Computer Networks 184 (2021) 107706

Contents lists available at ScienceDirect

Computer Networks

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

PrePass-Flow: A Machine Learning based technique to minimize ACL policy violation due to links failure in hybrid SDN

Muhammad Ibrar [a], Lei Wang [a,f,g,*], Gabriel-Miro Muntean [b], Aamir Akbar [c], Nadir Shah [d], Kaleem Razzaq Malik [e]

[a] School of Software, Dalian University of Technology, China
[b] School of Electronic Engineering, Dublin City University, Ireland
[c] Department of Computer Science, Abdul Wali Khan University Mardan (AWKUM), Pakistan
[d] Department of Computer Science, COMSATS University Islamabad, Wah Campus, Pakistan
[e] Department of Computer Science, Air University, Multan Campus, Pakistan
[f] Key Lab of Ubiquitous Network and Service Software of Liaoning Province, China
[g] Peng Cheng Laboratory, China

ARTICLE INFO

Keywords:
Hybrid SDN
Machine Learning
ACL
Link Failure Prediction
Network reachability

ABSTRACT

The centralized architecture of Software-Defined Networking (SDN) reduces networking complexity and improves network manageability by omitting the need for box-by-box troubleshooting and management. However, due to both budget constraints and maturity level of the SDN-capable devices, organizations often are reluctant to adopt SDN in practice. Therefore, instead of migrating to a pure SDN architecture, an incremental SDN deployment strategy is preferred in practice. In this paper, we consider an incremental SDN deployment strategy known as *hybrid SDN* - involving simultaneous use of both SDN switches and legacy switches. The links connected to an SDN switch are called SDN links, and the rest are called legacy links. An SDN controller can directly poll the status of the SDN links via the connected SDN switches. At the same time, the status of the legacy links passes through SDN switches and reaches the controller, causing delay. As a result, the controller does not have the current status of legacy links in real-time. This delay may lead to undesired outcomes. For example, it causes network reachability problems due to Access Control List (ACL) policies. Therefore, to minimize the impact of network-layer failure in hybrid SDN, we propose a Machine Learning (ML) based technique called *PrePass-Flow*. PrePass-Flow predicts link failures before their occurrence, recomputes the locations of ACL policies, and installs the ACL policies in the recomputed locations in a proactive manner. The main objective of PrePass-Flow is to minimize the ACL policy violations and network reachability problems due to ACL policies in case of link failures. For the link status prediction, PrePass-Flow uses two supervised ML-based models: 1) a Logistic Regression (LR) model, and 2) a Support Vector Machine (SVM) model. Testing results show that the LR model performs better than both the SVM model and an existing approach in terms of Packet Delivery Ratio (PDR) and ACL policy violations. For instance, the LR model's accuracy is 4% better, precision is 5% higher, sensitivity is 10% better, and Area Under the Curve (AUC) is 6% greater than the SVM model's corresponding results.

1. Introduction

Software-Defined Networking (SDN) [1,2] has emerged as a response to the limitations and complexity of the legacy network management. The basic idea of the SDN paradigm is that it separates the control plane from the data plane by offloading control plane functionalities from all network devices (i.e., routers, switches, and Access Points (AP)) to a logically centralized controller. However, due to budget constraints and SDN maturity level, organizations are often reluctant to adopt SDN in practice. For example, Google [3] spent about 8 years to replace its data center infrastructure with SDN devices. Some of the reasons behind the slow migration to an all SDN infrastructure and its implications are discussed in [1,4]. Instead, the researchers proposed an incremental approach to deploy SDN, replacing gradually the legacy devices with SDN-enabled devices. This strategy is termed *hybrid SDN* [1] and is a promising avenue that paves the way to a wide adoption of SDN in practice [4,5].

* Corresponding author.
E-mail addresses: mibrar@mail.dlut.edu.cn (M. Ibrar), lei.wang@dlut.edu.cn (L. Wang).

https://doi.org/10.1016/j.comnet.2020.107706
Received 14 July 2020; Received in revised form 30 October 2020; Accepted 20 November 2020
Available online 23 November 2020

A hybrid SDN is comprised of both legacy and SDN devices [6]. The legacy network devices are running the legacy network protocols, and the SDN devices run SDN protocols. Due to its unique nature, the hybrid SDN approach offers several challenges, in terms of traffic engineering (TE), loading balancing (LB), energy-efficient routing, SDN nodes deployment, link failure management, and implementation of access control list (ACL) policies [7–16].

ACL is a filter mechanism deployed at switches' (or routers') interfaces and end-system firewalls in order to enforce security. ACL filters the data flows based on predefined rules. One or more predefined ACL rules define the flow matching provisions based on some packet header values (e.g., source/destination IP addresses, transport layer port number, and transport protocol value). Based on the implemented ACL policy, a switch discards or forwards the flow. Moreover, ACL policies enable filtering of unauthorized flows early and adequately protecting the core services. Efficient implementation of ACL policies at switches' interfaces also helps increase network throughput, reduce end-to-end delay for data flows, and effectively distribute the flows at the edge and backbone network, contrary to other ACL (i.e., end-system firewalls) [17].

Existing approaches have addressed different challenges by proposing various alternative solutions to implement the ACL policies in hybrid SDN efficiently and correctly. Some of these challenges are summarized next.

i. *ACL policies disclosure:* In a SDN architecture, policies can be declared either by general-purpose or domain-specific languages, e.g., Frenetic [18] and Pyretic [19]. These languages allow operators of diverse types, i.e., parallel and sequential, to make ACL policies. A parallel operator checks the same set of packets against multiple ACL policies simultaneously. In contrast, a sequential operator reviews one ACL policy at a time.
ii. *Overlapping and conflict in ACL policies:* At the controller, declared ACL policies in a network may lead to unforeseen ACL policy violations [20,21], network reachability problem, packet loss [22,23], and overlaying and conflict in ACL policies [24] due to different network administrations or even under the same administrator over time.
iii. *Storing ACL policies at controller:* ACL policies can be held at the controller using various methods, like PGA [21]. PGA uses a graph to keep network ACL policies. When a packet arrives, PGA traverses through the graph and generates flow rules from the predefined ACL rules. Additionally, PGA detects redundancy and identifies conflicts among the ACL policies declared at the SDN controller.
iv. *Where to deploy?:* To minimize the unwanted traffic in the network, the ACL policies can be installed as flow rules at the different sets of switches using various criteria. Rashid et al. [13,14] implement the ACL policies at optimum locations (i.e., switches' interfaces) in the network such that it minimizes the total number of both implemented ACL policies and unwanted transmissions.
v. *Proactive or reactive ACL policies installation:* The controller can install the ACL policies in either a proactive or reactive manner [25]. In proactive manner, the controller installs the ACL policies for all users, active or passive, leading to the Ternary Content Addressable Memory (TCAM) memory problem. In contrast, in a reactive way, the controller installs the ACL policies only for active users. Therefore, the reactive approach uses more efficiently the limited TCAM memory at the switches.

We believe that the existing approaches still have limitations. For example, Rashid et al. [13] state that when a link fails, the SDN controller recomputes the locations for the implementation of ACL policies placements to ensure the network behaves correctly. They have assumed that the link failure was received in real-time to the controller in hybrid SDN, which is often not the case in such settings when legacy link failures are considered. The links connected to legacy switches are called legacy links, while the links connected to SDN switches are called SDN links. SDN controller directly gets the real-time link-state information from SDN switches. However, using a legacy routing protocol like OSPF [1,6], the legacy switches periodically broadcast their link-state information to all the nodes, including the SDN switches in the network.

When an SDN switch receives the link-state information of legacy switches, it sends them to the SDN controller. Thus, it takes a long time to send legacy links information to the SDN controller [4,6,7,26] and the legacy link failure information may not be received at the SDN controller in real-time. When a legacy link, which is far away from a SDN switch, fails and the link failure causes incorrect network behavior (i.e., either a violation of ACL policies by the network traffic or a network reachability problem due to implemented ACL policies on devices' interfaces) then the network behavior will persist for a longer time until the SDN controller receives the link failure notification and, subsequently, recomputes and implements the ACL policies according to the updated network topology. We attempt to address this problem by predicting the link failure before its occurrence, computing and implementing ACL policies, link failures.

To the best of the authors' knowledge, the state-of-the-art literature for hybrid SDN (e.g., [13,14,21,27,28]) overlooks the ACL policy configuration in advance for the case of legacy link failures. This paper proposes the use of Machine Learning (ML) algorithms to predict link failure and, subsequently recompute the ACL policy configuration considering the link failure. More particularly, we propose the use of two ML algorithms (i) Logistic Regression (LR) [29], (ii) Support Vector Machine (SVM) [30] to enable high resiliency to link failure and reduce the number of ACL policy violations. Recent studies show that ML techniques are resilient, flexible, and applicable in the communication network. They can be employed to manage critical issues such as fault management, traffic predication, traffic routing, and network security [31–34].

The main contributions of this paper are summarized as follows.

- In a hybrid SDN, the SDN controller receives legacy links failure information with delay, creating inconsistency at the SDN controller. The inconsistency leads to network reachability and ACL policy violation problems. ***PrePass-Flow* is proposed to address these problems**.
- In order to minimize the impact of a link failure in hybrid SDN, PrePass-Flow uses historical link information and predicts the potential link failure. **The prediction module in PrePass-Flow employs two supervised ML algorithms**, called Logistic Regression (LR) and Support Vector Machine (SVM).
- Following failure prediction, PrePass-Flow **recomputes the optimum locations for ACL policies** considering the link failure and installs the ACL policies such that the network behaves correctly as the link fails.

