PIQoS: A Programmable and Intelligent QoS Framework

Udaya Lekhala
Computer Science
Dalhousie University, Canada
udaya.lekhala@dal.ca

Israat Haque
Computer Science
Dalhousie University, Canada
israat@dal.ca

***Abstract*—Network management and Quality of Service (QoS) support are becoming more challenging with the increase in network traffic, size, and service requirements. To meet these challenges, we need a programmable and intelligent framework for automated QoS support; static or threshold-based approaches are not adequate. We propose a software-defined and machine-learning-based intelligent QoS framework called PIQoS. PIQoS enables software-defined networking (SDN) controllers to effectively, efficiently, and autonomously react, in a vendor agnostic way, to changes in network links by (1) placing link failure detection and recovery in the data plane and (2) applying supervised learning methods to the tasks of dynamically detecting link failures and congestion and appropriately reconfiguring the network so that it can continue to provide the required QoS as link properties change over time. To test the performance of a system based on PIQoS, we performed extensive simulation experiments in Mininet, using real network topologies. We also studied the comparative performance of several supervised learning methods applied to our specific detection and correction problems to determine which methods are most appropriate for this domain. Our simulation results highlight the potential efficacy of the PIQoS framework when applied in real networks.**

I. Introduction

Network monitoring and Quality of Service (QoS) support have become a challenge with an increase in network traffic, their dynamics, and service requirements. Quality factors such as throughput, delay, and path length are important for service providers to maintain and provide a consistent user experience. However, network states change unpredictably, which demands dynamic policy configuration and management. Thus, it is not efficient to solve this problem using static policy management [1]–[3] or dynamic policy management with predefined thresholds [4]. Instead, QoS can be improved by automating the process of error detection and the root cause analysis. This automated QoS framework can be designed using a programmable and intelligent network.

Software-defined networking (SDN) [5] allows network operators a programmatic and elegant way of dynamically implementing a wide-range of network policies and rapidly deploying new services. For instance, PolicyCop [4] configures network based on the dynamically changing policies. However, PolicyCop considers predefined policies and thresholds and consults a network manager in the case of a violation with a missing policy. PolicyCop furthermore deploys restoration-based link failure recovery scheme, i.e., upon detecting a link failure associated switch contacts the controller to recover from a failure. However, restoration-based recovery introduces a delay that can impact the required QoS.

In this paper, we propose a programmable and intelligent framework called *PIQoS* for QoS-based policy automation. In PIQoS, we first push the link failure recovery at the data plane using protection-based recovery scheme. In this scheme, a controller proactively installs alternative route between every source-destination pairs. We implement this recovery using the Fast Failover Group (FFG) of the OpenFlow protocol [6]. Note that the OpenFlow protocol enables communication between the data and control plane of an SDN architecture. The protection-based recovery scheme improves both the delay and throughput by avoiding the communication delay between the data and control planes.

QoS requirements are different in different networking applications. Machine learning can improve the QoS by understanding the traffic patterns in a network and suggesting a network manager with a better policy management in less time. Machine learning has been used in traffic classification and QoS improvement; however, there is no work on dynamic QoS-based policy management through root cause analysis of error conditions in networks using machine learning. Thus, in PIQoS we use flow-level traffic for root cause analysis and design two models using supervised machine learning algorithms. We consider supervised algorithms because of their prediction accuracy over unsupervised ones. The first model detects various errors in a network by analyzing the network state traffic and the second model predicts the root cause of the error to update corresponding policy for QoS provisioning. We consider both the link failure and network congestion to design and test the models, which are implemented in the controller to detect and react to link failures or congestions.

We use real network topology like USNET [7] in Mininet [8] emulator to evaluate the performance of PIQoS. In Mininet, we use a Ryu controller and a set of Open Virtual Switches [9]. The evaluation results reveal that PIQoS improves the link failure recovery time and network throughput compared to its counterpart. The results furthermore illustrate that the decision tree is the right supervised algorithm for the use cases considered in this paper. In a nutshell, we demonstrate that the learning based intelligent QoS provisioning is useful compared to a static or a threshold based QoS management.

