Journal of Network and Computer Applications 192 (2021) 103181

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

Deep Q-Network and Traffic Prediction based Routing Optimization in Software Defined Networks

EL Hocine Bouzidi [a,b,*], Abdelkader Outtagarts [a], Rami Langar [b,c], Raouf Boutaba [d]

[a] Nokia Bell Labs, Villarceaux Center - Route de Villejust, 91620 Nozay, France
[b] LIGM CNRS-UMR 8049, University Gustave Eiffel, 77420 Marne-la-Vallée, France
[c] SITE Department, ETS-Montréal, Montréal (QC), Canada
[d] D.R. Cheriton School of Computer Science, University of Waterloo, Waterloo ON, Canada

ARTICLE INFO

Keywords:
SDN
Prediction
Neural Networks
QoS
ONOS
DQN
LSTM

ABSTRACT

Software Defined Networking (SDN) is gaining momentum not only in research but also in IT industry representing the drivers of 5G networks, due to its capabilities of increasing the flexibility of a network and address a variety of network challenges, by logically centralizing the intelligence in software-based controllers. Thanks to Machine Learning (ML) techniques, the network performances and utilization can be optimized and enhanced. Neural Networks (NN) and Reinforcement Learning (RL), in particular, have demonstrated great success in cooperating with complex problems arising in network operation and management. To this end, we exploit in this paper, an SDN-based rules placement approach that aims to dynamically predict the traffic congestion by using mainly NN and learn optimal paths and reroute traffic to improve network utilization by deploying a Deep Q-Network (DQN) agent. To this end, we first formulate the Quality-of-Service (QoS)-aware routing problem as a Linear Program (LP), whose objective is to minimize the end-to-end (E2E) delay and link utilization. Then, we propose a simple yet efficient heuristic algorithm to solve it. Numerical results through emulation using ONOS controller and Mininet demonstrate that the proposed approach can significantly improve network performances in terms of decreasing the link utilization, the packet loss and the E2E delay.

1. Introduction

With the development of new technologies such as (5G), network traffic is expected to grow at an exponential rate, due to the wide range of applications with stringent and heterogeneous requirements, such as massive machine-type communications, ultra-reliable low-latency, enhanced mobile broadband communications, big data and cloud applications. Hence, to meet Quality-of-Service (QoS) and Service Level Agreement (SLA) requirements, it is essential to develop innovative ways in which traffic flows can be managed in real-time. Software Defined Networking (SDN) (ONF, 2012) is one of the key emerging technologies for the 5G vision, capable of increasing the flexibility of a network and reduce its costs.

SDN aims at decoupling the network intelligence from the network devices, enabling thus a centralization of network intelligence, a flexibility in traffic control and a simplicity in network management and operation. However, as the size of the network and the number of flows increase, the computational complexity of the control plane increases exponentially. Adapting traditional network policies to the continually changing network behavior is a challenging task. Indeed, to effectively avoid congestion and overloading links, both latency and throughput must be monitored continually and proactively, in order to quickly route packets to less used links. One of the first SDN protocol standards is OpenFlow (McKeown et al., 2008) that enables direct interaction with the forwarding plane of network devices. Although OpenFlow provides a mechanism to request throughput statistics, latest specifications of this protocol do not provide mechanisms to measure latency.

Moreover, existing routing algorithms are not suitable for SDN due to their convergence limit and the absence of a future vision on the evolution of network traffic. Several approaches have been proposed to cope with this challenge. Hyun and Hong (2017) proposed the Self-driving network concept, in which the routing decision is automatic and based on the analyzed telemetry collected from the network data plane, where analyzing collected statistics can be done by Artificial intelligence (AI) and Machine Learning (ML) techniques.

* Corresponding author at: Nokia Bell Labs, Villarceaux Center - Route de Villejust, 91620 Nozay, France.
E-mail addresses: el_hocine.bouzidi@nokia.com (E.H. Bouzidi), abdelkader.outtagarts@nokia-bell-labs.com (A. Outtagarts), rami.langar@u-pem.fr, rami.langar@etsmtl.ca (R. Langar), rboutaba@uwaterloo.ca (R. Boutaba).

https://doi.org/10.1016/j.jnca.2021.103181
Received 24 March 2021; Received in revised form 20 June 2021; Accepted 7 August 2021
Available online 13 August 2021

Pham Tran Anh et al. (2019) proposed Knowledge-Defined Networking (KDN), by introducing the Knowledge plane (KP) to the conventional SDN paradigm, that is responsible for learning the behavior of the network by applying ML on the collected data plane statistics.

Incorporating intelligence via ML to the SDN control plane is crucial to guarantee the requested QoS and optimize routing in SDN-based networks (Boutaba et al., 2018). Indeed, Feng and Shu (2005) used the linear prediction method Auto-Regressive Integrated Moving Average (ARIMA) to predict the future evolution of network traffic. However, such ML methods are not suitable to handle 5G networks and beyond due to their limits to efficiently cope with the large volume data. To this end, the deep learning techniques are well placed to intelligently make decisions on scheduling, bandwidth reservation, etc.

Reinforcement Learning (RL) techniques (Boyan and Littman, 1999) as well are gaining momentum in routing optimization. The principal idea behind it is to deploy an agent that periodically makes decision and automatically adjusts its strategy by learning a state-to-action mapping while maximizing a numerical reward. Khodayari and Yazdanpanah (2005) proposed a Q-routing algorithm looking for minimizing the average delivery time. The main drawback of RL techniques is the slowness to reach the best policy when exploring the entire system, making it unsuitable and inapplicable to large-scale networks with almost countless state number. By taking advantage of Deep Learning to speed up the learning process, combining RL techniques to Deep Learning is well placed to overcome limitations of RL, which referred to as Deep Reinforcement Learning (DRL). Consequently, using NNs instead of Q-tables in DRL makes it possible to achieve new network states, save the time processing and storage of Q-tables. Several methods of DRL have been used where the basic one is DQN (Mnih et al., 2013, 2015), that combines a DNN with Q-learning.

To this end, we propose, in this paper, a dynamic and efficient traffic engineering scheme, called Deep Q-Network and Traffic Prediction based Routing Optimization (DTPRO), which extends our previous work in Bouzidi et al. (2019) by adding a DQN agent as well as a traffic prediction module in order to optimize the network flow routing. Specifically, our proposed solution consists of three main phases. Firstly, we dynamically optimize the flow routing in the network by training a DQN agent. Secondly, we predict congestion and adjust the DQN reward function to provide better routing configurations. Finally, we route the network traffic based on a set of link weights given by the trained DQN agent, and at the same time reroute the existing traffic away from the congested paths by resolving a Linear Program (LP). The formulated LP represents the flow rules placement problem, where the objective is to minimize the total network delay, packet loss and link utilization. Note that, during the third phase, we have proposed an heuristic that interacts with the DQN agent and the traffic prediction in order to solve the formulated LP and optimize network performances. Experimental results, using the ONOS controller and Mininet, show that the proposed approach provides a promising enhancement against traditional routing algorithms.

The main contributions of our paper can be summarized as follows:

- First, we train a DQN agent with appropriate states and actions, in order to optimize the flow routing in the network.
- Second, we predict congestion and adjust the DQN reward function to provide better routing configurations.
- Third, we mathematically model the QoS-aware routing problem as a LP, which takes as inputs routing strategy given by the trained DQN agent and the predicted traffic. Our objective is to minimize the E2E delay, E2E link utilization and E2E packet loss. Then, we propose a simple yet efficient heuristic algorithm called DTPRO to solve the LP.
- Fourth, we implement the DTPRO approach using ONOS controller and Mininet.

The remainder of this paper is organized as follows. Section 2 presents related works. In Section 3, we discuss the architecture of the proposed framework and the rules placement algorithm. Section 4 evaluates the proposed method. We finally conclude this paper in Section 5.

2. Related works

In the following, we first survey the literature on traffic monitoring approaches for collecting data plane statistics in SDN. Then, we focus on the ML techniques used for traffic prediction and routing optimization.

2.1. Traffic monitoring

Recently, SDN (ONF, 2012) with OpenFlow protocol (McKeown et al., 2008) implementation is getting a lot of attention. There are many OpenFlow software implementing the SDN architecture, where the most used in the control plane are ONOS (Berde et al., 2014), POX (Kaur et al., 2014) and Ryu (SDN Controller Ryu, 2021), and OpenvSwitch (Pfaff et al., 2009) in the data plane. In traditional networks, many monitoring tools are available, such as, SNMP (Affandi et al., 2015), Cisco NetFlow (Cisco NetFlow, 2004), sFlow (Phaal et al., 2001). However, these monitoring tools are not compatible with the OpenFlow network. Several works have been proposed in OpenFlow monitoring. Chowdhury et al. (2014) proposed PayLess, a network monitoring framework for SDN, which is built on top of an OpenFlow controller's northbound API and provides a high-level RESTful API, and offers an adaptive scheduling algorithm for polling, that achieves the same level of accuracy as continuous switch polling with much less communication overhead. van Adrichem et al. (2014) proposed OpenNetMon, a network monitoring tool based on OpenFlow, that collects statistics such as throughput and packet loss from the edge devices in order to improve the measurement accuracy and reduce the computation overhead. PathMon (Wang et al., 2016) provides a way to collect per-flow statistics such as throughput and packet loss by inserting a separate set of flow entries called monitoring entries into every switch along a path to be monitored. In these works, statistics can be collected proactively by using $FlowStatisticsRequest$ message or reactively by using the $FlowRemoved$ notification message. However, they did not provide mechanisms to measure latency by using only the OpenFlow protocol.

There have been many studies on latency measurement in SDN using OpenFlow protocol (Azizi et al., 2015; Yu et al., 2015). The most relevant work is described in Yu et al. (2015) in which the authors present SLAM, a framework for Software-defined Latency Monitoring between any two network switches. The delay is measured inside the network by capturing directly path information from network devices. Different from these works where the latency is measured for a specific path, we propose in this paper to measure metrics, such as latency, throughput and per-flow size, for all critical links in the network, where the probability of congestion is important. In this way, an important number of paths can be covered without affecting performances.

2.2. Machine learning based QoS-aware routing

Optimizing flow routing in SDN-based networks is crucial to meet the needs for efficient resource allocation. For that reason, a wide range of solutions have been proposed in the literature.