The remaining paper is organized as follows. The related work is reviewed in Section 2. We present the problem definition and challenges in Section 3 and our proposed PrePass-Flow in Section 4. Section 5 describes the performance evaluation of the proposed PrePass-Flow solution and conclusions are drawn in Section 6.

2. Related work

This section classifies the related work into three categories. The first category in Section 2.1 describes the current literature related to the network-layer failure problem. The second category in Section 2.2 includes recent studies about ACL policy violation in a hybrid SDN architecture. The third category in Section 2.3 discusses some applications of ML-based algorithms for improving network performance.

2.1. Approaches related to network-layer failures

Recent studies reveal that network-layer failures, e.g., a switch failure or a link failure, frequently happen and show that such outages are disruptive [7–9,35–42]. Furthermore, link failures can last for about 30 min [43] and the average convergence times to recover from the link failure is normally more than 30 s [35,44]. The root causes of a link failure can be device deployment (change), OS reboot (incident), OSPF convergence (network connection), software (BIOS upgrade), hardware (power supply or line card replacement), and configuration (VPN tunneling or IP/MPLS routing) [38,45]. In [35,44], the authors proposed mechanisms to reduce the convergence time after outage in the network. V. Liu et al. also explain that switch or link failures impact 40% traffic in a network, except when the underlying network is designed to be resilient to such failures [41].

In modern heterogeneous network architectures, ML-based models can play an essential role in managing the network faults, including link failure [31]. Zhilong et al. [33] proposed a ML-based model to predict device failures in a SDN-based optical network. The researchers combined Double Exponential Smoothing (DES) [46] and SVM [30] models to forecast the risk of a device failure proactively. Moreover, studies show that almost 62% of network failures occur due to human error and maintenance such as when a network admin manually configures the legacy switches in a hybrid SDN. Additionally, about 80% of an Information Technology (IT) budget is required for network operation and management [24,47].

In a hybrid SDN, configuring the ACL policies on both SDN and legacy switches from the logically centralized SDN controller at runtime are now possible because the SDN controller has an abstract network view [48]. Furthermore, failure of a switch or link can cause network wide negative effects in terms of ACL policies violations and network reachability problems [13,14,49,50]. The research community [13,14,17] suggested new methods to minimize ACL policy violations in case of network faults in a network. However, in the case of hybrid SDN, it is still an open question to predict failures of legacy links, and recompute the location and reconfigure the ACL policies before a link fails. The current studies, e.g., [13] recompute the location of ACL policies and reconfigure the ACL policies after link failure and its detection at the SDN controller. Unfortunately, these current solutions may cause network congestions, ACL policy violations, packet loss, and network reachability problems due to ACL policies. In this work, the PrePass-Flow uses ML-based models to predict network-layer failure, compute the alternative ACL policies location proactively, and implement the ACL policy on the alternative locations (i.e., switch interfaces) before link failure occurs. This proactive way reduces both the reachability problem due to ACL policies and ACL policy violations in case of link failure.

2.2. Approaches related to ACL policy violation

In a communication network, ACL policies play a significant role in providing security, network reachability (connectivity), and reliability. Network devices such as routers or switches are configured with ACL policies. The primary purpose of ACL is to filter the unwanted traffic as early as possible and increase network performance (reduce bandwidth consumption and increase throughput [14,51]. Currently, ACL policies are widely installed in the network. However, the network size, complexity, and end-users' demands are growing day by day. Thus, maintaining ACL policies across the communication network such that the network runs correctly and efficiently is a challenging and complicated task for network operators [51]. Even a single ACL policy configuration may lead to network reachability problems due to ACL policies, service disruptions, and violations of ACL policies. In [17], the authors proposed a system that automatically updates ACL configurations according to network operator intent. The proposed system, based on a Language for ACL Intents (LAI) and a LAI model, automatically generates an ACL policies plan according to the requirements. For performance evaluation, the LAI model has been deployed in Abibab's global WAN. SecGuru [52] also proposed a model to confirm the accuracy of the ACL policies declared at the firewall in the network. However, these models do not consider the accuracy of the ACL policies distributively implemented at many switches' interfaces in the network.

It is a tedious task for the network operator to manually configure, manage, and monitor the ACL policies in all network devices (like switches'/routers' interfaces) in case a new link is added and/or a link fails. Such a case may lead to packets violating the ACL policies and to network reachability problems due to ACL policy. For example, in case of a topology change following link failure, an AutoConfig Model [13] was proposed to detect the forwarding device interfaces where the implemented ACL policies can cause ACL policies violations and network reachability issues. The proposed model uses a graph difference technique [53] to detect changes in the network topology. It is an NP problem to compute the locations (i.e, the switches'/routers' interfaces) for implementing the ACL policies in the network based on some performance parameters [51], such as for example, to minimize the amount of unwanted traffic generated in the network [14]. To solve this optimization problem, Rashid et al. used a decision-tree and K-partite graph for computing the optimum locations for implementing the ACL policies in a hybrid SDN architecture [14]. In [54], the authors proposed a model to minimize the inconsistency behavior between the already installed flow rules in SDN switches and new rules using a hybrid mechanism which is a combination of reactive and proactive mechanisms.

Automatic Test Packet Generation (ATPG) [55] is a debugging tool used to find the network wide issues, including ACL policies violations. ATPG generates testing packets and traverses through the network configuration. The authors of [56,57] proposed a model to detect conflict among the ACL policies before and after a network administrator updates the ACL policies at the controller, by using a multi-attribute graph matching algorithm [53]. If conflict is found in both policies, the controller removes those flow rules installed in the switches that violate the updated ACL policies. Their proposed model minimizes the number of packets violating ACL policies. PGA [21] declares the ACL policies at high abstract level and detects conflicts among the ACL policies declared by different network operators and at different times at the controller. Object Oriented SDN Framework (SDNSOC) [58] uses the object-oriented concept of the programming language to detect the conflicts among the flow rules generated by the controller as per ACL policies and network topology.

From the above discussion it is clear that link failures can affect the correctness and efficiency of ACL policies' implementation in hybrid SDN. Moreover, none of the existing approaches predict link failure in order to recompute the locations of ACL policies and then reconfigure the ACL policies at the new locations such that the network operates correctly and efficiently as intended by the network operators through the ACL policies.

2.3. Applications of ML-based prediction algorithms in networking

A ML algorithm receives data from the network, learns the pattern, and decides action based on the received data. In general, a ML algorithm includes the following steps (i) a preprocessing phase, in which data is filtered (ii) a training phase, in which the algorithm learns how to make decisions based on input data and (iii) a testing phase which verifies the correctness of algorithm decisions. ML algorithms can be based on unsupervised or supervised learning models. Unsupervised learning models use unlabeled training data, whereas supervised learning models employ labeled data for training purposes [59]. In a previous work [16], we have used the K-Nearest Neighbor Regression algorithm [60] to predict the reliability level of the legacy links. After the prediction phase, an unsupervised ML learning algorithm, called Reliable and Time-sensitive Deep Deterministic Policy Gradient

algorithm (RT-DDPG), was proposed to compute the path based on constraints, e.g., bandwidth, delay, and reliability. For a hybrid SDN, Tracy et al. [6] proposed to use a Linear Regression (LR) algorithm at the SDN controller to predict the current traffic load at legacy links by using both historical traffic load of legacy links and current load of the SDN links.

In the routing optimization process, traffic prediction in a network is vital and is an important research topic [61–65]. The main objective of traffic prediction models is to predict the traffic intensity from historical statistics. After the prediction, the SDN controller reroutes the traffic proactively or reactively on alternative paths. These proposed models are used in SDN to predict different QoS parameters, e.g., traffic, routing and network delay. Unlike them, in this work we consider a hybrid SDN architecture and predict link failures to minimize the network reachability and ACL policies' violation problems.

Alessandro et al. introduced a ML-based model called *Framework for building a failure prediction model* to predict abnormal conditions that are supposed to happen [66]. In the proposed model the Remaining Time to Failure (RTTF) of applications is predicted using various ML algorithms. Additionally, the proposed model allows the user to customize it according to the requirements. Other performance models are also used to predict system performance in terms of resources consumed and workload design [67–69]. These performance models can be divided into two categories: (a) white-box performance models, also called analytical models, and (b) black-box performance models [67]. White-box performance models use system contexts, e.g., service request and workload intensity, to predict the system's performance. In contrast, black-box performance models take the system as a black-box and use ML or statistical models to predict a system's performance. Black-box models generally require no knowledge about the system's internal behavior and can predict the system performance while the system is under field operation [68]. Didona et al. [67] predicted the performance in two case studies: a NoSQL datastore and a Total Order Broadcast (ToB) service. The proposed prediction model relies on analytical modeling and ML algorithms. Similar to this model, the author proposed a model to predict the accuracy of the system based workload and configuration parameters rather than hardware data, e.g., network and CUP [69]. Sanzo et al. proposed a ML-based prediction model to self-tune the running configuration of applications in single and multiple cloud regions [70–72]. However, the proposed models' primary intention is to increase performance rather than dependability and availability. The proposed work [66–70] is based on ML's particular usage since they provide performance prediction tools that cannot support what-if analysis in the wide — for example, examining the consequence of notable workload shifts outside the workload domain used during the ML training. Therefore, the proposed prediction tools stay bound to a specific scenario. Unlike these prediction models, our proposed PrePass-Flow approach predicts the network failure and proactively takes the necessary action to minimize the impact of network-layer failure, regardless of the scenario.