The rest of the paper is organized as follows. We present necessary background to understand PIQoS in Section II and the related work on QoS-based policy management in Section III. Next, we describe the proposed PIQoS framework for QoS-based policy automation in Section IV. Section V provides the details of experimental setup. The following section presents the details of the data derived for the models. The experimental evaluation results in Section VII demonstrate the effectiveness of the proposed framework following some concluding remarks in Section VIII.

II. Background

In this section, we present the necessary background to understand the operation of PIQoS. Link failure can occur in any networks. In SDN design, two types of link failure recovery approaches can be deployed: *restoration* and *protection* [10]. In the restoration scheme, a data plane switch contacts the controller upon detecting a link failure to receive alternative route configuration, which introduces communication delay. In the protection scheme, backup routes are configured before a failure occurs. Switches can locally detect a failure and redirect the affected traffic to alternative route without communicating the controller, which reduces the delay.

In SDN architecture, OpenFlow protocol supports Fast-Failover Group (FFG) to implement the failure recovery at the data plane. In particular, this switchover can be implemented using the Flow and Group tables of OpenFlow 1.3 [11] protocol. A switch maintains a group table with a number of active buckets. Each bucket is associated with a port from a route and only a single bucket is active at a time. The incoming packet flows through the port from an active bucket. In the case of a link failure, the next active port and bucket is chosen to redirect the affected traffic.

In machine learning, supervised algorithms use labeled data sets to create models, while unsupervised learning tries to discriminate patterns in unlabeled data. Each of these algorithms is better suited for a particular type of application (e.g., classification, regression or clustering) and the method to be used depends on the way the problem is being formulated. Additionally, different learning algorithms (e.g., decision trees, neural networks or support vector machines (SVM)) can be used to solve the same problem, and the best choice depends on the requirements of each scenario (e.g., performance and accuracy constraints).

In this paper, we consider both the supervised and unsupervised algorithms as the former offers better accuracy, whereas the latter one avoids labeling training data set. Thus, we test both algorithms to find the suitable algorithm for the proposed learning models of PIQoS. In addition, we consider different algorithms from both the supervised and unsupervised algorithms to choose a suitable one. In PIQoS, decision trees and k-means offer the best performance. In the following, we briefly outline these two algorithms.

In a decision tree algorithm, data is stored in a tree-like structure, where a node, branch, and leaf-node represent test, decision, and class label, respectively. Thus, a decision tree path defines the data classification rules. In our implementation, we used ID3 algorithms, an attribute that best classify training data is chosen as the root and repeat the same process at each branch until all attributes are covered and a complete tree is obtained. The attributes can be categorized based on their information gain, and an attribute with the highest information gain is the root node [12].

K-means clustering is an algorithm that partitions data points or observations into clusters, where each cluster of data points has a centroid defined by those data points. At the beginning of the clustering, we can randomly choose K centroids and iteratively optimize their positions [12]. Thus, we can find the best value of K and corresponding centroids using the training data set. Then, a test point belongs to a cluster whose centroid is the closest to that test point [12].

III. Related Work

In this section, we present existing works that are closely related to the proposed PIQoS framework. Various policy enforcement frameworks have been proposed for QoS provisioning [3], [13]–[15]. Most of these existing works target inflexible QoS architectures, which lack a broader network picture, reconfigurability, and adaptability [16]. SDN enables network programmability through hardware abstraction and global network view. A set of works exploited SDN architecture to realize efficient QoS provisioning.

QoS provisioning based on predefined classes of traffic with static QoS parameters is proposed in [17] and a similar predefined threshold-based QoS provisioning is presented in [18]. Tegueu *et al.* defined an architecture that guarantees dynamic QoS requirement of applications and efficient network-resource utilization, where an application-specific QoS demand is identified using deep packet inspection (DPI) over flow-based packet forwarding [19]. However, multiple applications may have similar QoS requirement and fall in the same QoS class. Thus, Wang *et al.* used DPI and semi-supervised learning algorithm to design a QoS-aware traffic classification framework for fine-grained traffic engineering [20].

PolicyCop [4] implemented an automatic QoS-based policy enforcement framework, to meet SLAs in a software-defined network. In the case of any policy violation, a policy adaptation module takes action accordingly to predefined policies. In the case of a missing policy, PolicyCop consults the network manager to install the newly required policy manually. Al-Jawad *et al.* used Neural network to learn the congestion that can violate QoS policies [21]. The authors deployed either rerouting or rate limiting in the case of policy violation to readjust the QoS requirements. The proposed PIQoS offers automatic QoS provisioning by deploying a learning module in the controller, which predicts the reasons for failures and congestion to take quick recovery actions.