Reza Parsaei et al. (2017) proposed approaches based on genetic and ant-colony algorithms to optimize the flow rule placement. However, these approaches are limited to certain situations and do not cover more complicated routing problems. Several research works have been made to install flow rules across pre-computed optimal paths, by exploiting the SDN controller's global visibility. Han et al. (2014) proposed an approach to reduce power consumption and network

congestion, where they compute the optimal topology that can accommodate the expected traffic demands, then a traffic load balancing by distributing flows across k-shortest paths of optimal topology, by resolving an Integer Linear Program (ILP) Problem.

Holt–Winters (HW), ARMA/ARIMA models (Cortez et al., 2006) are traditional linear prediction methods, that are widely used for network traffic forecasting. Works in Barabas et al. (2011) and Azzouni and Pujolle (2017) both deploy NNs based on SDN for network Traffic Matrix (TM) prediction. Barabas et al. (2011) presented an approach to predict aggregated Ethernet traffic with NNs employing multiresolution learning (MRL). They consider the problem of forecasting the transfer rate, by predicting future values, given a set of transfer rates observed on a specific link. Experimental results show that nonlinear traffic prediction based on NNs outperforms linear forecasting models (e.g. ARIMA, Auto-Regressive Auto-Regressive (ARAR), HW) which cannot meet the accuracy requirements. Azzouni and Pujolle (2017) presented a Long Short Term Memory (LSTM) based Recurrent Neural Network (RNN) framework, which makes use of model parameters to train prediction models for large scale TM prediction, by training and comparing LSTM models at various numbers of parameters and configurations. Simulations show that the proposed LSTM models converge quickly and achieve high prediction accuracy in a few seconds of computation.

Khodayari and Yazdanpanah (2005) used RL techniques in the context of routing optimization, where they present a Q-Routing algorithm discovering efficient routing policies in dynamically changing networks, without having to know in advance the network topology and traffic patterns. As in Q-Routing algorithms, the Q-Value is stored in a packet traversing the network. Even solutions proposed in this work can achieve low latency, high throughput, and adaptive routing, it can be inefficient when it comes to the Q-Table storage space and the time to reach the best policy.

DRL addresses this challenge by taking advantage of Deep Neural Networks (DNNs) to train the learning process of RL algorithms. Most relevant works are described in Yu et al. (2018) and Pham Tran Anh et al. (2019), where the DRL agent interacts with the network through three signals: state (Traffic Matrix), action (link-weight vector), and reward (improving network performances). DQN (Mnih et al., 2013, 2015) is one of the most used DRL methods in network routing optimization. Su et al. (2019) presented an adaptive DQN based energy and latency-aware routing protocol (DQELR) that adopts a DQN algorithm with both off-policy and on-policy methods to make routing decisions adaptively in different network conditions. Liu (2019) presented DRL-R (Deep Reinforcement Learning-based Routing) to cope with the problem of coexistence of Elephant flow/Mice flow/Coflow and multiple resources (bandwidth, cache and computing). The proposed approach is based on DQN and Deep Deterministic Policy Gradient (DDPG). Experimental results demonstrated the effectiveness of this solution in improving network performances. However, these works did not take into account the traffic prediction, since the DRL agent is triggered only once the first packet in a traffic flow is detected by the controller. Moreover, actions depend only on link weights modification not on flow rules installation.

To sum up, the aforementioned works present the following shortcomings: (i) the non-consideration of the mapping history between the network state and the corresponding action when predicting congestion, and (ii) the non-consideration of current and predicted congestion when rewarding actions. Based on these observations, we propose in this paper to train a DQN agent capable of optimizing routing by dynamically choosing the optimal path according to a rewarding function, while taking into account latency, throughput, and packet loss. In addition, we propose to deploy a traffic Prediction module to predict network congestion.

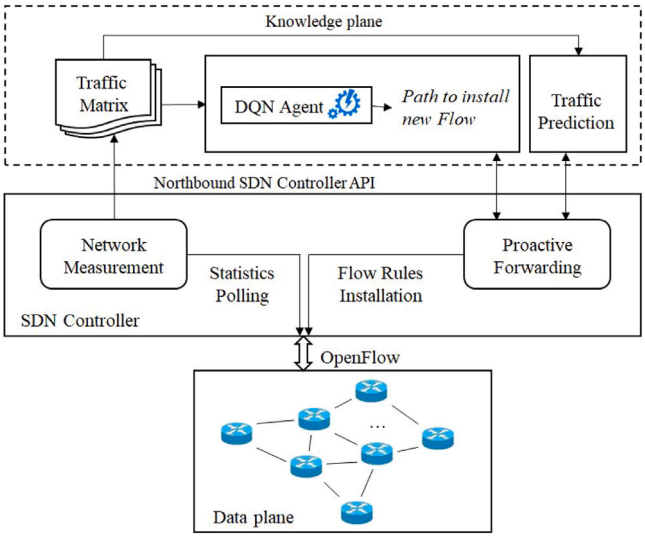

Fig. 1. DTPRO Architecture.

3. Proposed Deep Q-Network and Traffic Prediction based routing optimization (DTPRO) approach

Combining ML techniques with SDN is crucial to improve network performances. In this section, we first present our DTPRO approach. We start by explaining the global architecture. Thereafter, we describe the Network Measurement modules (i.e., Latency Measurement, Statistics), the DQN and Traffic Prediction modules. Finally, the mathematical model and the proposed heuristic will be described in the Proactive Forwarding module.

3.1. DTPRO architecture

Although SDN provides the centralization of network intelligence, a flexibility in traffic control and simplicity in network management and operation is still needed. KDN has been introduced as a new paradigm to bring the intelligence to the network management, using the telemetry data, which suggests to add the Knowledge plane (KP) to the conventional SDN paradigm, by adopting AI and cognitive system to build the network model.

In this context, we propose to design our framework according to the KDN paradigm, in which we exploit the control plane to have a global view of the network (cf. Fig. 1).

As depicted in Fig. 1, our proposed architecture consists of four planes: Data plane, Control plane, Management plane and Knowledge plane.

The Data plane consists of programmable forwarding devices in charge of data packet processing and forwarding. These devices have no embedded intelligence to take decisions and rely on the control plane to populate their forwarding tables and update their configurations based on the OpenFlow protocol.

The Control plane is considered as the brain of the SDN network, which incorporates the whole intelligence by centralizing the management and global view of the network in a specialized central controller, in which we deploy two main modules: Network Measurement and Proactive Forwarding modules. The Network Measurement module consists of two sub-modules: Statistics which continuously collects metrics such as the number of packets and bytes per flow to measure the throughput, and Latency Measurement. This latter sub-module measures continuously the network latency by periodically sending a packet probe to the data plane. On the other hand, the Proactive Forwarding module is responsible for determining the optimal routing strategy as

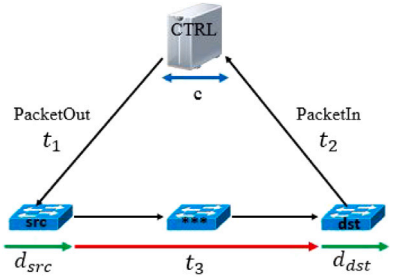

Fig. 2. Latency computation mechanism.

well as the predicted traffic from the KP by resolving an optimization problem as will be explained later in Section 3.5.

The Management plane ensures the correct operation and performance of the network by collecting the network measurement from the control plane, specifically from the Network Measurement module, in order to provide network analytic. The collected statistics will be analyzed and sent to the KP.

Finally, the KP exploits the control and the management planes by taking the data from the Management plane as input to be fed to ML algorithms, which convert them to the form of knowledge. Precisely, they learn the behavior of the network, by processing the collected statistics, then extract the optimal paths, representing the knowledge, to route flows by deploying a DQN agent, and finally, predict network congestion using prediction methods (i.e., LSTM, ARIMA, Linear Regression (LR)) in the Traffic Prediction module. Note that the routing strategy is determined by the DQN agent by exploiting the historical data of routing configurations.

In what follows, we detail further these modules.

3.2. Network Measurement

The Network Measurement module ensures the data plane monitoring based on the OpenFlow protocol, which is crucial in network management, and helps Operators to make decisions about Load Balancing, Routing, QoS, SLA and so on. The collection of statistics from the data plane can be either active or passive. In the active mode, the Controller sends and receives probe packets to the entire data plane network to measure statistics such as the Round-Trip-Time (RTT), Latency, Packet Loss. On the other hand, the passive mode corresponds to querying the statistics information from switches by using standard OpenFlow messages.

As stated earlier, the Network Measurement module consists of two sub-modules: Statistics and Latency Measurement. The Statistics module monitors the data plane according to the passive mode using the OpenFlow standard messages to read the information from the control plane. Specifically, it uses the `OFPStatsReply` message type, in which, the switch periodically reports its statistics through two types of messages: `PortStatsReply` and `FlowStatsReply`, to respectively measure the throughput and the per-flow size. Note that, the throughput can be measured based on the number of sent/received packet or bit reported in the `PortStatsReply` message.

The Latency Measurement module, on the other hand, uses the active monitoring mode by sending periodically a packet probe to the data plane to measure the latency based on the notifications' arrival times at the control plane. Specifically, it consists in firstly, measuring the time that takes a packet probe from the controller and traverse the path and return back to the controller ($Total_{Delay}$). Secondly, it measures the time that takes the packet probe to go from the controller to the first switch (i.e., t_1) and the last switch on the path (i.e., t_2), as shown in Fig. 2. (Tajiki et al., 2016; Barabas et al., 2011).

Some specific monitoring rules must be installed at the first and last switches (src, dst) respectively on the path that contains "`Send to Controller`", in order to measure the delay between the controller and the switches (i.e., t_1, t_2, as shown in Fig. 2 and to send the packet probe back to the controller at the last switch of the path. The switches between the first and the last switches in the path forward the packet probe by a pre-installed forwarding flow rules corresponding to the packet probe. The delay needed corresponds to the time t_3, and can be determined as follows:

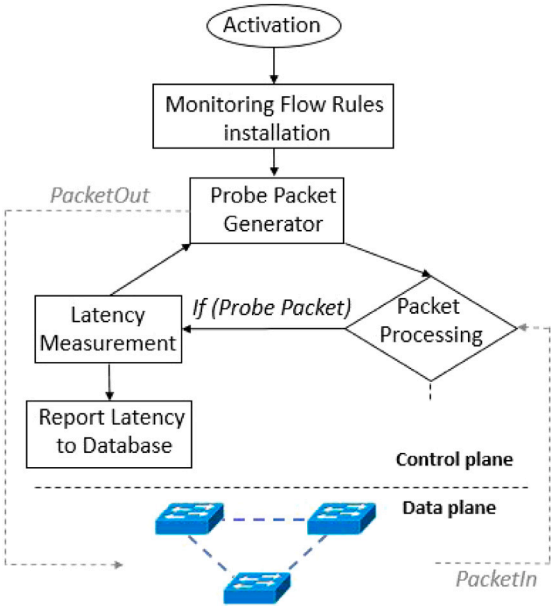

Fig. 3. Latency Measurement Design.

$$t_3 = Total_{Delay} - (t_1 + t_2 + d_{src} + d_{dst} + c) \quad (1)$$

Where d_{src}, d_{dst} are the processing times in the devices (src, dst) respectively, and c is the processing time in the controller. In order to accurately estimate the $Total_{Delay}$, t_1 and t_2 delays, the time is encapsulated in nanoseconds in the packet probe. Moreover, the monitoring rules installed to guide the packet probe do not interfere with the normal traffic.