3. Problem definition and challenges

In a communication network, network-layer failures, including link failures, happen frequently [7–9,35,37–40,42] and have a considerable impact on the overall network performance [41,50]. The network-layer failures affect specifically network reachability and increase the end-to-end delay. Furthermore, the network performance in terms of ACL policies [13–15,17] is also severely affected. For example, if a link is down or its status is changed, the data flows passing through the failed/changed link may violate the ACL policies (which, in turn, allows the network traffic to pass to an unlawful sub-network) and may result in dropping data flows (which causes the network reachability problem), as discussed in [13,16,73]. To better illustrate this point, we discuss a case study based on a real-life campus network next.

3.1. Case study: ACL policy violation and network reachability problem in a campus network

Suppose we have a campus network, as shown in Fig. 1(a). It is a hybrid SDN because only R5 is a SDN switch and the rest are legacy switches. We can see that the campus network consists of four sub-networks (or subnets). Additionally, subnet-1 and subnet-4 only consist of servers, where subnet-1 provides its services to subnet-2, and subnet-4 provides its services to subnet-3. We assume the following network scenario.

An administrator of the campus network implemented an ACL policy that subnet-1 (servers) can only be accessed from subnet-2, while subnet-4 can only be accessed from subnet-3. Also, subnet-2 and subnet-3 can communicate with each other. That is, subnet-2 cannot access subnet-4, and subnet-3 data traffic is not allowed to access servers of subnet-1. Particularly, interfaces i1.1 and i1.2 of switch R1 are configured with ACL policies to drop all flows originated from subnet-3. Furthermore, interfaces i5.1 and i5.3 of R5 are configured to drop all flows originated from subnet-2, as shown in Fig. 1(a). Additionally, other subnets can send traffic to each other.

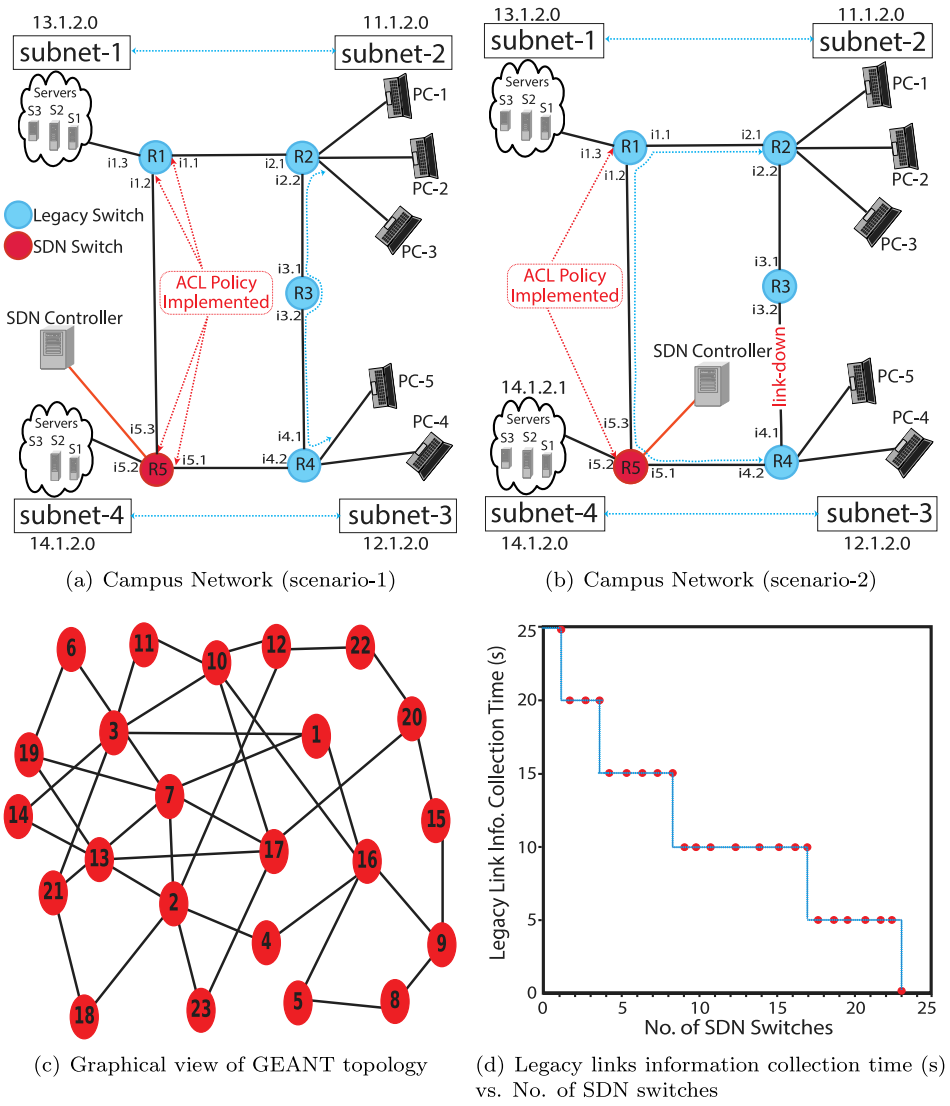
Now let us consider the path R4 to R2 via R3, carrying tens of thousands of flows from subnet-3 to subnet-2 and vice versa. Suppose the link (R4, R3) suddenly fails, as shown in Fig. 1(b). After the link is down, the routing protocol will find an alternative path (i.e., R2–R1–R5–R4) to send the data from subnet-2 to subnet-3. But, when the data from subnet-2 arrives at the interface i5.3 of R5, the data will be discarded due to the ACL policy (i.e., the administrator has imposed the ACL policy to discard the data originated from subnet-2, as already explained). Similarly, R1 will drop the data flows originated from subnet-3 and destined to subnet-2 following the path R4–R5–R1–R2. This causes the problem of network reachability. The physical topology has the path from subnet2 to subnet-3, but the subnet-2 and subnet-3 cannot communicate due to the ACL policy. It is a challenging task to detect such a network reachability due to ACL policy in a large network and may be unnoticed for a long time. Similarly, the ACL policy violation can also occur due to link failure as discussed in [13,73]

3.2. Limitations in current state-of-the-art

To minimize both ACL policy violation by packets and network reachability problem due to ACL policies, a mechanism is proposed in [13], based on recomputing the locations of ACL policies once the legacy link failure has occurred. The authors assume that the legacy link failure notification is received in real-time (i.e., instantly) at the SDN controller. In a hybrid SDN, the SDN controller can detect SDN link failure timely; however, it takes considerable time to detect legacy link failures at the SDN controller depending on the distance between the legacy link and the closest SDN switches. This is as the legacy switches use legacy protocols (like OSPF [6]), and they share their link state with the SDN switches using such legacy protocols. Then the SDN switches pass on this information to the SDN controller, as explained in [1,6,16]. This involves a long time, and so it takes to detect legacy link failure at the SDN controller.

For experimental validation of this concept, we have employed the publicly available GEANT topology[1] [74] with available traffic matrices to monitor the legacy links information concerning the incremental deployment of SDN switches, as shown in Fig. 1(d). We use the Mininet[2] network emulator with the POX controller[3] in this experiment. The topology consists of 74 links and 23 legacy switches. We have randomly selected a legacy switch in the GEANT topology and have observed how

[1] http://www.geant.net/upload/pdf/Topology_Oct_2004.pdf.
[2] http://mininet.org.
[3] http://github.com/noxrepo/pox.



(a) Campus Network (scenario-1)

(b) Campus Network (scenario-2)

(c) Graphical view of GEANT topology

(d) Legacy links information collection time (s) vs. No. of SDN switches

Fig. 1. A campus network (a) and (b) contains servers and users. (c) shows the GEANT topology. (d) shows legacy links information collection time (s) vs. No. of SDN switches.

long it takes to obtain the information of all its links at a node (because we can connect the SDN switch with the SDN controller through an out-band communication link [75]). Each legacy switch operates using the OSPF protocol and periodically broadcasts its link-state information to neighbors every 5 s. By incrementally replacing legacy switches with SDN switches, the legacy links' information collection time at the SDN controller decreases as the number of SDN switches increases, as shown in Fig. 1(d). Furthermore, we have observed that the controller does not have any update information about the network-layer failures, particularly legacy switches and legacy links. Thus, in a hybrid SDN, it takes a significant time to detect a far away legacy link failure at the SDN controller, to recompute the locations of ACL policies, and then to configure/implement the ACL policies at the newly computed locations. This confirms it will take significant time for a new ACL policy to be effective in case of a far away legacy link failure in hybrid SDN. This creates two basic problems.

1. **Network reachability problem**: packets will continue to be dropped until the legacy link state information is received at the SDN controller, the controller recomputes the locations of ACL policies and the ACL policies are implemented/installed at the new locations, as already discussed (and shown in Figs. 1(a) and 1(b)).
2. **ACL policy violation problem**: the controller will violate the ACL policies until the legacy link state information is received at the SDN controller, the controller recomputes the locations of ACL policies and the ACL policies are implemented/installed at the new locations, as already discussed (and reported in [13, 73]).

The current literature does not address these two problems associated with link failures [13,14,21,27,28]. Therefore, the main focus of our proposed approach PrePass-Flow is to address these two problems by predicting legacy link failures at the SDN controller before they occur. After this failure prediction, the SDN controller recomputes the locations of ACL policies and then installs them in advance at the new locations, saving valuable time and reducing the negative effects of link failures.

4. PrePass-Flow

This section presents and discusses the proposed technique, *PrePass-Flow*, which is used to reduce the effects of ACL policy violations and network reachability problems (as discussed in Section 3) due to link failures in a hybrid SDN. PrePass-Flow employs a ML algorithm and predicts the legacy links' status (i.e., fail or functional) from historical data.