IV. PIQoS Framework

In this section, we describe the policy automation framework, PIQoS. First, it pushes the link failure detection and recovery at the data plane to improve the delay and throughput,

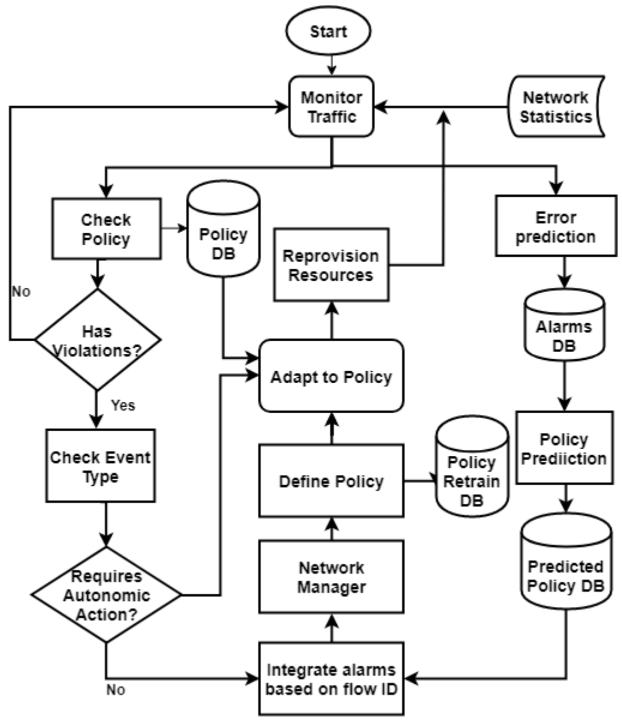

Fig. 1. The workflow of the proposed PIQoS framework.

which also scales with a large network. PIQoS framework furthermore has an learning-based policy automation module in the controller. It collects flow-level statistics from data plane switches. The controller monitors the network for both the policy violation and corresponding root cause analysis. The workflow of PIQoS framework is presented in Figure 1.

The static module: The network state traffic is sent to the *check policy* module, which checks for any policy violation based on corresponding predefined parameters and threshold at the initial stage of model development. In the case of any policy violation, *event classifier* classifies the event into one of the predefined cases. Depending on the classification a predefined action is taken; in the case of a missing action, the new policy is added to the system. We consider the above policy management of PIQoS as static, which mainly helps the adaptive ML-based model to adapt over time by proving additional classified traffic.

The adaptive module: In addition, the monitored traffic is passed to train the *error prediction* model, which predicts any errors or unusual network states. These states are then sent to the *alarms database*. All alarms from different modules are integrated using flow IDs and then ranked according to their priority. The priority is assigned based on the error prediction model outcome of minor, major, or initial threshold. The alarms are then propagated to the *policy prediction* model, where the root cause of the errors and corresponding policy are predicted. The associated data are then forwarded to the *predicted policy database (PPDB)*.

The network manager has access to the PPDB to observe the critical issues with predicted reasons and solutions. The network manager can define a new policy for the observed errors to accommodate any future events. The newly defined policies are accumulated in the policy database which can be used to retrain the models on a timely basis. As the model is trained over time, it becomes more accurate in predicting solutions. Thus, the learning models help the network manager resolve any unusual issues promptly, where prediction is based on past experiences.

V. Experimental Setup

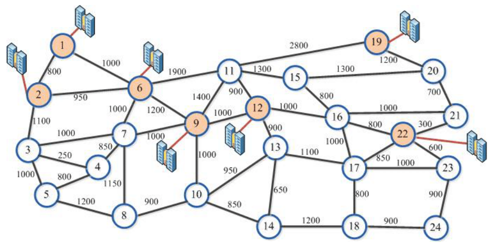

Fig. 2. The USNET topology.