It is worth noting that the Latency Measurement module is designed according to the Open Services Gateway Initiative (OSGI) standard that implements an application as a complete and dynamic component model. It is indeed composed of four components: Monitoring Flow Rules Installation, Probe Packet Generator, Packet Processing and Latency Measurement, as shown in Fig. 3. When activating the Latency Measurement module we start by installing the probe packet flow rules to guide the probe packet along the path to be monitored. The Probe Packet Generator will then send periodically a specific probe packet to the data plane, that matches the monitoring flow rules already installed by the previous component. Thereafter, the Packet Processing component listens to the incoming `PacketIn`, then, if the latter corresponds to the probe packet, the Packet Processing component extracts the times between the controller and switches and sends them to the Latency Measurement component that measures the latency as explained above and stores the latency in a centralized database.

3.3. Traffic prediction models

In order to avoid congestion and improve network performances, it is important to predict the future evolution of network traffic.

To do so, we propose here to use the well-known LSTM model to predict the network latency and compare it with two other prediction models: ARIMA, and LR. In what follows, we detail these three models and show how we adapt them to predict the E2E network latency.

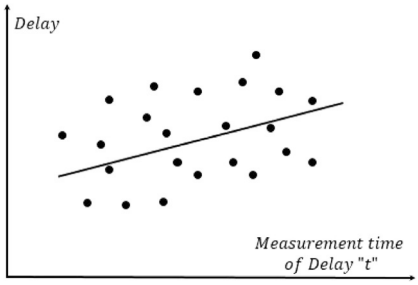

Fig. 4. Linear Regression.

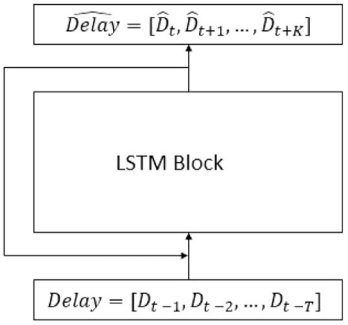

Fig. 5. LSTM based network Latency prediction.

3.3.1. Linear regression (LR)

it is a statistical prediction method that attempts to model the relationship between an explanatory variable and a dependent variable, by fitting a linear equation to observed data, where the objective is to predict the outcome of the dependent variables from the explanatory variables (Kavitha et al., 2016). Identifying the dependency among a single variable refers to the uni-variate regression as shown in the Eq. (2), where x is the explanatory variable, y is the dependent variable, λ and μ are constants and ε is the error:

$$\hat{y} = \lambda.x + \mu + \varepsilon \tag{2}$$

As described in Fig. 4, a valuable way of modeling the relationship between the measuring time of delays (i.e., the explanatory variable t) and estimated ones (i.e., the dependent variable $\widehat{Delay}$) is by using a correlation coefficient, which indicates the strength of the association of the observed data for the two variables. We can build the regression equation between the time of sampling and the delay, by using the equation in (3):

$$\widehat{Delay} = \lambda.t + \mu + \varepsilon \tag{3}$$

Where t is the sampling time, λ and μ are obtained from a set of measurements S: (i.e., Delay, Measurement time) so that the gap between the measurement and the model in (3) is minimized. Note that the regression quality is determined by the size of previous values (i.e., training data).

3.3.2. ARIMA model

Using this model, the latency can be considered as a stochastic process, expressed as a linear combination of p past observations (i.e., Latency Measurement): $D_{t-1}, D_{t-2}, \ldots, D_{t-p}$, and q past white noises: $e_{t-1}, e_{t-2}, \ldots, e_{t-q}$, together with a random error in the same time series, as shown in the following equation (Brockwell and A. Davis, 2002):

$$D_t = c + \phi_1 D_{t-1} + \cdots + \phi_p D_{t-p} + e_t + \theta_1 e_{t-1} + \cdots + \theta_q e_{t-q} \tag{4}$$

Where, p, the number of past observations represents the Auto-Regressive (AR) degree, q represents the number of Moving Average (MA) order and d is the degree of differentiation (Brockwell and A. Davis, 2002). It is worth noting that D_t is the latency to be predicted using previous samples of the latency time series.

Recall that the identification of the ARIMA model involves several steps where the mains are: (i) checking the stationarity to be able to use correctly the ARIMA model, (ii) order (p, d, q) must be estimated, we refer to Auto-Correlation Function (ACF) and Partial Auto-Correlation Function (PACF) to estimate these orders (Brockwell and A. Davis, 2002).

3.3.3. Long Short-Term Memory (LSTM)

with the continuous development of network services, the network traffic is constantly expanding showing more burst and more complexity. Consequently, the traffic evolution shows non-linearity. This makes the RNN, in particular its LSTM variant model, well placed to learn complex non-linear patterns. One of the advantages of LSTM is that several models can be used based on each type of time series forecasting such as the shape of the input/output of the NN. In this way, we propose to model the traffic latency prediction using LSTM model (Hochreiter and Schmidhuber, 1997) for multi-step time series forecasting as follows:

- LSTM Inputs:
  - Sample: S
  - Time Steps: T
  - Features: n
  - Learning Window: w
  - Units: nbr_units
  - Delay Vector: $\hat{D} = \{\hat{d}_{(u,v)}\}$
- LSTM Output:
  - Predicted Delay Vector: $\hat{D} = \{\hat{d}_{(u,v)}\}$

Where the samples S represents the number of training examples, the Timesteps T refers to the LSTM memory capacity and the Feature n is the amount of features in every time step. These three elements construct the three-dimensional structure $[Samples, Timesteps, Features]$ expected by the LSTM model. The Learning Window w refers to the number of previous time-slots to learn from in order to predict the future Delay vector. This parameter is used to avoid learning in long sequences that can result in high computational complexity. The nbr_unit parameter represents the number of neurons and finally the $Delay\ vector$ refers to the previous latency Measurements to be fed to the LSTM Model. The output of the LSTM model is the predicted vector of link delays. Fig. 5 shows the overall architecture of the proposed LSTM model.

It is worth noting that, before applying an LSTM model on the dataset, we need to transform the data as follows: (i) the Time series data must be stationary, ii) the Time series must be transformed into a supervised learning problem. More specifically, the previous observations are used to predict the current observation, and (iii) It is necessary to have a specific scale responding to the default tangent hyperbolic activation function $Tanh$ of the LSTM model.

3.4. Routing optimization model based on DQN

To meet the requested QoS, flows must be routed following the best routing strategy. As the traffic state (stream) depends on the type of data transported in the network, choosing the best routing path to that stream improves its QoS. To this end, we propose here to deploy a

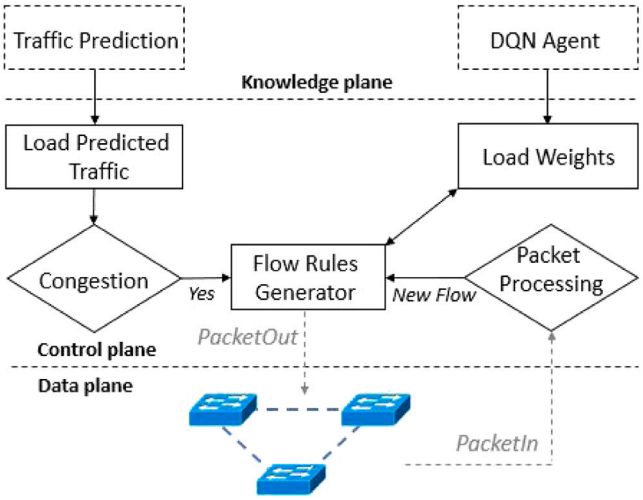

Fig. 6. Proactive Forwarding Design.

DQN agent that dynamically determines the optimal path. We model this DQN as follows:

Given a network topology represented as a non-oriented graph $G(V, E, C)$ where V, E and C are, respectively, the vertex, the edge and the link capacity sets, and $|V| = n$ represents the number of network nodes, the DQN agent interacts with the environment through three signals: State, Action, and Reward. "State" is the $(n \times n)$ Traffic Matrix representing the current network load. The "Action" taken by the agent is the link-weight vector, and the "Reward" r of the agent is related to the QoS parameters, which are mainly: the average of Network latency $(\overline{L})$, the average of Data rate $(\overline{W})$ and the average of Packet Loss $(\overline{PL})$. In this case, the Reward r can be determined as follows:

$$r = \alpha.\overline{W} - \beta.\overline{L} - \gamma.\overline{PL} \tag{5}$$

Where $\alpha, \beta, \gamma \in [0, 1]$ are the adjustable weights determined by the routing strategy. Our objective here is to determine the optimal policy π mapping the set of states to the set of actions in order to maximize the reward r. Note that the routing strategy is determined by a set of weights and periodically updated at the beginning of each time epoch T_{DQN}, which is initialized to 1 h in this work.

3.5. Proactive forwarding formulation and resolution

The Proactive Forwarding module is responsible for routing flows according to the optimal routing strategy, by using the current and predicted Traffic Matrix. It is designed according to the OSGI model and combines four components: (i) Load Predicted Traffic, (ii) Load Weights, (iii) Packet Processing and (iv) Flow Rules Generator, as shown in Fig. 6. The two sub-modules Load Predicted Traffic and Load Weights are respectively responsible for collecting the predicted traffic and the set of link weights from the KP layer. The Packet Processing listens to the `PacketIn` coming from the data plane and finally the Flow Rules Generator is responsible for flow routing. Two events automatically trigger the latter: the first one corresponds to generating flow rules for new incoming flows by using the collected weights by Load Weights sub-modules, to calculate their corresponding paths, and the second one happens when a congestion is detected, where the action is to reroute the current traffic to the less used paths.