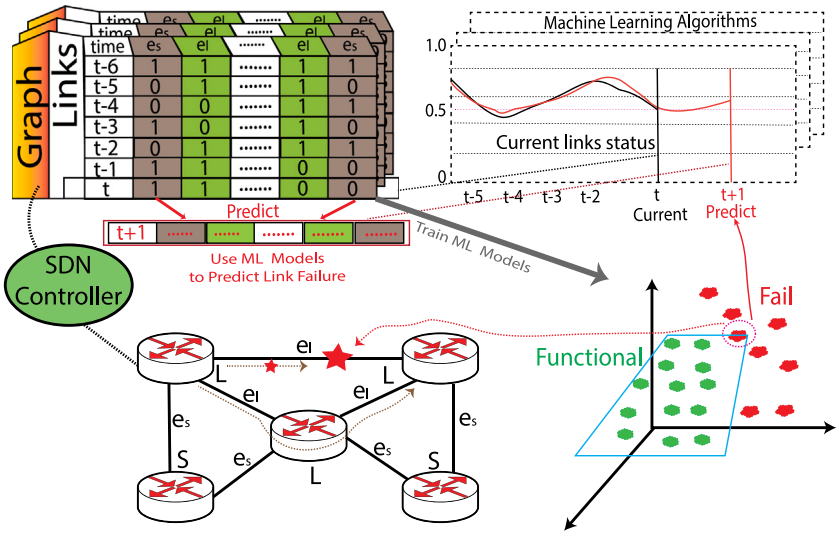


Fig. 2. ML based link's failure prediction model and reroute the flow.

Specifically, PrePass-Flow deploys ML-based prediction models in the SDN controller, as shown in Fig. 2. The controller systematically communicates with the SDN switches using the OpenFlow protocol, and the SDN switches interact with the legacy switches to receive Link State Parameters (LSP) including link failure information about legacy links using a traditional (i.e., legacy) protocol (e.g., OSPF protocol [6]). The SDN switches then pass on LSPs received from the legacy switches to the SDN controller. The controller acquires the status of the legacy links from the LSPs forwarded by the SDN switches. The ML algorithms are trained with the received links' information. When a failure is predicted, the PrePass-Flow initiates protection measures by recomputing the locations of ACL policies and implementing the ACL policies at the recomputed locations (if necessary) to avoid ACL policies violations by the packets and flow disturbances. Furthermore, the controller sends control messages to SDN switches about the detour flows.

The proposed prediction model is shown in Fig. 2. The figure shows how the SDN controller obtained historical link status information from time $t-6$ to t, inputs it into the prediction module $P(.)$ and the model predicts the link failure status at time $t+1$ (i.e., fail = 0, functional = 1). If the prediction module $P(.)$ predicts the potential link failure, it initiates a proactive protection measure to reduce ACL policy violations and network reachability issues. PrePass-Flow uses the panopticon [48] model for the SDN switches deployment. In the panopticon model, the authors deploy the SDN switches among the legacy switches so that every flow between source and destination pair traverses through a SDN switch. Therefore, a single SDN switch is sufficient to enforce the end-to-end ACL policy. Consequently, the proposed model enables customized flow forwarding if a flow passed through multiple SDN switches. For computing the optimum locations of ACL policies in a hybrid SDN, we use the AutoConfig [13,14] model. The details of PrePass-Flow's components are given in the following subsections.

4.1. Network model

Consider that a hybrid SDN network, $G = (N, E)$, consists of a set of switches N and a set of interconnected links E. Suppose in G, $S \subset N$ indicates a set of SDN switches, and $L = N \setminus S$ is the set of legacy switches. In PrePass-Flow, each link $e \in E$ is associated with the link prediction $P(e)$ module. This prediction module predicts the status s of the link being either functional or down (e.g., $s = 0$, or $s = 1$).

G consists of a set of paths $X = \{x_z\}$, where a path x_z may be multi-hop and can be represented as follows:

$$x_z = (e_s, e_l, \dots, e_{(|x_z|)}), \tag{1}$$

where are the SDN links and e_l are the legacy links in the path x_z, as shown in Eq. (1). The prediction module $P(.)$ assigns a state to each link $e \in E$, i.e., functional = 1 or fail = 0 based on attributes. The attributes can be link change, software or hardware upgrading, and configuration, as shown in Table 1. These attributes refer to external conditions $k \subset K$, which influence the link state at the given time slot $t \in T$, as shown in Eq. (2),

$$P_t(e|K) = 1 - \sum_{k \in K} (1 - P(e|k)), \tag{2}$$

$$s.t., \quad P_t(e|K) : k_e \rightarrow [0, 1], \forall e \in E,$$

where $P_t(e|k) : k_e \rightarrow [0, 1]$ indicates the existing state of a link at time t. Eq. (2) shows the existence prediction of a link e in a set external conditions $k \subset K$.

As specified in Section 3, in a hybrid SDN, the SDN controller does not have the current link failure information of legacy links $e_l \in E$ in a given time $T^* \subseteq T$. $P(.)$ computes the status of all links as follows:

$$P_{e \in E}(K_G/T^*) = \left[\underbrace{\sum_{e_l \in E_{G_L}} (P_{e_l}(K_{e_l}/T^+)) +}_{\text{(b) Legacy links}} \underbrace{\sum_{e_s \in E_{G_S}} P_{e_s}(K_{e_s}/T^*)}_{\text{(a) SDN links}} \right], \textit{where } T^+ > T^*, \tag{3}$$

$$G_L = (N_{G_L}, E_{G_L}), \; G_S = (N_{G_S}, E_{G_S}), \; (G_L \cup G_S) \subseteq G,$$

$$(E_{G_L}, E_{G_S}) \subseteq E, \; T^+ > T^*, \; \forall e \in E,$$

where Eq. (3) shows that $P(.)$ computes the status of $e_s \in E_{G_S}$ directly (i.e., in time T^*). However the controller does not have any updated link failure information about the $e_l \in E_{G_L}$ in time T^*. This problem creates inconsistency/uncertainty at the controller. The controller may forward the flows on a broken link or may violate ACL rules during the flow installation process or may have a network reachability problem due to ACL policies. PrePass-Flow's goal is to reduce the violation of ACL policies by the packets and network reachability problems due to ACL policies during network-layer failures. Consequently, in a hybrid SDN, PrePass-Flow uses ML models to predict the status of $e_l \in E$ and $e_s \in E$ in advance.

4.2. ML models for link failure prediction

In order to evaluate the performance of PrePass-Flow, we have used the realistic parameters of network-layer failure mentioned in Table 1.

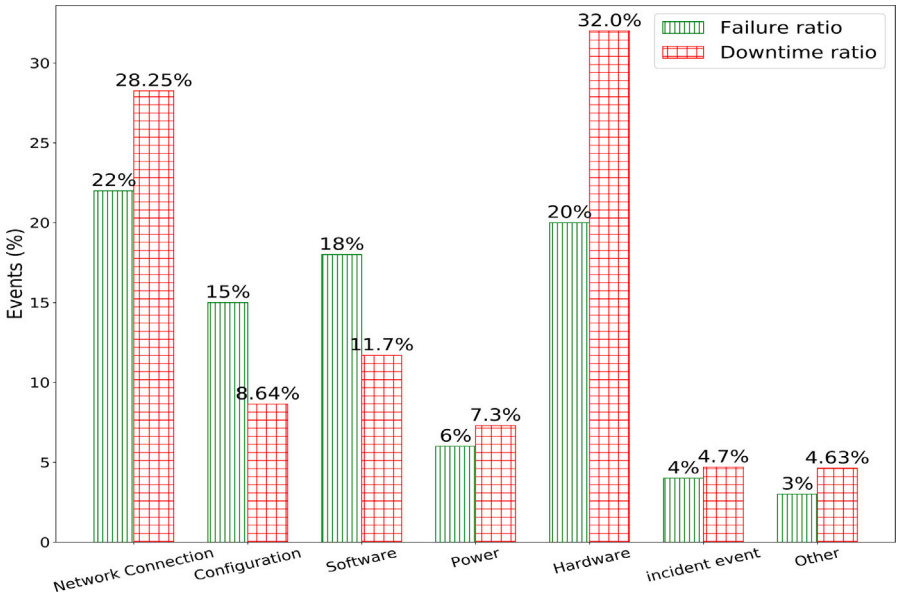


Fig. 3. Total failure ratio vs. total downtime ratio to the major causes.

Table 1
Major causes of network-layer failure.

Failure	Cause of failure
Network connection	OSPF convergence
Configuration	Modifying primary-backup routing
Software	IOS upgrade
Power	Power/UPS failure
Hardware	Replacement of power supply/line card
Incident event	OS reboot
Other	Unreported or DoS attack

Moreover, Fig. 3 shows the impact of major causes of link failure, and compares the failure ratio and downtime ratio. In Fig. 3 the sum of values is less than 100% because it includes only cleaned syslog data, following filtering out data of any transient events or multiple entries. For instance, multiple *down* messages generated for the same link within a close time interval are grouped, and the earliest start and end times for the failure is saved only. The filtered out data accounts for the remaining percentage to 100%. We can see from Fig. 3 that the network connection, hardware, software, and configuration issues significantly impact on the link failure. The link downtime ratio is more in case of hardware issues followed by network connection and software. Other causes affect less the downtime ratio, which means links are down for a short time. However, link failure for a short time also creates problems in a hybrid SDN network. PrePass-Flow considers all types of link failures, e.g., for a short or long duration, and predicts link's status to minimize the link failure's impact on the network reachability and ACL policy violation.