In this section, we outline the experimental setup used to evaluate the performance of PIQoS. We evaluate the performance of the proposed framework in Mininet emulator using a Ryu controller. The controller controls the USNET topology (shown in Figure 2) formed out of a collection of software switches (OVS). We use *iperf* to generate traffic and a Ryu controller collects the statistics on network states at a regular interval. The collected statistics are fed to the policy management module deployed in the controller. We use Scikit and Weka to implement the proposed learning models.

In order to evaluate the resiliency performance of PIQoS, we consider a ten node mesh topology as well as the USNET topology, where each switch has an associated host. We measure the end-to-end delay and throughput to compare and contrast the performance of PIQoS with PolicyCop. We define the delay as the total time a packet takes to travel from a source to a destination. Applications like VoIP and online games require a small delay to meet the QoS. The supervised machine learning models are trained and tested on the USNET topology, where link failure is tested with twenty-four switches each having one host, whereas congestion is checked with the same number of switches each having four hosts.

VI. Model Evaluation Data

A machine learning model can be trained and tested using either offline or online data. The former option allows gathering a large amount of data for model training and testing, whereas online or real-time network steaming can be used as feedback or input to the model [22]. The streaming data is mostly used in the production network to detect and analyze error conditions on-the-fly. In this paper, we consider offline data to demonstrate the effectiveness of the proposed PIQoS framework; however, it can be deployed in a production network. We configure the Ryu controller to collect statistics at a regular interval to capture a wide range of features like

TABLE I
SAMPLE DATA FOR POLICY PREDICTION MODEL

Time stamp	Flow ID	Source Node	Dest. Node	Source Out-port	Dest in Port	Packets	Bytes	Link speed	Source Outport Capacity	Dest inport Capac-ity	Source port Speed	Dest port Speed	Delay (ms)	Class
2018-11-1116:23:27	32	17	8	5	8	2	2132	0	10	10	0	0	7	Link-Fail
2018-11-1116:23:28	32	8	6	3	5	2	2132	0	10	10	0	0	7	Link-Fail
2018-11-1116:28:28	112	8	11	2	7	439	658340	22.8	10	10	22.1	25.3	2.371	Congestion

throughput, delay, number of packets and bytes, port speed and capacity, and flow ID to evaluate our model. The captured data has associated timestamp to help the network manager to track incidents.

TABLE II
THE LIST OF FEATURES USED IN PIQOS.

Feature	Explanation
Time stamp	When a flow starts at a link
Flow ID	ID of a flow or a path
Source Node	Source node of a link in a flow
Destination Node	Destination node of a link in a flow
Source Outport	Port from which a source node sends packets
Destination Inport	Port from which a destination node receive packets
Packets	Total number of packets sent in a link
Bytes	No of bytes sent in a link
Linkspeed (B/Sec)	Speed of a link
Source Outport Capacity	Maximum capacity of a source outport
Destination Inport Capacity	Maximum capacity of a destination inport
Sourceport Speed (B/s)	Speed at which a source outport sends packets
Destination port Speed (B/s)	Speed at which a destination inport receives packets
Delay (ms)	Time taken for a packet to be transferred from a source to a destination in a flow
Error Prediction Classification	A flow has an issue or not (error/no error)
Policy Prediction Classification	Classification type (link failure/congestion)

TABLE III
THE NUMBER OF FLOWS AND LINKS USED FOR TRAINING AND TESTING MODELS.

ML Models	Total Flows	Total Links	Training Flows	Training Links	Testing Flows	Testing Links
Error Prediction	318	1020	225	714	93	306
Policy Prediction	151	465	106	326	45	139

In the case of error prediction model, we use parameters like Flow ID and associated source-destination hosts, ports, the corresponding number of bytes and packets, and timestamp. We also gather information about throughput and delay. Data collection considered a different level of granularity like a switch, port, or flow. The sample data for the policy prediction model is presented in Table I. We consider a total of 1500 instances for the error prediction model and 800 instances for the policy prediction model. We consider link failure and congestion as the unusual conditions in the network, which can be extended to cover other use cases. Note that due to the space limitation we are not presenting sample data from error prediction model, which is similar to that of policy prediction model.

We use the set of features shown in Table II for the proposed error and policy prediction models. There are fifteen features, which are chosen based on the standard parameters and metrics used in real networks for policy management and QoS provisioning. We plan to explore other features in our future research. The total number of flows and links used for training and testing our models is shown in Table III. In the case of supervised learning, we use labeled training data to train the error prediction model about error conditions like link failure and congestion. The labeled error data is then used to train the policy prediction model to classify link failure vs. congestion. In the case of unsupervised learning, we follow the same procedure except that the data does not have any labeling.

