success',
                           'details': ' deleted. : QoS ID=%s' % rule_ids}
                if vid != VLANID_NONE:
                    del_msg.setdefault(REST_VLANID, vid)
                msg.append(del_msg)

        return REST_COMMAND_RESULT, msg

    @rest_command
    def set_meter(self, rest, vlan_id, waiters):
        if self.version == ofproto_v1_0.OFP_VERSION:
            raise ValueError('set_meter operation is not supported')

        msgs = []
        msg = self._set_meter(rest, waiters)
        msgs.append(msg)
        return REST_COMMAND_RESULT, msgs

    def _set_meter(self, rest, waiters):
        cmd = self.dp.ofproto.OFPMC_ADD
        try:
            self.ofctl.mod_meter_entry(self.dp, rest, cmd)
        except:
            raise ValueError('Invalid meter parameter.')

        msg = {'result': 'success',
               'details': 'Meter added. : Meter ID=%s' %
               rest[REST_METER_ID]}
        return msg

    @rest_command
    def get_meter(self, rest, vlan_id, waiters):
        if (self.version == ofproto_v1_0.OFP_VERSION or
                self.version == ofproto_v1_2.OFP_VERSION):
            raise ValueError('get_meter operation is not supported')

        msgs = self.ofctl.get_meter_stats(self.dp, waiters)
        return REST_COMMAND_RESULT, msgs

    @rest_command
    def delete_meter(self, rest, vlan_id, waiters):
        if (self.version == ofproto_v1_0.OFP_VERSION or
```

```
                self.version == ofproto_v1_2.OFP_VERSION):
            raise ValueError('delete_meter operation is not supported')

        cmd = self.dp.ofproto.OFPMC_DELETE
        try:
            self.ofctl.mod_meter_entry(self.dp, rest, cmd)
        except:
            raise ValueError('Invalid meter parameter.')

        msg = {'result': 'success',
               'details': 'Meter deleted. : Meter ID=%s' %
               rest[REST_METER_ID]}
        return REST_COMMAND_RESULT, msg

    def _to_of_flow(self, cookie, priority, match, actions):
        flow = {'cookie': cookie,
                'priority': priority,
                'flags': 0,
                'idle_timeout': 0,
                'hard_timeout': 0,
                'match': match,
                'actions': actions}
        return flow

    def _to_rest_rule(self, flow):
        ruleid = QoS._cookie_to_qosid(flow[REST_COOKIE])
        rule = {REST_QOS_ID: ruleid}
        rule.update({REST_PRIORITY: flow[REST_PRIORITY]})
        rule.update(Match.to_rest(flow))
        rule.update(Action.to_rest(flow))
        return rule


class Match(object):

    _CONVERT = {REST_DL_TYPE:
                {REST_DL_TYPE_ARP: ether.ETH_TYPE_ARP,
                 REST_DL_TYPE_IPV4: ether.ETH_TYPE_IP,
                 REST_DL_TYPE_IPV6: ether.ETH_TYPE_IPV6},
                REST_NW_PROTO:
                {REST_NW_PROTO_TCP: inet.IPPROTO_TCP,
                 REST_NW_PROTO_UDP: inet.IPPROTO_UDP,
                 REST_NW_PROTO_ICMP: inet.IPPROTO_ICMP,
                 REST_NW_PROTO_ICMPV6: inet.IPPROTO_ICMPV6}}

    @staticmethod
    def to_openflow(rest):

        def __inv_combi(msg):
            raise ValueError('Invalid combination: [%s]' % msg)

        def __inv_2and1(*args):
            __inv_combi('%s=%s and %s' % (args[0], args[1], args[2]))

```

```
        def __inv_2and2(*args):
            __inv_combi('%s=%s and %s=%s' % (
                args[0], args[1], args[2], args[3]))

        def __inv_1and1(*args):
            __inv_combi('%s and %s' % (args[0], args[1]))

        def __inv_1and2(*args):
            __inv_combi('%s and %s=%s' % (args[0], args[1], args[2]))

        match = {}

        # error check
        dl_type = rest.get(REST_DL_TYPE)
        nw_proto = rest.get(REST_NW_PROTO)
        if dl_type is not None:
            if dl_type == REST_DL_TYPE_ARP:
                if REST_SRC_IPV6 in rest:
                    __inv_2and1(
                        REST_DL_TYPE, REST_DL_TYPE_ARP, REST_SRC_IPV6)
                if REST_DST_IPV6 in rest:
                    __inv_2and1(
                        REST_DL_TYPE, REST_DL_TYPE_ARP, REST_DST_IPV6)
                if REST_DSCP in rest:
                    __inv_2and1(
                        REST_DL_TYPE, REST_DL_TYPE_ARP, REST_DSCP)
                if nw_proto:
                    __inv_2and1(
                        REST_DL_TYPE, REST_DL_TYPE_ARP, REST_NW_PROTO)
            elif dl_type == REST_DL_TYPE_IPV4:
                if REST_SRC_IPV6 in rest:
                    __inv_2and1(
                        REST_DL_TYPE, REST_DL_TYPE_IPV4, REST_SRC_IPV6)
                if REST_DST_IPV6 in rest:
                    __inv_2and1(
                        REST_DL_TYPE, REST_DL_TYPE_IPV4, REST_DST_IPV6)
                if nw_proto == REST_NW_PROTO_ICMPV6:
                    __inv_2and2(
                        REST_DL_TYPE, REST_DL_TYPE_IPV4,
                        REST_NW_PROTO, REST_NW_PROTO_ICMPV6)
            elif dl_type == REST_DL_TYPE_IPV6:
                if REST_SRC_IP in rest:
                    __inv_2and1(
                        REST_DL_TYPE, REST_DL_TYPE_IPV6, REST_SRC_IP)
                if REST_DST_IP in rest:
                    __inv_2and1(
                        REST_DL_TYPE, REST_DL_TYPE_IPV6, REST_DST_IP)
                if nw_proto == REST_NW_PROTO_ICMP:
                    __inv_2and2(
                        REST_DL_TYPE, REST_DL_TYPE_IPV6,
                        REST_NW_PROTO, REST_NW_PROTO_ICMP)
            else:
                raise ValueError('Unknown dl_type : %s' % dl_type)
        else:
```