PrePass-Flow assumes the history of any e_i link represented by as 5-tuple: $e_i = (Day_{time}, link_{downtimestart}, link_{uptime}, link_{downfrequency}, linkfailure_{reason})$. $link_{downtimestart}$ represents the occurrence time of link failure; $link_{uptime}$ is the time when the link has recovered or has become operational again; $link_{downfrequency}$ measures how many times the specific link was down, and $linkfailure_{reason}$ indicates the link failure reason (i.e., power, configuration, hardware, and software), as mentioned in Table 1. The total down duration of a link is $link_{downtime} = link_{uptime} - link_{downtimestart}$.

In PrePass-Flow, this 5-tuple is chosen as an input or independent variable to the ML models which predict the status (output or dependent variable) of the link. For example, when a new link-down event occurs, $e_x = (Day_{time}, link_{downtimestart}, link_{uptime}, link_{downfrequency}, linkfailure_{reason})$, based on the historical statistics of link e_x, the SDN controller uses the ML models to predict the status s of the link e_x. Let $Z = \{(e_1, s_1), (e_2, s_2), \ldots, (e_n, s_n)\} \in (e, s)^n$ be the data on n (5-tuple) independent variables, where the random pairs (e_i, s_i), e_i represent an instance in a D-dimensional feature space of the link like $e_i = [e_{i1}, e_{i2}, \ldots, e_{iD}]$ and s_i is the dependent variable/output of each associated link e_i. The objective of ML models is to predict the status s for a new link-down events $e_x = [e_{x1}, e_{x2}, \ldots, e_{xn}]$ using learn function $P : e \rightarrow s$. In PrePass-Flow, we suggest two different algorithms of binary classifications of ML, i.e., the Logistic Regression (LR) and Support Vector Machine (SVM) models, to classify the link's status, (i.e, functional s = 1 or down s = 0), as described in the following subsections.

4.2.1. Logistic regression model

Logistic Regression (LR) [29] is a statistical model, which is one of the most commonly used supervised ML-based models for two-class (binary) classifications. In PrePass-Flow, we use LR model as the baseline for binary classification problem, i.e., link e is either functional ($s = 1$) or fail ($s = 0$). Furthermore, an LR model predicates and describes the correlation between one output/dependent binary variable (e.g., 0 or 1) and independent/input variables. In PrePass-Flow, 0 and 1 represent the status s of the link e. LR model uses *logit* function to predict the occurrence probability of a binary event:

$$logit = \frac{p(e)}{1 - p(e)}, \tag{4}$$

where *logit* shows the ratio of the probability that a link failure event occurs to the possibility that it does not happen. Suppose the condition of the probability $p(s = 1|e = p(e))$, where 1 represents the class. For this, PrePass-Flow uses a logistic or sigmoid function, as shown in Eq. (5a), where Eq. (5b) shows the sigmoid function curve equation.

$$p(e) = \frac{1}{(1 + exp^{-\alpha e})}, \tag{5a}$$

$$\alpha e = log\frac{p(e)}{(1 - p(e))}. \tag{5b}$$

Parameter prediction in LR model: In PrePass-Flow, the goal of learning in the LR model is basically to predict the occurrence probability of a link failure event $p(e)$. In the model, the equation parameters in the LR two-class classification are provided as $\overbrace{\alpha}$ vector, as a *logit* equation. For parameter prediction, LR uses the *Maximum Likelihood*

method. Using *Maximum Likelihood* method, we label the data-set M samples either 0 or 1.

- Samples with label 1: the goal of the LR model is to predict $\overbrace{\alpha}$ such that $\overbrace{p(e)}$ is as close as possible to *1*.
- Samples with label 0: the LR model should predict $\overbrace{\alpha}$ such that $\overbrace{p(e)}$ is as close as possible to *0* or predicts $\overbrace{\alpha}$ such that $1 - \overbrace{p(e)}$ is as close as possible to *1*.

Mathematically, for every sample M with label 1, LR estimates $\overbrace{\alpha}$, such that the product of all class 1 samples are supposed to be as close to 1 as possible, as given below.

$$\prod_{s \in s_i = 1} p(e_i). \tag{6}$$

Similarly, for the group sample with 0, the LR predicts the $\overbrace{\alpha}$, such that the product of complement conditional probability is supposed to be as close to 1 as possible, as given below,

$$\prod_{s \in s_i = 0} (1 - p(e_i)), \tag{7}$$

where Eq. (7) uses the maximum value of $\overbrace{\alpha}$, and s_i represents the status of link e_i.

Combining these requirements, in the proposed PrePass-Flow, we define the data parameters such that the product of both these product s is maximum over all elements of the data-set, as shown in Eq. (8):

$$MLF(\alpha) = \prod_{s \in s_i = 1} p(e_i) \times \prod_{s \in s_i = 0} (1 - p(e_i)), \tag{8}$$

where function in Eq. (8) is called *Maximum Likelihood Function (MLF)* and here we need to optimize the MLF, as shown in Eq. (9) :

$$\alpha = \arg\max_{\alpha} \quad MLF(\alpha), \tag{9a}$$

$$MLF(\alpha) = \sum_{i=1}^{m} s_i \alpha e_i - log(1 + exp^{\alpha e_i}). \tag{9b}$$

Note that the main objective of Eq. (9a) is to find the value of α to maximize MLF and Eq. (9b) shows the simplest form of MLF that needs to be optimized. Consider Eq. (9b), which contains log and exponential components; such transcendental equations cannot be computed precisely; however, a numerical method can be used for finding an approximation solution. Therefore, in the proposed PrePass-Flow, we use the *Newton Raphsin Approach (NRA)* to find a good approximation quickly, as shown in Eq. (10a).

$$\nabla_{\alpha} MLF(\alpha) = \nabla_{\alpha} MLF(\alpha^{\dagger}) + \left(\alpha - \alpha^{\dagger}\right) \nabla_{\alpha\alpha} MLF(\alpha^{\dagger}), \tag{10a}$$

$$\alpha^{t+1} = \alpha^{t} - \frac{\nabla_{\alpha} MLF(\alpha^{t})}{\nabla_{\alpha\alpha} MLF(\alpha^{t})}. \tag{10b}$$

where, Eq. (10b) represents the *t-iteration* of NRA and it is also called *Newton Raphsin Equation (NRE)*. After *t-iteration*, then α will eventually converge into an approximate coefficient vector. The NRE (see Eq. (10b)) involves computing gradients *w.r.t* ∇_{α}. For this, let determine the gradients, as shown in Eq. (11):

$$\nabla_{\alpha} MLF = \nabla_{\alpha} \sum_{i=1}^{m} s_i \alpha e_i - log\left(1 + exp^{\alpha e_i}\right), \tag{11}$$

where we compute the MLF gradients. To compute the denominator terms of NRE (see Eq. (10b)), which is also called *Hessian Matrix*, we convert the gradient vector into a matrix representation, as shown in Eqs. (12) and (13):

$$\nabla_{\alpha} MLF = E^{T}(s - p). \tag{12}$$

$$\nabla_{\alpha\alpha} MLF = -E^{T} p_i (1 - p_i) E. \tag{13a}$$

$$\nabla_{\alpha\alpha} MLF = -E^{T} Y E. \tag{13b}$$

where Eqs. (12) represents the matrix notation. Additionally, we also represent Eq. (13a) in a matrix Y notation, where Y is a diagonal $n \times n$ matrix and the status of the ith link in the matrix is equal to $p_i(1 - p_i)$, as shown in Eq. (13b).

Algorithm 1: PrePass-Flow Logistic Regression

Input: Matrix E contains M samples, status s_i, threshold value tv, $max_{iter} = n$
Output: status s_i prediction of a link e_i

```
// constant complexity time O(c)
Function PrePass_Flow_LR(E, s_i, tv, n)
    // initial value for α and difference
    α = rep(0, ncol(E))    // O(c)
    difference_value = DV    // O(c)
    // iterations-counter
    iterations-counter = 0    // O(c)
    while (DV > tv) do    // O(n)
        p = vector(Function(E, α))    // O(n^2 + c)
        // compute diagonal matrix of weights Y
        Y = diagonal_mat(p(1-p))    // O(n^2 + c)
        // α value at t
        α(t) = compute (t(E)YE)) t(E)(s-p)    // O(n^2 + c)
        // α updation
        α = α + α(t)    // O(n + c)
        DV = sum(α(t)^2)    // O(n + c)
        iterations-counter+=1
        if (iterations-counter > n ) then stop    // O(n)
        return
    Function (E, α)
        α = vector(α)    // O(c)
        return exp^{E·α} / (1+exp^{E·α})    // O(c)
return α
```

After obtaining the matrix representations, we put Eqs. (12) and (13) in the *Newton Raphsin Equation (NRE)* (see Eq. (10b)) to obtain the final optimization form, as shown in Eq. (14):

$$\alpha^{t+1} = \alpha^{t} + \left(E^{T} Y^{t} E\right)^{-1} E^{T}(s - p), \tag{14}$$

where we execute Eq. (14) t times iteratively (Newton update) to predict the status of s. In each iteration p and Y will be updated, as we recompute it by using an updated α. After t iterations, we can get the full converged coefficient value also called maximum likelihood estimator α, as shown in Algorithm 1.