The idea here is to give more accuracy to the routing strategy selected by the DQN agent as explained in the previous Section 3.4, by incorporating the prediction aspect, where the objective is to determine which link to route which flow in order to minimize the E2E network latency and balance the network load by minimizing link utilization.

In this context, the physical infrastructure can be represented by a capacitated graph $G(V, E, C)$, where V denotes the set of nodes or switches and E the set of edges representing the physical or virtual links in the network. Each link is characterized by its current and predicted propagation delay $d_{(u,v)}, \widehat{d_{(u,v)}}$, bandwidth capacity $C_{(u,v)}$ and threshold $TS_{(u,v)}$. We assume the network has m nodes ($|V| = m$), ne links ($|E| = ne$) and l flows. The current and predicted Traffic Matrix $TM, \widehat{TM}$ representing, respectively, the current and predicted volume of traffic flows between all pairs of origin and destination nodes in the network. Each flow f_i is characterized by its size $FS_{(f_i,u,v)}$ and the path to which is affected P_{f_i}.

The network may have the capacity limitation constraint. To this end, we define three variables $E_{(f_i,u,v)}$, $LU_{(u,v)}$, $MLU_{(u,v)}$ as follows:

$$E_{(f_i,u,v)} = \begin{cases} 1, & \text{if } f_i \in (u,v) \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

$$LU_{(u,v)} = \sum_{f_i \in F} (E_{(f_i,u,v)} \times FS_{(f_i,u,v)}) \tag{7}$$

$$MLU_{(u,v)} = C_{(u,v)} * \lambda_{(u,v)} \tag{8}$$

Where the symmetric Binary matrix $E = E_{(f_i,u,v)}$ denotes whether the flow rule f is allocated to the link (u, v) or not. Also $LU_{(u,v)}$ denotes the current link utilization and $MLU_{(u,v)}$ represents the Maximum Link Utilization of link (u, v). In the Eq. (8), λ depends on link characteristics and $C_{(u,v)}$ is the link capacity.

As the network traffic may have several classes based on latency sensitivity, it will be necessary to give more priority to those latency sensitive applications. To this end, we also define two variables: δ and st_{f_i,s_j}. We assume each switch has q priority, the variable δ represents the priority of flow f_i in switch s_j and the decision variable st_{f_i,s_j} denotes whether the flow f_i is being routed or waiting in the queue.

$$\delta = \begin{cases} \delta_{f_i,s_j}, & \delta_{f_i,s_j} \in [1,q], \text{if } f_i \in s_j \\ 0, & \text{otherwise} \end{cases} \tag{9}$$

$$st_{f_i,s_j} = \begin{cases} 1, & \text{if } f_i \text{ is routed} \\ 0, & \text{if } f_i \text{ is waited} \end{cases} \tag{10}$$

Hence, our problem can be formulated mathematically as a LP as follows:

- Objective:
  - Minimize $\tau.D + \nu.LU + \zeta.PL$
- Subject to:
  - Delay limitation: $\forall (u,v) \in E$: $Max(d_{(u,v)}, \widehat{d}_{(u,v)}) < TS_{(u,v)}$
  - Link capacity limitation: $\forall (u,v) \in E : LU_{(u,v)} < MLU_{(u,v)}$
  - Path capacity limitation: $\forall (u,v) \in FP : MLU_{(u,v)} - LU_{(u,v)} \leq Cap\_Av\_Path(FP)$
  - Path Flow priority: $\forall s_k \in Path\ P, \forall f_i, f_j \in s_k^2 :$ $\delta_{f_i,P} > \delta_{f_j,P} \Rightarrow \sum_{s_k \in P} st_{f_i,s_k} \geq \sum_{s_k \in P} st_{f_j,s_k}$
  - Demand satisfaction: $\forall (u,v) \in Path\ P, \forall f_i \in F :$ $\sum_{(u,v) \in Path, f_i \in F} FS_{(f_i,u,v)} = dem_i$

The delay and link capacity limitation constraints are specified so that we force each link to not be delayed and overloaded. The Path Flow priority constraint ensures that the flow with high priority must be routed first and finally the Demand satisfaction constraint ensures that the traffic demand dem_i sent through any path source src_i must be equal to that in the path destination dst_i. Our objective is to minimize the network delay D, the link utilization LU and the packet loss PL,

while the total demands are always satisfied. Note that τ, ν, ζ are the weighting factors related to the degree of importance of D, LU and PL, respectively. These latters are defined as follows:

$$D = \frac{1}{ne} \sum_{(u,v)\in E} d_{(u,v)} \tag{11}$$

$$LU = \frac{1}{ne} \sum_{(u,v)\in E} LU_{(u,v)} \tag{12}$$

$$PL = \frac{1}{ne} \sum_{f_i \in F} (dem_i - \sum_{(u,v)\in P_{f_i}} FS_{(f_i,u,v)}) \tag{13}$$

The formulated problem can be considered as a multi-commodity flow problem, which are known to be $NP\text{-}hard$. Furthermore, it is assumed to be solved by the SDN Controller for each incoming flow. However, as the size of the network and the number of flows increase, the computational complexity increases exponentially. Clearly, such approach is not feasible in practice, since it generates high overhead due to the frequent updates of the flow tables.

To cope with this problem, and reduce the computation time and complexity, we propose here a simple yet efficient heuristic algorithm, called DTPRO algorithm. DTPRO allows a high-quality traffic allocation, while minimizing the total network latency and packet loss. Algorithm 1 shows the detail of the proposed heuristic. It takes as inputs the current and predicted Traffic Matrix (TM and $\widehat{TM}$, respectively), the current and predicted matrix of link delays (D and $\widehat{D}$, respectively), and the matrix of link delays threshold TS that defines the maximum tolerable delays of each link (u, v). It is worth noting that, the DTPRO is executed for each incoming flow as well as when detecting a congestion.

Algorithm 1: DTPRO algorithm

```
1: procedure (G(V, E, C), TM, TM̂)
2:     schedule_every (T_DQN)
3:         Links_Weights ← get optimal Weights
4:                 from DQN agent
5:         Update_Net_Config(Links_Weights)
6:     end schedule
7:     t ← 0
8:     while (d_(u,v) > TS_(u,v) or d̂_(u,v) > TS_(u,v) or
       LU_(u,v) > MLU_(u,v)) do
9:         CP ← Get_Congested_Path(u, v)
10:        F ← Sorted_Flows_Priority(CP, δ)
11:        R ← F[t].flow
12:        SP ← Sorted_Backup_Paths(R)
13:        for each path p ∈ SP do
14:            if p can route R then
15:                Install_Rule(p, R)
16:                Remove_Rule(CP, R)
17:                Break
18:            end if
19:        end for
20:        t ← t + 1
21:    end while
22:    Adjust_Reward(α, β, γ)
23: end procedure
24: ω ← 10^-2, α, β, γ ← 10^-1
25: procedure ADJUST_REWARD(α, β, γ)
26:    if d_(u,v) > TS_(u,v) or d̂_(u,v) > TS_(u,v) then
27:        β ← β + ω
28:    if LU_(u,v) > MLU_(u,v) then
29:        α ← α + ω
30:        γ ← γ + ω
31: end procedure
```

Our algorithm works as follows. At the beginning of each time epoch T_{DQN}, it requests the DQN agent to get the optimal link weights $Links\_Weights$, then it updates the network configuration (lines 2–6).

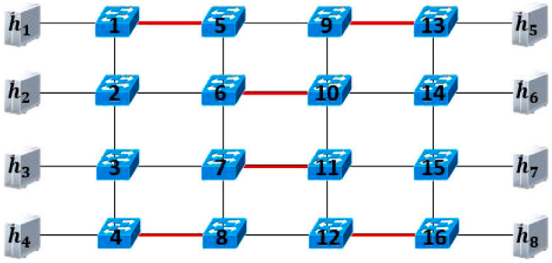

Fig. 7. Emulated Topology.

It is worth noting that, the T_{DQN} corresponds to the time interval to apply a new routing strategy obtained from the DQN agent. Moreover, to show the impact of using different network configurations or routing strategies, this parameter is initialized to 1 h. However, this parameter can be modified by the network administrator in the order of days or weeks.

Thereafter, since we measure continuously current and predicted Traffic Matrix, if a congestion occurs, in which the current or predicted link delay is greater than a certain threshold or the link is overloaded (line 8), our algorithm finds the flow rule R corresponding to the flow with maximum size in the congested path CP. Then, it sorts all other paths (denoted by SP) by the delay matching the flow rule R (lines 9–12). In this case, the flow rule R must be rerouted to a path in SP (lines 13–19). If no path in SP can accommodate the corresponding flow size, the flow is discarded and our algorithm goes to the next flow in the CP (line 20). Finally, it adjusts the parameters (α, β, γ) of the DQN reward function r, so that the DQN agent avoids a similar transition (line 22). The $Adjust\_Reward$ function (lines 25–31) adjusts the parameter β (Eq. (5)) if a link is delayed or predicted to be delayed. On the other hand, the α and γ parameters are adjusted when a link is overloaded or is predicted to be overloaded.

4. Performance evaluation

In this section, we evaluate the efficiency of our proposed approach. We start by presenting our environmental setup. Then, we present the experimental results.

4.1. Experimental setup

First, we implement the Network Measurement module (Latency Measurement and Statistics) as well as the Proactive Forwarding module as cooperating modules for the Java-based OpenFlow controller ONOS (Berde et al., 2014), based on our previous framework developed in Bouzidi et al. (2018). Then, the DQN agent and the Traffic Prediction are implemented based on Python,[1] which are dockerized on Docker[2] Containers. Note that the DQN agent and the Traffic Prediction interact with the Proactive Forwarding module based on the ONOS Northbound API. We used the network emulation tool OpenvSwitch (Pfaff et al., 2009) to implement the experimental topology illustrated in Fig. 7. To generate traffic among hosts, we used Iperf,[3] which consists of a set of flows between a set of hosts $(h_1, \ldots, h_8)$. The Network Measurement module collects statistics (network latency, throughput, and per-flow size) from the devices and reports those time-series statistics to the InfluxDB[4] database each time interval Ti which is equals to 5 seconds. The link-labels in Fig. 7 show the capacity C for each link.