```
            if REST_SRC_IP in rest:
                if REST_SRC_IPV6 in rest:
                    __inv_1and1(REST_SRC_IP, REST_SRC_IPV6)
                if REST_DST_IPV6 in rest:
                    __inv_1and1(REST_SRC_IP, REST_DST_IPV6)
                if nw_proto == REST_NW_PROTO_ICMPV6:
                    __inv_1and2(
                        REST_SRC_IP, REST_NW_PROTO, REST_NW_PROTO_ICMPV6)
                rest[REST_DL_TYPE] = REST_DL_TYPE_IPV4
            elif REST_DST_IP in rest:
                if REST_SRC_IPV6 in rest:
                    __inv_1and1(REST_DST_IP, REST_SRC_IPV6)
                if REST_DST_IPV6 in rest:
                    __inv_1and1(REST_DST_IP, REST_DST_IPV6)
                if nw_proto == REST_NW_PROTO_ICMPV6:
                    __inv_1and2(
                        REST_DST_IP, REST_NW_PROTO, REST_NW_PROTO_ICMPV6)
                rest[REST_DL_TYPE] = REST_DL_TYPE_IPV4
            elif REST_SRC_IPV6 in rest:
                if nw_proto == REST_NW_PROTO_ICMP:
                    __inv_1and2(
                        REST_SRC_IPV6, REST_NW_PROTO, REST_NW_PROTO_ICMP)
                rest[REST_DL_TYPE] = REST_DL_TYPE_IPV6
            elif REST_DST_IPV6 in rest:
                if nw_proto == REST_NW_PROTO_ICMP:
                    __inv_1and2(
                        REST_DST_IPV6, REST_NW_PROTO, REST_NW_PROTO_ICMP)
                rest[REST_DL_TYPE] = REST_DL_TYPE_IPV6
            elif REST_DSCP in rest:
                # Apply dl_type ipv4, if doesn't specify dl_type
                rest[REST_DL_TYPE] = REST_DL_TYPE_IPV4
            else:
                if nw_proto == REST_NW_PROTO_ICMP:
                    rest[REST_DL_TYPE] = REST_DL_TYPE_IPV4
                elif nw_proto == REST_NW_PROTO_ICMPV6:
                    rest[REST_DL_TYPE] = REST_DL_TYPE_IPV6
                elif nw_proto == REST_NW_PROTO_TCP or \
                        nw_proto == REST_NW_PROTO_UDP:
                    raise ValueError('no dl_type was specified')
                else:
                    raise ValueError('Unknown nw_proto: %s' % nw_proto)

        for key, value in rest.items():
            if key in Match._CONVERT:
                if value in Match._CONVERT[key]:
                    match.setdefault(key, Match._CONVERT[key][value])
                else:
                    raise ValueError('Invalid rule parameter. : key=%s' % key)
            else:
```

```
            match.setdefault(key, value)

    return match

@staticmethod
def to_rest(openflow):
    of_match = openflow[REST_MATCH]

    mac_dontcare = mac.haddr_to_str(mac.DONTCARE)
    ip_dontcare = '0.0.0.0'
    ipv6_dontcare = '::'

    match = {}
    for key, value in of_match.items():
        if key == REST_SRC_MAC or key == REST_DST_MAC:
            if value == mac_dontcare:
                continue
        elif key == REST_SRC_IP or key == REST_DST_IP:
            if value == ip_dontcare:
                continue
        elif key == REST_SRC_IPV6 or key == REST_DST_IPV6:
            if value == ipv6_dontcare:
                continue
        elif value == 0:
            continue

        if key in Match._CONVERT:
            conv = Match._CONVERT[key]
            conv = dict((value, key) for key, value in conv.items())
            match.setdefault(key, conv[value])
        else:
            match.setdefault(key, value)

    return match

@staticmethod
def to_mod_openflow(of_match):
    mac_dontcare = mac.haddr_to_str(mac.DONTCARE)
    ip_dontcare = '0.0.0.0'
    ipv6_dontcare = '::'

    match = {}
    for key, value in of_match.items():
        if key == REST_SRC_MAC or key == REST_DST_MAC:
            if value == mac_dontcare:
                continue
        elif key == REST_SRC_IP or key == REST_DST_IP:
            if value == ip_dontcare:
                continue
        elif key == REST_SRC_IPV6 or key == REST_DST_IPV6:
            if value == ipv6_dontcare:
                continue
        elif value == 0:
            continue
```

```
            match.setdefault(key, value)

        return match


class Action(object):

    @staticmethod
    def to_rest(flow):
        if REST_ACTION in flow:
            actions = []
            for act in flow[REST_ACTION]:
                field_value = re.search('SET_FIELD: \{ip_dscp:(\d+)', act)
                if field_value:
                    actions.append({REST_ACTION_MARK: field_value.group(1)})
                meter_value = re.search('METER:(\d+)', act)
                if meter_value:
                    actions.append({REST_ACTION_METER: meter_value.group(1)})
                queue_value = re.search('SET_QUEUE:(\d+)', act)
                if queue_value:
                    actions.append({REST_ACTION_QUEUE: queue_value.group(1)})
            action = {REST_ACTION: actions}
        else:
            action = {REST_ACTION: 'Unknown action type.'}

        return action
```