In Algorithm 1, we set the initial value of α for quicker convergence and threshold value tv. The threshold value uses to check whether the PrePass-Flow Logistic Regression (LR) Algorithm is converging or not. After t iterations, the value of α drops below the threshold value TV because it will become smaller and smaller in each iteration. In Algorithm 1, the α in each iteration moves closer to its highest likelihood value. In case the value of α does not drop below the threshold value, then the algorithm terminates as it does not converge. To handle this problem, we initialize a counter to count the iterations and terminate the algorithm. Once the coefficient is predicted, then we put the values in Eq. (5), to obtain the status of a link: either 0 or 1. For optimization process, we use the following filter mechanism:

$$p(e_i) = \begin{cases} s_i = 1 & \iff \quad p(e) > threshold\ value \\ s_i = 0 & \iff \quad p(e) \leq threshold\ value \end{cases},$$

to chose the threshold. Here a value above the threshold means the link is functional and a below threshold value means the link is down.

4.2.2. Support vector machine

Another supervised ML approach, called Support Vector Machine (SVM), used for binary classification [30] is employed in PrePass-Flow.

We use the historical data of link failure to train the support vectors of SVM to make intelligent predictions about the link status (e.g., either functional or down). To differentiate between the status of the link, we used a *Maximal Margin Classifier (MMC)* in the SVM model. MMC uses a hyperplane line for optimal binary classification, and the mathematical representation for this hyperplane is:

$$p(e) = \alpha^T \cdot e + a,$$

where e indicates the link (input parameter) also known as feature vector, and the hyperplane of MMC is defined by vector α and a bias value a.

In the SVM model, we assume that the training sample consists of $Z = \{e_i, s_i\}$, where $i = 1, 2, 3, \ldots, m$, and m represents the total number of links. The status of a link s_i can be -1 or 1, (i.e., $s_i \in \{-1, 1\}$), where $s_i = 1$ means $p(e) > 0$ and $s_i = -1$ shows that $p(e) < 0$. For the vector α and bias a values, the SVM model finds the closest points to the hyperplane; therefore, the perpendicular distance from e_i to the hyperplane can be mathematically expressed as:

$$\left|\alpha^T \cdot e_i + a\right| / \|\alpha\| = s_i \cdot \left(\alpha^T \cdot e_i + a\right) / \|\alpha\|.$$

The prime objective of the SVM model is to get the values of a and α which maximize the distance, as shown in Eq. (15):

$$\arg\max_{\alpha,a} \left\{ \frac{2}{\|\alpha\|} \min_i \left[s_i \cdot \left(\alpha^T \cdot e_i + a\right)\right] \right\}, \tag{15}$$

where for an optimum hyperplane, the values of $\|\alpha\|$ and bias a should be optimized in Eq. (15), as shown in Eq. (16).

$$\arg\min_{\alpha,a} \left\{ \frac{1}{2} \|\alpha\|^2 \right\}. \tag{16}$$

In order to get the optimum hyperplane, the $\frac{2}{\|\alpha\|}$ is replaced with the equivalent factor $\frac{1}{2}\|\alpha\|^2$. In the SVM model, if the data contains the non-linear distribution, then a positive slack variable ε is used in the optimization process, (i.e., ε_i, *where* $i = 1, \ldots, q$). By using the slack variable ε, we can optimize the binary classification of SVM and minimize the non-linear distribution, as shown in Algorithm 2. Algorithm 2 is an optimized status s_i classifier algorithm for SVM model on sample $Z = \{e_i, s_i\}$, where $0 < \varepsilon_i < 1$ means that data is classified correctly and if the value of $\varepsilon_i > 1$, then data is classified incorrectly. In SVM, if $\varepsilon_i > 1$ (i.e., data is perfectly separable) leads to an overfitting problem.

Algorithm 2: SVM Classifier

Input: sample $Z = \{e_i, s_i\}$, slack variable ε_i, where $i = 1, \ldots, q$
Output: status s_i of a link e_i
// constant complexity time $\mathcal{O}(c)$
$\varepsilon \leftarrow 0$ // $\mathcal{O}(c)$
// hyperplane-upper bound of all the sum of ε
hyperplane = H // $\mathcal{O}(c)$
for $i = 1$ **to** n **do** // $\mathcal{O}(n)$
  $\varepsilon = \varepsilon + \varepsilon_i$ // $\mathcal{O}(c)$
  $H \geq \varepsilon$ // Eq. (18) // $\mathcal{O}(c)$
if $\varepsilon \leq H$ **then** // $\mathcal{O}(c)$
  $p(e_i) = \begin{cases} s_i = 1 \iff & \alpha \cdot e_i + b \geq 1 - \varepsilon_i \\ s_i = -1 \iff & \alpha \cdot e_i + b \leq -1 + \varepsilon_i \end{cases}$
else if $\varepsilon > H$ **then** // $\mathcal{O}(c)$
  // use radial kernel function K(.)
  // Eq. (19)
else // $\mathcal{O}(c)$
  overfitting problem
return s_i

To handle the overfitting problem, Soft Margin Classifier (SMC) [76] is used in SVM. SMC optimizes the non-linear distribution, as shown in Eq. (17):

$$\begin{aligned} &\max N \\ &\alpha_1, \alpha_2, \ldots, \alpha_m, \varepsilon_1, \varepsilon_2, \ldots, \varepsilon_q, \\ &subject\ to \quad \sum_{i=1}^{m} \alpha_i^2 = 1, \\ &s_i(\alpha_0 + \alpha_1 e_{i1} + \cdots + \alpha_m e_{im}) \geq N(1 - \varepsilon_i), \\ &s_i(\alpha_0 + \alpha_1 e_{i1} + \cdots + \alpha_m e_{im}) \geq N(1 - \varepsilon_i), \end{aligned} \tag{17}$$

$$\sum_{i=1}^{q} \varepsilon_i \leq H, \quad \varepsilon_i \geq 0, \tag{18}$$

where N represents the margin width and ε represents the slack variables allowing observations that data is not perfectly separable/classified. The main objective of SMC is to increase the margin width as much as possible. While H is the hyperplane, is in fact the upper bound of the sum of all ε values, as shown in Eq. (18). A large value of H shows a large margin width and low value denotes less tolerance of miss-classifications [30].

The SVM model uses kernel functions $K(.)$ for linear separation of data. The kernel function reduces the computational power as well as does the faster calculations. In the PrePass-Flow, we use a radial kernel model, as shown in Eq. (19):

$$K(e_i, e_{i'}) = exp\left(-\gamma \sum_{i=1}^{q} (e_{ij} - e_{i'j})^2\right), \tag{19}$$

where γ is the tuning parameter of the kernel function K, which controls the variance of the model. A low value of γ leads to high variance [30].

4.3. Complexity analysis of LR and SVM algorithm

Suppose we have M samples (training samples) in the dataset. The independent variables are $e_1, e_1, \ldots e_n$, and each independent variable is associated with a value of the dependent variable (output) represented by s. In the LR and SVM algorithms, s is either $s = 1$ or $s = 0$. LR Algorithm 1 consists of statements and while loop to compute LR coefficients α after the n iterations. The statements take constant time $\mathcal{O}(c)$. Subsequently, Step 6, Step 7, and Step 8 involve computing the values in the matrix. The complexity is $\mathcal{O}(n^2 + c)$. Therefore, the worst-case complexity of Algorithm 1 is: $\mathcal{O}((c) + (n + c) + (n^2 + c)) \approx \mathcal{O}(n^2)$. Similarly, Algorithm 2 consists of statements and a loop. Therefore, the worst-case complexity of Algorithm 2 is: $\mathcal{O}((c) + (n)) \approx \mathcal{O}(n)$.

5. Performance evaluation and results

5.1. Experimental setup

To evaluate the performance of PrePass-Flow, our test-bed includes traces of a real-life network.[4] The dataset consists of 516 links and 38 switches. For evaluation, we constructed a hybrid SDN topology, using a random topology generator [77], in which we increased the network size. For simulation, we have used the Mininet[5] network emulator with POX[6] controller. The simulation was carried on Intel core-i7 PC with 3.40 GHz CPU and 12 GB of RAM, running 64 bits Ubuntu 16.04 LTS OS having Linux kernel version 4.4.

The Python library (NetworkX) is used to implement the network topology in Mininet. In hybrid SDN, we assume that OpenFlow switches are the subset of legacy switches in a network. In Mininet, an OpenFlow Switch can act as a legacy switch by setting the

[4] COMSATS University Islamabad (CUI), Attock Campus, Pakistan, https://attock.comsats.edu.pk/.
[5] http://mininet.org.
[6] http://github.com/noxrepo/pox.

OVS mode as standalone and disconnect it from the SDN controller. Legacy switches run the OSPF protocol [6], and SDN switches use the OpenFlow protocol.[7] Upon session establishment, the controller should send an $OFPT\_FEATURES\_REQUEST$ message after 10 s. The switch responds with an $OFPT\_FEATURES\_REPLY$ message. Along with other information, the capabilities field is a bitmap that uses a combination of the different flags, like $OFPC\_FLOW\_STATS$, $OFPC\_PORT\_STATS$, and $OFPC\_FLOW\_MONITORING$. When a link or a port down. Then the SDN switch sends $OFPPC\_PORT\_DOWN$ and $OFPPS\_LINK\_DOWN$ to the controller. In hybrid SDN, the centralized controller is capable of modifying the routing table on both legacy and OpenFlow switches, using the Telekinesis module [78]. We specified the ACL policies at the POX controller. POX controller installs the ACL policies for a flow at SDN switches. The controller obtains the link status information from the network regularly. This is referred to as a time slot (i.e., 10 s). We randomly selected different percentages (%) of SDN switches e.g., 5%, 10%, 15%, and 20%. Furthermore, we allowed every switch to receive randomly generated flows (from 2 to 15) from an end-host having a delay in [2, 15] ms interval. Additionally, for each flow, the link bandwidth demand was also generated randomly in the [5, 15] Mbps interval. To place the ACL policies on optimum interfaces in a hybrid SDN, we used the concept introduced by the [14].