[1] https://www.python.org/.
[2] https://www.docker.com/.
[3] https://iperf.fr/.
[4] https://www.influxdata.com/time-series-platform/influxdb/.

Table 1
DQN parameters for DTPRO.

Name	Values
Dense layers	2
State size	48
Action space size / Output shape	24,60,120
Q-target network update frequency	300
Learning rate	0.01
Discounted factor	0.95
Mini-batch size	32
Final exploration rate	0.2
Memory size	500 units
Number of episodes	120
Episode capacity	360000 steps

We built and trained the DQN model by using the Tensorflow[5] library, by deploying separately two NNs, one for *Q-Network* and the other for *Q-target*, which have the same architecture. The parameters of the DQN are illustrated in Table 1. In particular, both the *Q-Network* and the *Q-target* consists of 2 dense layers. In order to show the impact of the action space size, which corresponds to the number of network configurations for each input traffic matrix state, we trained separately three DQN agents with 24, 60 and 120 network configuration, respectively. The state vector size is 48, which corresponds to the input shape of each NN, and the outputs of the NNs for each training correspond to the actions space sizes 24, 60 and 120, respectively. Consequently, the architectures corresponding to each training are (48, 24), (48, 60) and (48, 120). During the training phase, we adopt ϵ-greedy method as action selection method and the final exploration rate is fixed at 0.2, while the *Q-target* parameters are copied from the *Q-Network* every 300 steps. The learning rate and discounted factor are, respectively, 0.01 and 0.95. Finally, as we trained three different DQN agents, each training corresponds to 120 episodes.

On the other hand, we built and trained the LSTM model by using Keras Library,[6] where the number of dense layers is 2. We used Adam (Kingma and Ba, 2014) for learning the NN parameters with a learning rate 0.01. We used *Relu* as activation function. The LSTM method outputs a predicted vector. To this end, we referred to the traffic matrices used for DQN, by dividing them into multiple input/output samples and the size of the output sample corresponds to the prediction interval Pi.

Note that, in order to improve the performances, the training of ARIMA, LSTM and DQN models is done offline. These trained models are then saved in such a way that only one step is needed to get the optimal path from the DQN agent or the predicted traffic from LSTM or ARIMA. In parallel, these models continue to learn from these new steps.

4.2. Prediction accuracy evaluation

To quantitatively evaluate the overall performance of our prediction models, we use the Root Mean Square Error (RMSE), defined as the difference between the predicted values and the actual values by computing the root of the average sum of squared errors. It can be expressed as follows:

$$RMSE(model) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{x}_i - x_i)^2} \tag{14}$$

where $\hat{x}_i$ and x_i are, respectively, the normalized predicted value and the normalized actual value for the same time interval and N corresponds to the total number of predictions or the size of the prediction interval Pi, which is initialized to 5 s.

[5] https://www.tensorflow.org/.
[6] https://keras.io/.

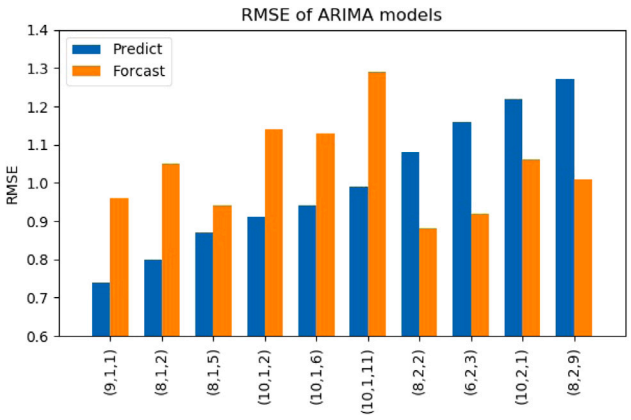

Fig. 8. Selecting ARIMA models while varying the p, d, q parameters.

4.3. Experimental results

In order to evaluate the performance of our proposed DTPRO solution, we first select the best predictions models to the network traffic evolution. Then, we evaluate the DQN model, while using different parameters and configurations and finally, we compare the proposed solution with different routing schemes, while taking into account the obtained traffic predictions models.

First, we estimate the best ARIMA model to the proposed network traffic evolution, which is done by estimating the ARIMA parameters (Brockwell and A. Davis, 2002): the order of the AR term (p), the order of the MA term (q) and the number of differencing needed to make the time series stationary (d). The time series should be stationary by having values around a defined mean. However, the network traffic could have non-stationary evolution over time. To this end, differencing the time series is a one way to make it stationary and the right order of differencing corresponds to (d). In our experiments, we referred to the ACF plot (Brockwell and A. Davis, 2002) for differencing and the Augmented Dickey Fuller (ADF) (Mushtaq, 2011) to check if the series is stationary. In this way, while the differenced series are not stationary, we increment d. The order of the AR term p corresponds to the number of lags ($x_{t-1}, x_{t-2}, \ldots, x_{t-p}$) that must be used as predictors. In this work, we referred to the PACF (Brockwell and A. Davis, 2002) to estimate the parameter p, by checking if there is a correlation between a specific lag and the time series. Finally, the parameter q corresponds to the number of lagged prediction errors needed in the ARIMA model and similar to PACF to estimate the parameter p, we used the ACF to estimate the parameter q.

Fig. 8 compares the prediction and forecasting accuracy of a set of best ARIMA models to our network traffic, using the RMSE metric. The parameters are estimated based on methods presented above. In this way, the forecasting term is used to indicate the prediction of future values given past values of time series, while the prediction term is used to do estimation whether in future, current or past. From this figure, we can see that models with $d = 1$ give a good prediction and forecasting accuracy regarding the network traffic. The evolution can be repeated over time. We hence refer to this model with best prediction accuracy as the one with ($p = 9, d = 1, q = 1$). This model will be used in our next experiments.

Regarding the LSTM model, in order to ensure that its estimation accuracy is good enough and to avoid the over-fitting problem when the network is trained, it is required to find the best number of neurons and the number of training epochs. To this end, we plot in Fig. 9 the average of the Loss function and the average of RMSE under different number of training epochs. In this way, we measure both the RMSE and Loss function for each 500 epochs. For the sake of simplicity, we fixed the number of nodes to 100 nodes. In this experiment, we decided

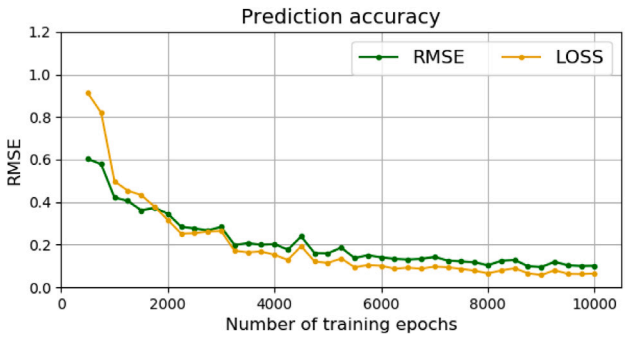

Fig. 9. LSTM Loss and RMSE under different number of training epochs.

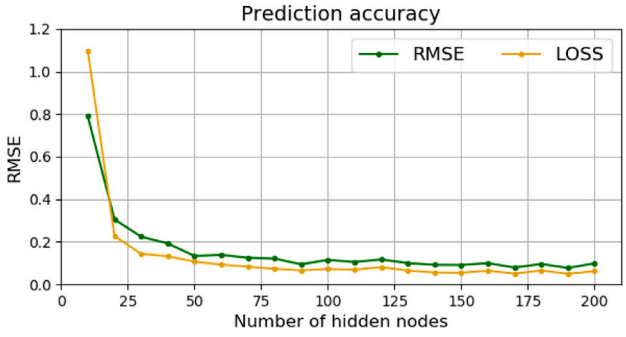

Fig. 10. LSTM Loss and RMSE under different number of hidden nodes.

about the number of training epochs when the Loss function converges and starts to be stable, which is ensured from this figure by both the RMSE and Loss function. We can see also that both the RMSE and Loss function start to be stable and achieve a good estimation accuracy when the number of training epochs is 9000. The next experiment will focus on identifying the number of hidden nodes.

As stated earlier, the number of hidden nodes of the LSTM network is crucial to achieve a stable network configuration. To this end, we plot in Fig. 10 the average of the Loss function and the average of RMSE under different number of hidden nodes by increasing the number of hidden nodes by 10 for each measurement. Similar to the previous experiment, we decided about the number of hidden nodes when the Loss function converges and starts to be stable. We can clearly see from that figure that both the Loss function and the RMSE converge and start to be stable and achieve a good estimation accuracy when the number of hidden nodes is 150.

From the results obtained in the two previous experiments, we decided about the best LSTM model, which corresponds to 9000 training epochs and 150 hidden nodes, alongside to the initial parameters presented above in this section.

On the other hand, the LR parameters λ and μ, as indicated in Eq. (3), are estimated online, which means that when the prediction is triggered, the Traffic Prediction based LR looks for the N previous measurement, to identify the parameters λ and μ, then it predicts the future evolution of network traffic by using the Eq. (3). In Fig. 11, we plot the variation of λ, μ parameters while training the LR prediction model. We can see that these parameters are dynamically changed while changing the data intervals.

Fig. 12 compares the prediction accuracy of the different methods presented earlier (i.e., LSTM, ARIMA and LR) by measuring the impact of the prediction interval Pi on the RMSE metric, which is increased by 3 s for each measurement. Recall that the ARIMA model used in this experiment is $(p = 9, d = 1, q = 1)$ and the used LSTM model is the one with 9000 training steps and 150 hidden nodes and the LR model is dynamically estimated.

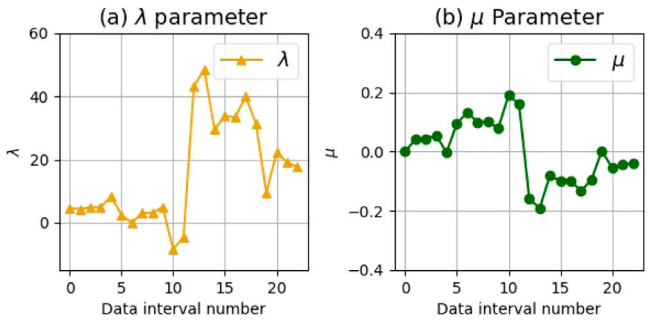

Fig. 11. LR parameters λ, μ under different data intervals.