We use k-fold cross-validation to test the proposed models to avoid over-fitting and selection bias problems. The cross-validation also gives an insight on how a model can be generalized to an independent data set. In the k-fold cross-validation, we partition the training data set into five bins. At each of the k iterations, one bin is used for testing and rest of the k-1 bins are used to train the model.

VII. DISCUSSION ON THE EVALUATION RESULTS

A. *Resiliency of PIQoS*

In this section, we present the resiliency and efficiency of PIQoS, which is compared with PolicyCop. The evaluation results are shown in Figure 3 and Figure 4. The policy management module of the controller gathered the network statistics every three seconds. Thus, PolicyCop can learn any link failure event once a new set of statistics is gathered. Then, Policycop reactively installs an alternative route to redirect the affected traffic. In PIQoS, we pushed that recovery at the data plane using restoration scheme, i.e., the controller installs alternative route at each switch to recover from a failure. Thus, upon detecting a link failure, a switch uses FFG of OpenFlow protocol to locally redirect the traffic to an alternative route without controller's intervention.

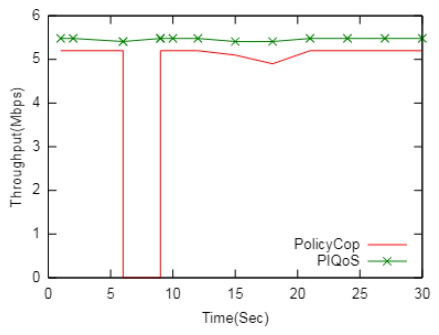

Fig. 3. The average throughput with local link failure recovery.

The average throughput of PolicyCop and PIQoS are shown in Figure 3. The result illustrates that the performance of PolicyCop drops in between two consecutive statistics gathering events as the controller needs to learn a failure event to reconstruct a new route. PIQoS, on the other hand, locally recovers from a link failure, which immediately improves the throughput. Local recovery scheme furthermore reduces control packet propagation between the data and control plane to scale with a large network.

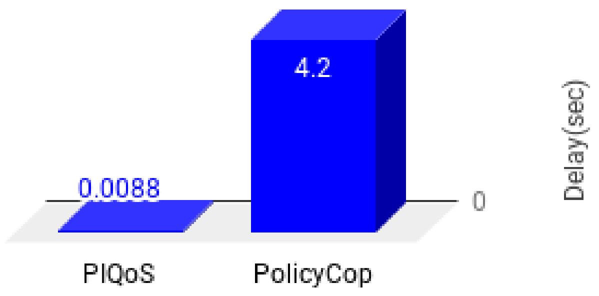

Fig. 4. The average delay with local link failure recovery.

The average delay experienced by PolicyCop and PIQoS is presented in Figure4. The link failure recovery time of PolicyCop is on the order of seconds, whereas it is on the order milliseconds in PIQoS. It is because PolicyCop reactively restores the route, which involves communications between data and control plane. Thus, we conclude that protection link failure recovery at the data plane is useful to improve both the delay and throughput to offer better QoS.

B. Intelligence of PIQoS

PIQoS proposes dynamic policy management using machine learning. The PIQoS framework can significantly improve network error detection and recovery and avoid time-consuming and error-prone network diagnosis and policy management. We propose two machine learning models to dynamically detect error conditions and update policy accordingly.

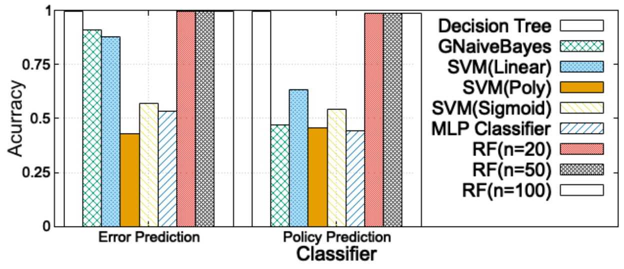

Fig. 5. The accuracy of different supervised algorithms for error and policy prediction models.