5.2. Training LR and SVM models

For training and testing the machine learning models, i.e., Logistic Regression (LR) [29] and Support Vector Machine (SVM) [30], we have utilized the real network dataset discussed in Section 5.1. The dataset consists of links' failure statistics of about 1 year, from April 15, 2019 to March 25, 2020.

Dataset: The main file of the dataset (syslog file) contains an event log that includes data about the type of network element experienced, type of event, small descriptive text, and ID of the event. These events were detected by SNMP monitoring on the interface of the switches. After cleaning the syslog data, we filtered out all failure events such as switch failure and link failure. The resultant filtered set consisted of 79% actionable events in about 810,000 events per hour and was used to analyze and train the LR and SVM models. As the filtered set is not large enough, we have used the leave-one-out cross-validation approach [79]. Additionally, we divided the dataset into two subsets: 70% training dataset and 30% testing dataset. In the training and testing phases, the Scikit-learn library is used. After the training phase, we generate synthesized traffic to test the proposed model.

5.3. Validation and evaluation of LR and SVM models

To train and evaluate the outcome of the two models, we split the dataset into training (70%) and testing (30%) sets. We evaluate the performance using a confusion matrix approach, which is a useful method to demonstrate the prediction results of ML models such as LR and SVM. In Confusion Matrix (CM), $s = 1$ represents a link is functional and $s = 0$ shows that it is down, as shown in Table 2. From the CM values, distinct performance parameters can be computed.

To evaluate the prediction interval of both LR and SVM, we have considered four parameters: (i) accuracy, (ii) precision, (iii) sensitivity, and (iv) Receiver Operator Characteristics (ROC) and Area Under the Curve (AUC). We selected these parameters because they are broadly used in binary classification models.

[7] https://opennetworking.org/wp-content~/uploads/2014/10/openflow-switch-v1.5.1.pdf.

Table 2
Confusion matrix.

		Predicted link status	
		1	0
Actual link status	1	True Positive (TP)	False Positive (FP)
	0	False Negative (FN)	True Negative (TN)

Table 3
LR and SVM model results comparison.

Model	Accuracy	Precision	Sensitivity
LR	0.87	0.74	0.81
SVM	0.83	0.69	0.71

- Accuracy: It is referred to how correctly (in %) a model classified all samples. However, accuracy does not show positive and negative classifications. It is appropriate when the observations distrusted reasonably between the two classes.

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

- Precision: It shows the ability of a model to find all appropriate occurrences. This means the % of links predicted to down that actual town.

$$Precision = \frac{TP}{TP + FP}$$

- Sensitivity: It is known as the Recall or true positive rate because it refers to how well the model classifies true values. It is the proportion of entries with the target condition and gives positive test results.

$$Sensitivity = \frac{TP}{FN + TP}$$

- Receiver Operator Characteristics (ROC) and Area Under the Curve (AUC): For model performance, the ROC and AUC metrics are used. The higher value of AUC shows that the model's performance is excellent and lower means the model is not good enough.

The PrePass-Flow examines the sensitivity and precision parameters along with accuracy. However, the accuracy parameter does not always indicate the actual results, mainly when dealing with a non-linear dataset. Therefore, we also compute the precision and sensitivity of the proposed PrePass-Flow. Table 3 shows the computed values of parameters for both models. We can see that the LR model performs better in terms of accuracy, precision, and sensitivity. The accuracy of the LR model is 87%, while the SVM model's accuracy is 83%. The LR model has a much higher precision score than the SVM model. The precision parameter indicates that the precision of true positive or negative was high in LR. Moreover, the LR also performs better in the sensitivity evaluation.

Furthermore, in Fig. 4, the ROC plot of the outcome of both models are shown. We can see that LR's AUC (0.79) is higher than the SVM (0.73). The AUC curve of LR shows that at 0.5 False Positive Rate, the True Positive Rate is almost 1 compared to the SVM model. From the above results and analysis, we can conclude that the LR model can predict the link status, e.g., $s = 0$ or $s = 1$ more accurately than the SVM model.

5.4. PrePass-Flow: Performance parameters

We evaluate the performance of the proposed technique, PrePass-Flow, by considering that there are a various number of ACL policies and switches that change in time. Furthermore, we compare the performance with a state-of-art model called AutoConfig [13]. For comparison and simulation, we consider the following parameters.

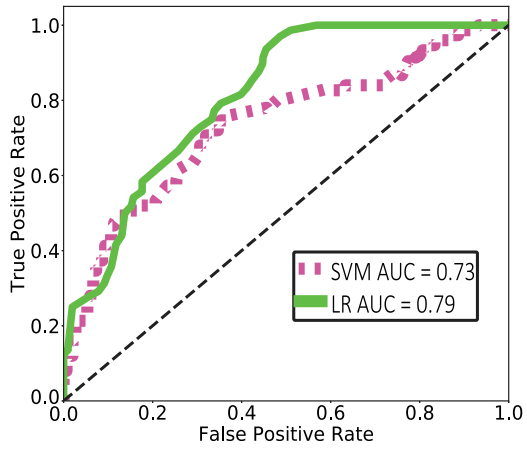


Fig. 4. ROC and AUC curve for LR vs. SVM model.

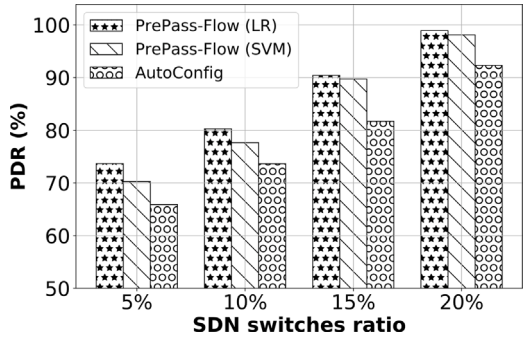


Fig. 5. SDN switches ratio vs. PDR.

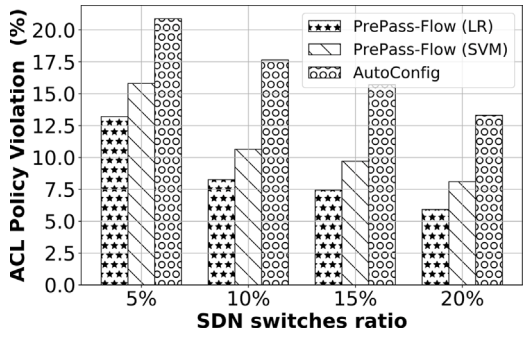


Fig. 6. Impact of SDN switches (%) on ACL policy violation.

i. *Impact of SDN switches (percentage):* It is to randomly select the % of SDN switch (i.e., 5%, 10%, 15%, and 20%) from the total number of switches.
ii. *Impact of links failures (percentage):* It is varying the links failure ratio in the network during simulation, e.g., 1%, 5%, 10%, 15%, and 20%.
iii. *Packets Delivery Ratio (PDR) (percentage):* PDR ratio shows the performance of the model. According to the defined ACL policy, the ratio between the total number of packets sent from the source node, and the total number of packets received at the target node is calculated. The performance of a model is considered good if its PDR ratio is high.
iv. *Packets policy violated (percentage):* It is the number of ACL policy violations in the network. For the ACL rules implementation in the network, we used the method proposed in [14]. Consequently, we also consider the number of packets dropped due to network reachability problem because it also comes under the ACL policy violations.

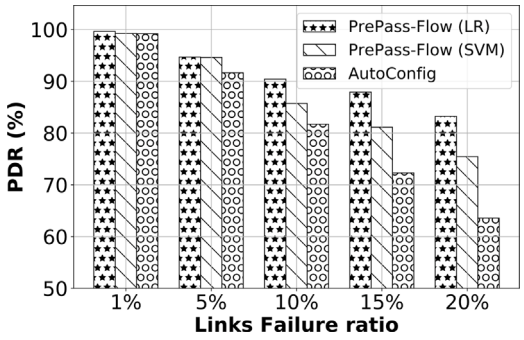


Fig. 7. Link failure effect on PDR.

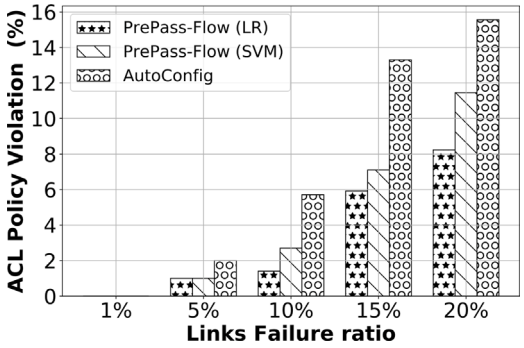


Fig. 8. Link failure effect on ACL policy violation.

5.5. Simulations and results

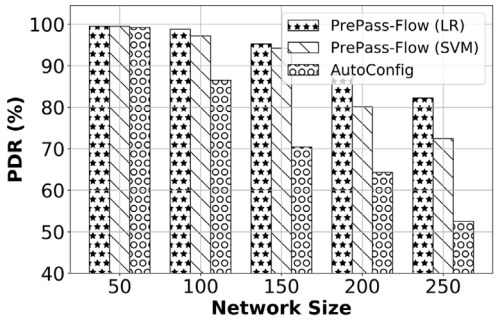
5.5.1. Impact of SDN switches on PDR

In the simulation, we kept the links' failure ratio constant (10%). From the results, we compare the impact of SDN switches on PDR, as shown in Fig. 5. We can see that PDR has increased when the number of SDN switches increases in the network. Furthermore, when the SDN switches increased to 15%, the PDR of the PrePass-Flow (LR) was 90%, and the PDR of PrePass-Flow (SVM) was almost 90%. The PDR of the AutoConfig model was always lower than the two models used in the PrePass-Flow.