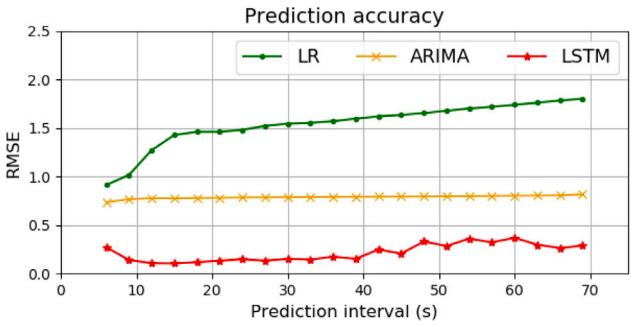

Fig. 12. RMSE of LSTM, ARIMA and LR prediction methods.

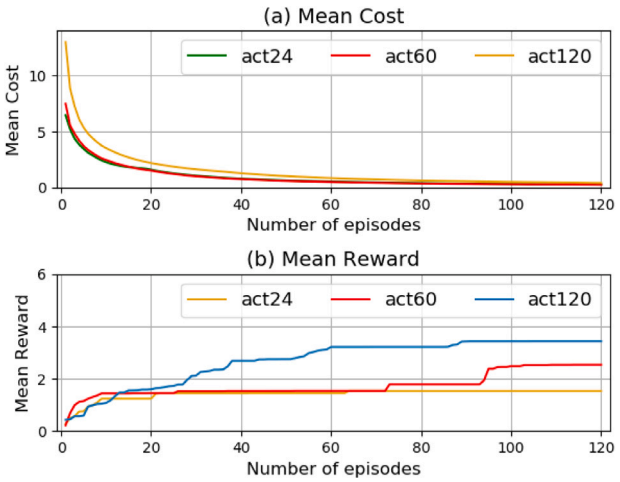

Fig. 13. Impact of varying the action space size while training the DQN agent.

From Fig. 12, we can see that the RMSE of LSTM is quasi-stable when increasing the prediction interval Pi. We can see also that ARIMA achieves a clear stability when increasing the prediction interval Pi, due to the nature of the network traffic, which is periodic in a certain level. However, the increase in the LR RMSE is clearly visible, which can be interpreted as the increasing of the distance between the predicted points in the line $\lambda.t + \mu$ and the dispersed traffic points. Furthermore, we can see that ARIMA outperforms LR due to its capacity to estimate the correlation with the previous lags of the time series. Finally, it is clearly visible that LSTM outperforms all others prediction methods due to its capacity to learn long term dependencies. In what follows, we determine the best DQN model for the proposed network traffic evolution.

Recall that, the principal idea of the proposed DQN model is to learn the best policy mapping the set of states (i.e., traffic matrices) to the set of actions (i.e., network configurations), while maximizing a

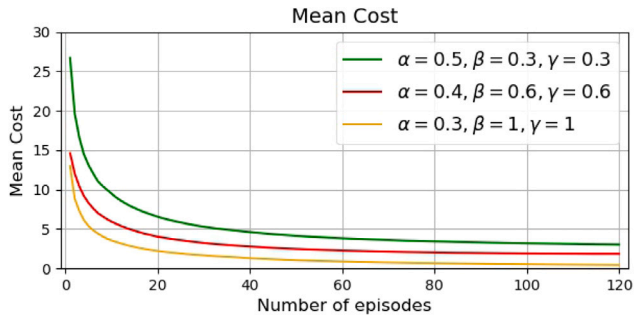

Fig. 14. Impact of varying the reward function parameters α, β, γ while training the DQN agent.

numerical reward defined in Eq. (5). In this way it is required to find the best action structure, in order to ensure a good estimation accuracy of the DQN model, while avoiding the over-fitting problem when the DQN is trained. To this end, we plot in Fig. 13 the evolution of the average of the Loss function and the average of the reward function under different number of training episodes and the three action space sizes (24, 60, 120). The DQN is trained based on the initial parameters as stated earlier in this section. In the experiment, we decide about the best action structure when the Loss function converges and the reward function starts to be stable.

From Fig. 13 we can see that the Loss function for the three action space configuration converge. However, we can observe from Fig. 13(b) that the more we increase the action space capacity, the more the reward function is increased. The increasing reflects the existence of appropriate new network configurations (i.e., network paths), so the queues of nodes could be unloaded and lead to decreasing both packet loss and network latency. As a result from this experiment, we decide to take the action space size as 120.

As mentioned in the Eq. (5), the reward function is related to Data rate (W), Network latency (L) and Packet Loss (PL) with parameters α, β and γ, respectively. These parameters play an important role to determine, in one side, the importance of each factor and on the other side the convergence of the Loss function. To this end, we plot in Fig. 14 the evolution of the average of the Loss function under different number of training episodes, while changing the reward function parameters. From Fig. 14 we can see that, the less we give importance to both packet loss and network latency, the more the Loss function converges and gives small values. As a result, we decide that the best DQN model corresponds to the one with action space size 120, and the reward function parameters ($\alpha = 0.3, \beta = 1, \gamma = 1$). In the next experiments, we assess the performance of our proposed heuristic while using the aforementioned DQN model and prediction methods.

In order to evaluate the performance of our proposed solution DTPRO, which corresponds to combining the best obtained DQN and LSTM models, we compare it with the following baselines:

- Hop-count (HC) based routing, which is the default routing metric used by ONOS.
- Reduced DTPRO by using the obtained DQN model for routing optimization and at the same time disabling the Traffic Prediction module.
- DTPROv1, which consists in using the obtained DQN model for routing optimization and the obtained ARIMA model (instead of LSTM) for Traffic Prediction.
- DTPROv2, which consists in using the obtained DQN model for routing optimization and LR for Traffic Prediction.

Fig. 15 plots the packet loss, the delay and the link utilization for all schemes (DTPRO, DTPROv1, DTPROv2, Reduced DTPRO, and HC). We can see that the HC approach causes obviously considerable packet loss and increases the link utilization, since all the flows are forwarded

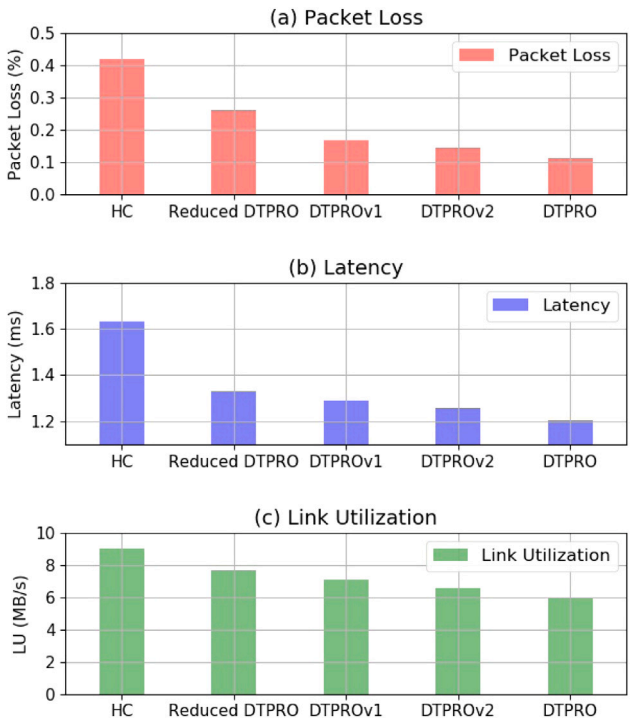

Fig. 15. Packet Loss, Delay and Link utilization under rule placement algorithm based on DQN with and without prediction.

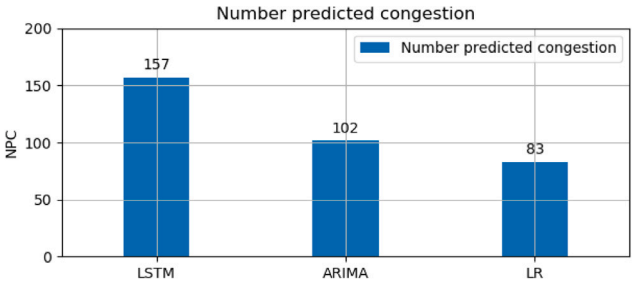

Fig. 16. Number of predicted congestion while combining the DQN with different traffic predictions methods.

to shortest paths that shares the same link with minimum capacity. On the other hand, when using the DQN agent without prediction (i.e., the Reduced DTPRO scheme), considerable packet loss is still observed, increasing thus the link utilization, due to the incapacity of DQN to predict the future evolution of network traffic. Finally, we can see that DTPRO outperforms all other schemes, especially DTPROv1 and DTPROv2, where the packet loss, delay and link utilization are decreased. This is related to the high accuracy of LSTM in predicting network congestion, compared to ARIMA and LR prediction methods.

We plot in Fig. 16 the number of predicted congestion, while combining the DQN with different traffic predictions methods. It is worth noting that, the congestion happens when the network latency or traffic load exceed a certain threshold which is fixed to 80% of each link capacity. In the proposed network traffic evolution from 3000 states defined above in this section, we force certain states to overload the network for some specific network configurations. From this figure, we can see that LSTM outperforms others methods due to its high accuracy for predicting the future evolution compared to ARIMA and LR.

To improve the Quality of Experience (QoE), it is necessary to provide wider varieties of services than just a single class of best-effort service (Shenker, 1995). To this end, we propose to evaluate our

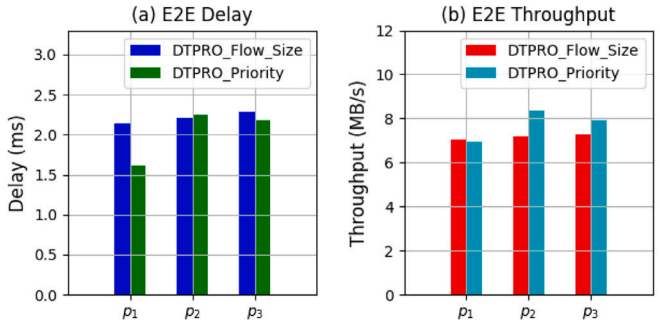

Fig. 17. DTPRO with and without priority.

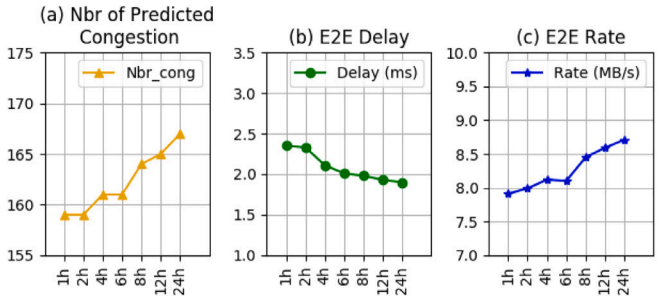

Fig. 18. Impact of varying the T_{DQN} time interval on the network performance.