The error prediction model predicts whether there is any error conditions in the observed data using predefined thresholds. The error data is then sent to the policy prediction model for root cause analysis, i.e., to identify the actual error and its location. The automatic policy update module can use the prediction outcome to update the associated policy. In the case of a missing policy, a network manager can add one. In either case, the errors and corresponding predictions are also sent to the network manager as an alarm. This notification helps the manager to identify any unusual error rates and the root cause behind that errors.

We have chosen supervised learning to design both of our models because of their accuracy. At first we have evaluated different supervised machine learning algorithms to find the best algorithm. In order to choose an appropriate supervised algorithms, we have done an initial investigation on Naive Bayes, Support Vector Machine (SVM), Random forests (RF) and decision trees. The results are illustrated in Figure 5.

RF consists of several decision trees and offers good prediction accuracy. However, this accuracy cannot be generalized similar to decision trees. The accuracy of Naive Bayes and SVM is similar; however, they tend to under-fit our models. Among all the algorithms decision tree offers the best accuracy for both the models. Furthermore, decision trees can analyze a large volume of data using standard computing resources in a reasonable time and are not sensitive to imbalanced data.

TABLE IV
THE ACCURACY OF DECISION TREE AND RANDOM FOREST

Classifier	Error Prediction	Policy Prediction
Decision Trees	0.9986013986	0.9967741935
RF(n=20)	0.9985915493	0.9870967742
RF(n=50)	0.9985915493	0.9903225806
RF(n=100)	0.9971929479	0.9903225806

Thus, we have chosen decision trees, in particular, the ID3 algorithm in our models. We have used other decision tree algorithms like C4.5, where ID3 offers the best results. Also, it is simple, accurate, and computationally efficient. The results in Table IV furthermore illustrate that both of our proposed models offer high accuracy.

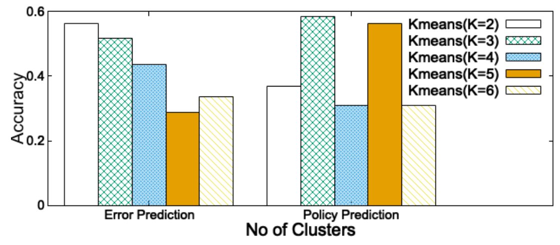

Fig. 6. The accuracy of k-means clustering algorithm for error and policy prediction models.

C. Comparison with Unsupervised Learning

We have observed that the accuracy of our models using decision tree is quite impressive. However, supervised learning algorithms require labeling training data, which may not be feasible for large networks. Thus, we design a prediction model using unsupervised machine learning algorithm. We have first analyzed the performance of k-means and DBSCAN algorithms to choose the best unsupervised algorithm. The results in Figure 6 and Figure 7 illustrate that k-means offer better accuracy compared to DBSCAN. In particular, the DBSCAN algorithm is not suitable for high dimensional data because of the curse of dimensionality problem. We have not tested local outlier factor and auto-encoders, which is part of our future work.

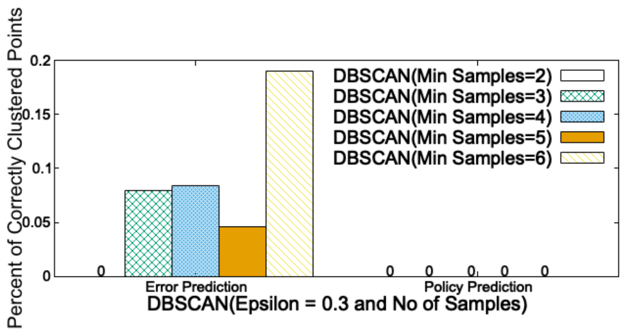

Fig. 7. The accuracy of DBSCAN algorithm for error and policy prediction models.

In Figure 6, we observe that both of our models offer their best accuracy for k=2 clusters. The cluster members are nicely fit in their respective group when there are two clusters, and the corresponding accuracy is around 50%. Thus, the supervised decision tree algorithm offers significantly higher accuracy compared to the unsupervised algorithms. We conclude that there is a tradeoff between the prediction accuracy and computation complexity of the learning models if we consider supervised vs. unsupervised algorithms. In our future work, we plan to explore semi-supervised algorithms to balance between the accuracy and complexity.