DTPRO approach with and without *priority* according to the following simulation scenario: we use initially specific paths for Latency Sensitive Application (LSA), Throughput Sensitive Application (TSA) and Packet Loss Sensitive Application (PLSA). Then, we force these paths to be delayed and congested. Based on a specific set of priorities (p_1, p_2, p_3), Fig. 17 compares the following baselines:

- DTPRO_Flow_Size corresponds to the proposed DTPRO heuristic which sorts the traffic flows according to their size to reroute the traffic once there is a congestion.
- DTPRO_Priority corresponds to the proposed DTPRO heuristic which sorts the traffic flows according to their *priority* to reroute the traffic once there is a congestion.

The set of priorities are selected as follows:

- p_1: $p(LSA) = 10$, $p(TSA) = 5$, $p(PLSA) = 1$
- p_2: $p(LSA) = 5$, $p(TSA) = 10$, $p(PLSA) = 1$
- p_3: $p(LSA) = 1$, $p(TSA) = 5$, $p(PLSA) = 10$

From Fig. 17(a) we can see that, when giving more priority to LSA in p_1, DTPRO_Priority performs better than DTPRO_Flow_Size in decreasing the E2E Delay. However, when giving more priority to TSA in p_2 and PLSA in p_3 the E2E Delay for both schemes is close to each other. On the other hand, we can see from Fig. 17(b) that, when giving more priority to TSA in p_2 or PLSA in p_3, DTPRO_Priority performs better than DTPRO_Flow_Size in decreasing the E2E Throughput. However, when giving more priority to LSA in p_1, the performances for both schemes are close with no improvement in the E2E Throughput. The reason of improving the performances is that the flows corresponding to these application types in the congested paths are routed first. This allows the DTPRO approach to be used in a context of Network Slicing where each slice is dedicated for specific traffic types.

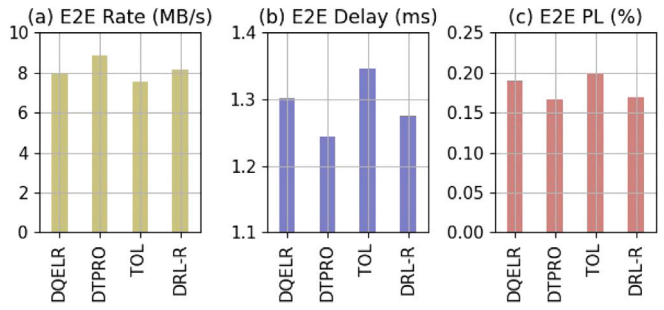

Fig. 19. Comparative analysis between DTPRO, TOL, DRL-R and DQELR.

Fig. 18 plots the impact of varying the T_{DQN} time interval on the network performance (i.e., Number of Predicted Network Congestion, the E2E Delay and the E2E Throughput or Rate). This parameter is varied in the set of intervals: [1h, 2h, 4h, 6h, 8h, 12h, 24h]. Recall that, the T_{DQN} parameter corresponds to the time interval to apply a new routing strategy obtained from the DQN agent.

From this figure we can observe that both the Network Congestion and E2E throughput increase with the increase of the time interval T_{DQN}. On the other hand, the E2E Delay decreases with the increase of T_{DQN}. The reason is that the DQN state space increases with the increasing of the T_{DQN} time interval, where new transitions and new network configurations are detected. Moreover, the LSTM model in the Traffic Prediction module is able to predict new traffic evolutions, which leads to predict new network congestion and rerouting the traffic more efficiently, decreasing thus the E2E Delay and increasing the E2E Throughput.

Finally, Fig. 19 shows a comparative analysis between the proposed approach DTPRO and three main approaches indicated in the related works: (1) Deep Reinforcement Learning-based Routing (DRL-R) (Liu, 2019), which represents a traffic optimization solution based on DQN, (2) Traffic Optimization based on LSTM (TOL) (Azzouni and Pujolle, 2017), which corresponds to the use of the LSTM RNN framework for predicting the network traffic matrix, and (3) Deep Q-Network-based Energy and Latency-aware Routing (DQELR) (Su et al., 2019), which is a routing optimization solution based on DQN.

As the design of DTPRO is modular, in which the knowledge plane consists of two separated modules: (i) routing optimization based on DQN and (ii) Traffic Prediction based on LSTM, the comparison of our proposed approach with DRL-R and DQELR is done by deactivating the traffic prediction module and using only the routing optimization module. On the other hand, for the TOL approach, we used only the Traffic Prediction module.

We used the same experimental topology, shown in Fig. 7, for all approaches. For DRL-R (Liu, 2019), the DQN Q-Network and Q-Target consist of two hidden layers with 30 neurons. The Relu corresponds to the activation function and the learning rate is fixed to 0.001, as proposed in Liu (2019). For the DQELR model (Su et al., 2019), the input layer consists of four nodes. Three hidden layers with 300, 150 and 15 nodes, respectively, are also used. Finally, the TOL approach consists in using the LSTM model in Azzouni and Pujolle (2017). All these models are trained based on the aforementioned parameters. Then, we measured the E2E Delay, the E2E Throughput and the E2E Packet Loss for each approach for a time interval of 24h, as shown in Fig. 19.

From this figure, we can see that the TOL approach causes obviously more packet loss and delay, and provides less network throughput compared to all other approaches. This can be explained by the fact that, this approach does not take into account the best routing strategy,

increasing thus the number of congestion links and leading to a new network behavior with low prediction accuracy. Second, we can see that DRL-R and DQELR perform better than TOL since both approaches consist in defining routing strategies by training DQN agents, while maximizing certain rewards related specifically to network throughput and delay. On the other hand, DRL-R performs slightly better than DQELR, since adding DQN based routing decisions to packets in the DQELR approach impacts the performances and decentralizes the routing decisions, which is contradictory to the SDN design. Finally, the superiority of our proposed DTPRO method over all other approaches, in terms of E2E Throughput, E2E Delay and E2E Packet Loss, is clearly visible. The reasons are: i) different from DRL-R and DQELR, the action of our proposed DQN model is to modify the links weights vector, which is equivalent to defining all flow paths in one action instead of defining the path for each incoming flow, as defined in DRL-R and DQELR, ii) the TOL approach predicts only the traffic matrix without considering the routing strategy, and (iii) combining the DQN agent with the traffic prediction based on LSTM allows unseen transitions from the DQN agent to be predicted by the traffic prediction module by considering the previous experiences in the DQN agent and at the same time the future behavior of network traffic in LSTM.

5. Conclusion

In this paper, we presented a method for rules placement in Software-defined Networks based on real-time statistics measurement and traffic prediction, which are implemented separately as a cooperating modules on both the Control Plane and the KP layers. By taking advantage of the KP, the network routing is dynamically optimized by deploying a DQN agent that dynamically determines the optimal policy mapping the set of states (Traffic Matrices) to the set of actions (changing the vector of link weights). In addition, we proposed to deploy a Traffic Prediction module based on the well known prediction methods LSTM, in order to avoid congestion. To this end, we have mathematically formulated the QoS-aware routing problem as a LP, where the corresponding optimization problem is to minimize the total network latency, packet loss and link utilization. To solve this optimization problem, a simple yet efficient heuristic algorithm was proposed and implemented, called Deep Q-Network and Traffic Prediction based Routing Optimization (DTPRO) that dynamically interacts with the external DQN agent module to get the set of link weights, and the Traffic Prediction to avoid congestion. Experimental results using the ONOS controller and OpenvSwitch, showed that the DQN agent is able to learn a mapping between the Traffic Matrix state and the set of link weights to route the traffic flows. However, DQN itself is not well adapted for predicting the future evolution of the network traffic. By combining DQN with Traffic Prediction, we showed that network latency, packet loss and link utilization can be decreased. Moreover, we showed that LSTM achieves a high estimation accuracy, which outperforms traditional prediction methods, and decreases both E2E delay and packet loss.

As future work, we plan to further exploit our solution in the context of distributed SDN controllers, where their number, locations as well as the associated set of data plane switches can be optimized by deploying a DQN agent.

CRediT authorship contribution statement

EL Hocine Bouzidi: Conceptualization, Methodology, Investigation, Software development and integration, Writing – original draft. **Abdelkader Outtagarts:** Data curation, Software, Supervision, Validation, Reviewing and editing. **Rami Langar:** Supervision, Validation, Writing – review & editing. **Raouf Boutaba:** Validation, Reviewing and editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work was partially supported by the FUI SCORPION project, France (Grant No. 17/00464).

References

Affandi, A., Riyanto, D., Pratomo, I., Kusrahardjo, G., 2015. Design and implementation fast response system monitoring server using Simple Network Management Protocol (SNMP). In: 2015 International Seminar on Intelligent Technology and Its Applications (ISITIA). pp. 385–390.

Azizi, M., Benaini, R., Mamoun, M.B., 2015. Delay measurement in OpenFlow-enabled MPLS-TP network. Mod. Appl. Sci. 9 (3), 90.

Azzouni, A., Pujolle, G., 2017. A long short-term memory recurrent neural network framework for network traffic matrix prediction. arXiv:1705.05690.

Barabas, M., Boanea, G., Rus, A.B., Dobrota, V., Domingo-Pascual, J., 2011. Evaluation of network traffic prediction based on neural networks with multi-task learning and multiresolution decomposition. In: 2011 IEEE 7th International Conference on Intelligent Computer Communication and Processing. pp. 95–102. http://dx.doi.org/10.1109/ICCP.2011.6047849.

Berde, P., Gerola, M., Hart, J., Higuchi, Y., Kobayashi, M., Koide, T., Lantz, B., O'Connor, B., Radoslavov, P., Snow, W., et al., 2014. ONOS: towards an open, distributed SDN OS. In: Proceedings of the Third Workshop on Hot Topics in Software Defined Networking. pp. 1–6.

Boutaba, R., Salahuddin, M., Limam, N., Ayoubi, S., Shahriar, N., Estrada-Solano, F., Caicedo Rendon, O., 2018. A comprehensive survey on machine learning for networking: Evolution, applications and research opportunities. J. Internet Serv. Appl. http://dx.doi.org/10.1186/s13174-018-0087-2.

Bouzidi, E.H., Luong, D., Outtagarts, A., Hebbar, A., Langar, R., 2018. Online-based learning for predictive network latency in software-defined networks. In: 2018 IEEE Global Communications Conference (GLOBECOM). pp. 1–6. http://dx.doi.org/10.1109/GLOCOM.2018.8648063.

Bouzidi, E.H., Outtagarts, A., Langar, R., 2019. Deep reinforcement learning application for network latency management in software defined networks. In: 2019 IEEE Global Communications Conference (GLOBECOM). pp. 1–6. http://dx.doi.org/10.1109/GLOBECOM38437.2019.9013221.

Boyan, J., Littman, M., 1999. Packet routing in dynamically changing networks: A reinforcement learning approach. Adv. Neural Inf. Process. Syst. 6.

Brockwell, P., A. Davis, R., 2002. An introduction to time series and forecasting. Technometrics 39, http://dx.doi.org/10.1007/978-1-4757-2526-1.

Chowdhury, S.R., Bari, M.F., Ahmed, R., Boutaba, R., 2014. PayLess: A low cost network monitoring framework for Software Defined Networks. In: 2014 IEEE Network Operations and Management Symposium (NOMS). pp. 1–9.

Cisco NetFlow, 2004. Online.

Cortez, P., Rio, M., Rocha, M., Sousa, P., 2006. Internet traffic forecasting using neural networks. In: The 2006 IEEE International Joint Conference on Neural Network Proceedings. pp. 2635–2642.

Feng, H., Shu, Y., 2005. Study on network traffic prediction techniques. In: Proceedings. 2005 International Conference on Wireless Communications, Networking and Mobile Computing, 2005, vol. 2. pp. 1041–1044. http://dx.doi.org/10.1109/WCNM.2005.1544219.

Han, Y., Seo, S., Li, J., Hyun, J., Yoo, J., Hong, J.W., 2014. Software defined networking-based traffic engineering for data center networks. In: The 16th Asia-Pacific Network Operations and Management Symposium. pp. 1–6. http://dx.doi.org/10.1109/APNOMS.2014.6996601.

Hochreiter, S., Schmidhuber, J., 1997. Long short-term memory. Neural Comput. 9, 1735–1780. http://dx.doi.org/10.1162/neco.1997.9.8.1735.

Hyun, J., Hong, J.W., 2017. Knowledge-defined networking using in-band network telemetry. In: 2017 19th Asia-Pacific Network Operations and Management Symposium (APNOMS). pp. 54–57.

Kaur, S., Singh, J., Ghumman, N., 2014. Network programmability using POX controller. http://dx.doi.org/10.13140/RG.2.1.1950.6961.

Kavitha, S., Varuna, S., Ramya, R., 2016. A comparative analysis on linear regression and support vector regression. In: 2016 Online International Conference on Green Engineering and Technologies (IC-GET). pp. 1–5.

Khodayari, S., Yazdanpanah, M.J., 2005. Network routing based on reinforcement learning in dynamically changing networks. In: 17th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'05). pp. 5–366. http://dx.doi.org/10.1109/ICTAI.2005.91.

Kingma, D., Ba, J., 2014. Adam: A method for stochastic optimization. In: International Conference on Learning Representations.

Liu, W., 2019. Intelligent routing based on deep reinforcement learning in software-defined data-center networks. In: 2019 IEEE Symposium on Computers and Communications (ISCC). pp. 1–6.

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., Turner, J., 2008. OpenFlow: Enabling innovation in campus networks. Comput. Commun. Rev. 38, 69–74. http://dx.doi.org/10.1145/1355734.1355746.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M., 2013. Playing atari with deep reinforcement learning. arXiv:1312.5602.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A., Veness, J., Bellemare, M., Graves, A., Riedmiller, M., Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D., 2015. Human-level control through deep reinforcement learning. Nature 518, 529–533. http://dx.doi.org/10.1038/nature14236.

Mushtaq, R., 2011. Augmented Dickey Fuller test. SSRN Electron. J. http://dx.doi.org/10.2139/ssrn.1911068.

ONF, 2012. Software-defined networking: the new norm for networks. Online.

Pfaff, B., Pettit, J., Amidon, K., Casado, M., Koponen, T., Shenker, S., 2009. Extending networking into the virtualization layer. In: Hotnets.

Phaal, P., Panchen, S., McKee, N., 2001. Online.

Pham Tran Anh, Q., Hadjadj-Aoul, Y., Outtagarts, A., 2019. Deep Reinforcement Learning Based QoS-Aware Routing in Knowledge-Defined Networking: Potentiale und Grenzen in der Aus- und Weiterbildung studentischer Tutorinnen und Tutoren. pp. 14–26. http://dx.doi.org/10.1007/978-3-030-14413-5_2.

Reza Parsaei, M., Mohammadi, R., Javidan, R., 2017. A new adaptive traffic engineering method for telesurgery using ACO algorithm over Software Defined Networks. Eur. Res. Telemed. 6, http://dx.doi.org/10.1016/j.eurtel.2017.10.003.

SDN Controller Ryu, 2021. Online.

Shenker, S., 1995. Fundamental design issues for the future internet. IEEE J. Sel. Areas Commun. 13 (7), 1176–1188.

Su, Y., Fan, R., Fu, X., Jin, Z., 2019. DQELR: An adaptive deep Q-network-based energy- and latency-aware routing protocol design for underwater acoustic sensor networks. IEEE Access 7, 9091–9104.

Tajiki, M.M., Akbari, B., Mokari, N., 2016. QRTP:Qos-aware resource reallocation based on traffic prediction in software defined cloud networks. In: 2016 8th International Symposium on Telecommunications (IST). pp. 527–532. http://dx.doi.org/10.1109/ISTEL.2016.7881877.

van Adrichem, N.L.M., Doerr, C., Kuipers, F.A., 2014. OpenNetMon: Network monitoring in OpenFlow software-defined networks. In: 2014 IEEE Network Operations and Management Symposium (NOMS). pp. 1–8.

Wang, M.-H., Wu, S.-Y., Yen, L.-H., Tseng, C.-C., 2016. PathMon: Path-specific traffic monitoring in OpenFlow-enabled networks. In: 2016 Eighth International Conference on Ubiquitous and Future Networks (ICUFN). pp. 775–780.

Yu, C., Lan, J., Guo, Z., Hu, Y., 2018. DROM: Optimizing the routing in software-defined networks with deep reinforcement learning. IEEE Access 6, 64533–64539. http://dx.doi.org/10.1109/ACCESS.2018.2877686.

Yu, C., Lumezanu, C., Sharma, A., Xu, Q., Jiang, G., Madhyastha, H.V., 2015. Software-defined latency monitoring in data center networks. In: International Conference on Passive and Active Network Measurement. Springer, pp. 360–372.

EL Hocine Bouzidi received the engineering degree in computer science from the National School of Computer Science of Algiers (ESI), in 2010, and the M.Sc. degree in computer science in safe embedded and mobile systems from le CNAM, France, in 2016. He is currently pursuing the Ph.D. degree in telecom engineering with Nokia Bell Labs and Paris Est Marne la Vallée University, France. He joined National School of Computer Science of Algiers (ESI), as a System and Network Engineer, from 2012 to 2015. He joined Orange Lab, France, for his master's degree internship, as a software developer. His research interests include 5G networks, networks resource management, software defined networks (SDN), and network function virtualization (NFV).

Abdelkader Outtagarts IEEE senior member is a senior researcher and technical leader in NFV and SDN orchestration, machine learning and automation in Nokia Bell Labs. Abdelkader received the M.Sc. and Ph.D. degrees from the INSA de Lyon in France, in 1990 and 1994 respectively, on automation of energy systems. In 1999, he receives a M.Sc. on software engineering from Ecole de Technologie Supérieure of Montreal in Canada. His professional experience of over 20 years, on information systems and telecommunications, energy efficiency, software engineering, cloud computing, data mining, NFV micro services and SDN orchestration and automation, is mainly acquired in France and Canada in R&D teams (Alcatel-Lucent, Nextenso, Hydro-Quebec, UTILICASE and SCII Technology), in research laboratories in Lyon (INSA), Montreal (ETS, CRIM) and Nozay (Nokia Bell labs).

Rami Langar (Member, IEEE) received the M.Sc. degree in network and computer science from University Pierre and Marie Curie (now Sorbonne University) in 2002, and the Ph.D. degree in network and computer science from Telecom ParisTech, Paris, France, in 2006. He was a Post-Doctoral Research Fellow with the School of Computer Science, University of Waterloo, Waterloo, ON, Canada, from 2006 to 2008, and an Associate Professor with LIP6, University Pierre and Marie Curie, from 2008 to 2016. He is currently a Full Professor with University Gustave Eiffel (UGE), France. He is involved in many European and National French research projects, such ANR 5G-INSIGHT, ANR ABCD, FUI SCORPION, FUI ELASTIC Networks, FUI PODIUM, MobileCloud (FP7), GOLDFISH (FP7), etc. His research interests include resource management in future wireless systems, cloud-RAN, network slicing in 5G/5G+/6G, software-defined wireless networks, and mobile cloud offloading. He was a co-recipient of the Best Paper Award from the IEEE/IFIP International Conference on Network and Service Management 2014 (IEEE/IFIP CNSM 2014). He was the Chair of the IEEE ComSoc Technical Committee on Information Infrastructure and Networking (TCIIN) from January 2018 to December 2019.

Raouf Boutaba (M'93–SM'01–F'12) received the M.Sc. and Ph.D. degrees in computer science from the Pierre and Marie Curie University, Paris, France, in 1990 and 1994, respectively. He is currently a Professor of computer science with the University of Waterloo, Waterloo, ON, Canada. His research interests include resource and service management in networks and distributed systems. Dr. Boutaba is a Fellow of the IEEE, the Engineering Institute of Canada, and the Canadian Academy of Engineering. He served as a Distinguished Speaker for the IEEE Computer and Communications Societies. He is the founding Editor-in-Chief of the IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT (2007–2010), and he is on the editorial boards of other journals. He was the recipient of several Best Paper Awards and other recognitions, such as the Premier's Research Excellence Award, the IEEE Hal Sobol Award in 2007, the Fred W. Ellersick Prize in 2008, the Joe LociCero Award and the Dan Stokesbury Award in 2009, the Salah Aidarous Award in 2012, and the McNaughton Gold Medal in 2014.