VIII. Conclusions

At present, companies spend a significant amount of time to diagnosis their network to detect and recover from various error conditions that violate QoS and SLA. In this paper, we have shown that using intelligent models in network monitoring and management helps better QoS provisioning and SLA. In particular, we have designed a programmable and smart QoS framework called PIQoS. It first pushes link failure recovery at the data plane to improve the delay and throughput. Furthermore, the proposed framework offers two supervised machine learning models for efficient network state diagnosis and selecting corresponding management policies. We implement and evaluate the proposed framework in a realistic simulation environment and demonstrate that proposed models can accurately predict link failure or network congestion and let the system update management policy accordingly.

References

[1] J. Babiarz, K. Chan, and F. Baker, "Configuration guidelines for DiffServ service classes," Tech. Rep., 2006.

[2] I. Minei, "MPLS DiffServ-aware traffic engineering," *Juniper Networks*, 2004.

[3] L. Lymberopoulos, E. Lupu, and M. Sloman, "An adaptive policy-based framework for network services management," *Journal of Network and Systems Management*, vol. 11, no. 3, pp. 277–303, 2003.

[4] M. F. Bari *et al.*, "PolicyCop: An autonomic QoS policy enforcement framework for software defined networks," in *IEEE SDN For Future Networks and Services (SDN4FNS)*, 2013, pp. 1–7.

[5] M. Casado *et al.*, "Ethane: Taking control of the enterprise," in *ACM SIGCOMM Comp. Comm. Review*, vol. 37, no. 4. ACM, 2007, pp. 1–12.

[6] N. Beheshti and Y. Zhang, "Fast failover for control traffic in software-defined networks," in *Global Communications Conference (GLOBECOM), 2012 IEEE*. IEEE, 2012, pp. 2665–2670.

[7] "USNET topology," https://www.researchgate.net/figure/Network-topologies-aNSFNET-bUSNET_fig3_321952636.

[8] "Mininet," http://mininet.org/.

[9] "Open vSwitch," https://www.openvswitch.org/.

[10] D. R. Fonseca *et al.*, "A survey on fault management in software-defined networks," *IEEE Comm. Surveys Tutorials*, vol. 19, no. 4, pp. 2284–2321, June 2017.

[11] Open Networking Foundation, "Openflow switch specification," September 2012.

[12] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning – Data Mining, Inference, and Prediction*, 2009.

[13] N. Samaan and A. Karmouch, "An automated policy-based management framework for differentiated communication systems," *IEEE Journal on Selected Areas in Comm.*, vol. 23, no. 12, pp. 2236–2247, 2005.

[14] S. Shanbhag and T. Wolf, "Automated composition of data-path functionality in the future internet," *IEEE Network*, vol. 25, no. 6, 2011.

[15] S. Cabuk *et al.*, "Towards automated security policy enforcement in multi-tenant virtual data centers," *Journal of Computer Security*, vol. 18, no. 1, pp. 89–121, 2010.

[16] F. Le Faucheur and W. Lai, "Requirements for support of differentiated services-aware MPLS traffic engineering," Tech. Rep., 2003.

[17] A. T. Oliveira *et al.*, "SDN-based architecture for providing QoS to high performance distributed applications," in *IEEE Symposium on Comp. and Comm. (ISCC)*, June 2018.

[18] F. Volpato, M. P. D. Silva, A. L. Goncalves, and M. A. Dantas, "An autonomic QoS management architecture for Software-Defined Networking environments," in *2017 IEEE Symposium on Comp. and Comm. (ISCC)*, July 2017, pp. 418–423.

[19] F. Tegueu *et al.*, "Towards application driven networking," in *IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*, June 2016.

[20] P. Wang, S.-C. Lin, and M. Luo, "A framework for QoS-aware traffic classification using semi-supervised machine learning in SDNs," in *IEEE International Conference on Services Computing (SCC)*, July 2016.

[21] A. Al-Jawad *et al.*, "Policy-based QoS management framework for software-defined networks," in *International Symposium on Networks, Comp. and Comm. (ISNCC)*, June 2018.

[22] R. Boutaba *et al.*, "A comprehensive survey on machine learning for networking: evolution, applications and research opportunities," *Journal of Internet Services and Applications*, vol. 9, no. 1, p. 16, 2018.