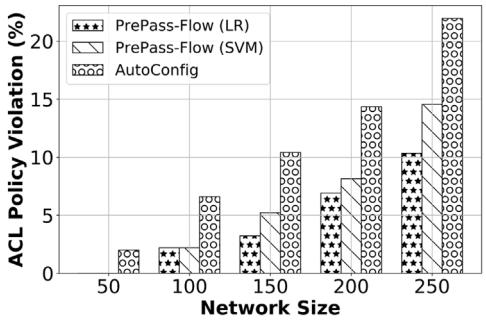
It is worth noting that when the SDN switches ratio was 20%, the PDR ratio of both models of PrePass-Flow was almost 100%. The main reason for this is that when the ratio of SDN switches increases, the controller obtains more efficiently the status of the SDN links. Therefore, the PrePass-Flow (LR) model can predict the status of the links early, can compute the alternative path and install the ACL rules in advance for the affected flows. However, the AutoConfig model installs the ACL rules once the link failure occurs, which results in low PDR.

5.5.2. Impact of SDN switches on ACL policy violation

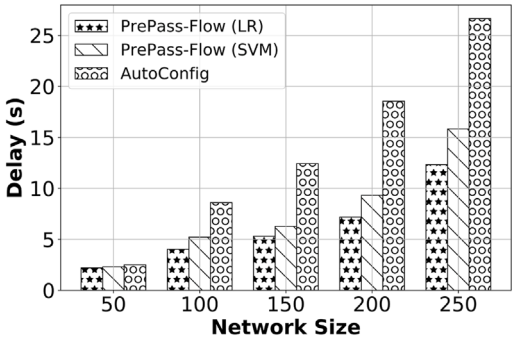
To find the impact of SDN switches on ACL policy violation, we kept the link failure ratio constant (10%), while varying the SDN switches ratio from 5% to 20%. The results show that the proposed PrePass-Flow (LR) and PrePass-Flow (SVM) performed better than the AutoConfig model, as shown in Fig. 6. Furthermore, it is worth noting that the number of SDN switches also has an impact on the ACL policy violation ratio. The main reason is that when the number of SDN switches was low, the controller had received few updated status info of the legacy links. Less updated status information of links' failure determined high number of ACL policy violations. In Fig. 6, when the SDN switches ratio



(a) Network Size vs. PDR (%)



(b) Network Size vs. ACL policy violation (%)



(c) Network Size vs. Delay (s)

Fig. 9. Impact of network size on PDR (%), ACL policy violation (%), and end-to-end delay (s).

was 5%, the ACL violation in AutoConfig is more than 20%. However, in the PrePass-Flow (LR) model, the ACL policy violation was only 13%. In the case of the PrePass-Flow (SVM) model, the ACL policy violation was more than 15%, but lower than the AutoConfig model. Additionally, when SDN switches ratio was 20%, the PrePass-Flow (LR) model performs notably well.

5.5.3. Impact of links' failure on PDR

To find the impact of links' failure on PDR, we performed a simulation in which we varied the ratio of links failure while the percentage of SDN switches was kept constant at 10%. The results are shown in Fig. 7. We can see that the links' failure ratio drastically decreases the PDR ratio in the AutoConfig model. However, the proposed PrePass-Flow (LR) and PrePass-Flow (SVM) model variants achieved high PDR ratio results. It is worth seeing that PrePass-Flow (LR) model achieved more than 80% PDR ratio even when the links' failure ratio was 20%. Additionally, PrePass-Flow (SVM) model had about 75% PDR ratio. The main reason for the high PDR ratio of the proposed PrePass-Flow is accurately predicting the links' failures and rerouting flows efficiently. The AutoConfig model underperformed when the links' failure was 20%, and SDN switches were 10%.

5.5.4. Impact of links' failure on ACL policy

In a hybrid SDN network, links' failure has a high impact on the ACL policy violations. The reason is that the legacy links status information passes through SDN switches to reach the controller, causing delays. Therefore, the controller does not have the updated information about legacy links and may forward the packet on a broken link or cause a policy violation. The results in Fig. 8 show that increases in link failure ratio also determine increases in the ACL policy violation. Additionally, the proposed PrePass-Flow (LR) can predict more efficiently than PrePass-Flow (SVM). Therefore, in the PrePass-Flow (LR), the ACL violation was very low compared to the other two models. However, PrePass-Flow (SVM) performed better than the AutoConfig model.

From the above results, we can conclude that the prediction of a link failure in a hybrid SDN is a critical parameter because it dramatically affects the PDR ratio. The results show that PrePass-Flow (LR) performed the best among the compared approaches in terms of PDR ratio reducing the impact of link failures on ACL policy violation and network performance.

5.5.5. Impact of network size

To evaluate the PrePass-Flow performance using an extensive network, we have created a hybrid SDN topology and increased the number of switches from 50 to 250; the results are shown in Fig. 9. In this scenario, we have considered 10% SDN switches, and the remaining ones were used as legacy switches. We have kept the failure ratio fixed at 15%. From the simulation results, we can see that our proposed PrePass-Flow models performed better than the existing approach (AutoConfig) in terms of PDR (Fig. 9(a)), ACL policy violation (Fig. 9(b)), and delay (Fig. 9(c)).

The results plotted in Fig. 9(a) show that as the network size increased, the PDR decreased in all models. However, in both proposed models, i.e., PrePass-Flow (LR) and PrePass-Flow (SVM), the effect was considerably less evident as compared to that of the AutoConfig model. Additionally, it is worth noting that the proposed models performed better when the SDN ratio was only 10%, and link failure was 15%.

Furthermore, in Fig. 9(b), the ACL policy violations in the PrePass-Flow models were considerably fewer compared to the existing approach. The results show that the AutoConfig model performed relatively well in a small network, but it performed inadequately in a large network. In the PrePass-Flow (LR), ACL policy violations are 10% lower than those experienced by the AutoConfig model. Moreover, Fig. 9(c) shows the results of the end-to-end delay, which is an essential parameter of the network performance. From the results, we can see that the intended models had lower end-to-end delay than the AutoConfig model when the network size reached 250 switches.

The main reason for the outstanding performance of the proposed models in PrePass-Flow is the proactive prediction of the outages. After the prediction the PrePass-Flow proactively computes the alternative paths and installs the ACL policy (if applicable).

6. Conclusion

In this paper, we proposed *PrePass-Flow:* an ML-based technique that reduces ACL policy violations and network reachability problems due to installed ACL policies on the interfaces, which occur due to network-layer failures in a hybrid SDN. The PrePass Flow employs two ML models: Logistic Regression (LR) and Support Vector Machine (SVM). The PrePass-Flow is deployed into a SDN controller, where the models are trained and used to predict network-layer link failures. When a network-layer failure is predicted, the PrePass-Flow initiates protection measures in advance, rerouting the flows as well as installing ACL policies (if necessary) on alternative paths to avoid ACL violations, network reachability problems, and flow disturbances. For performance evaluation, we have used real-life network traces and computed the accuracy, precision, and sensitivity parameters. The results show that the PrePass-Flow (LR) model has 87% accuracy, and the PrePass-Flow (SVM) model has 83%. Furthermore, the PrePass-Flow (LR) model outperforms PrePass-Flow (SVM) model and an existing approach in terms of PDR ratio and number of ACL policy violations. The future work will include using unsupervised ML models such as clustering or deep learning for software level failure prediction.

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 supported in part by the ‘‘National Key Research and Development Plan" with No. 2017YFC0821003-2, ‘‘Dalian Science and Technology Innovation Fund" with No. 2019J11CY004. G.-M. Muntean acknowledges the support of the Science Foundation Ireland Research Centres Programme grants 12/RC/2289_P2 (Insight) and 16/SP/3804 (ENABLE). Nadir Shah wishes to acknowledge the support of Higher Education Commission (HEC), Pakistan under the project No. 5372/Federal/NRPU/RD/HEC/2016, titled ‘‘FAULT LOCALIZATION IN TERM OF SECURITY AND MANAGEMENT POLICY IN SOFTWARE DEFINED NETWORKS.

References

[1] L. Csikor, M. Szalay, G. Retvari, G. Pongracz, D.P. Pezaros, L. Toka, Transition to SDN is HARMLESS: Hybrid architecture for migrating legacy ethernet switches to SDN, IEEE/ACM Trans. Netw. 28 (1) (2020) 275–288.

[2] R. Alvizu, G. Maier, N. Kukreja, A. Pattavina, R. Morro, A. Capello, C. Cavazzoni, Comprehensive survey on T-SDN: Software-defined networking for transport networks, IEEE Commun. Surv. Tutor. 19 (4) (2017) 2232–2283.

[3] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, et al., B4: Experience with a globally-deployed software defined WAN, ACM SIGCOMM Comput. Commun. Rev. 43 (4) (2013) 3–14.

[4] R. Amin, M. Reisslein, N. Shah, Hybrid SDN networks: A survey of existing approaches, IEEE Commun. Surv. Tutor. 20 (4) (2018) 3259–3306.

[5] N. Shah, P. Giaccone, D.B. Rawat, A. Rayes, N. Zhao, Solutions for adopting software defined network in practice, Int. J. Commun. Syst. 32 (17) (2019) e3990.

[6] T.Y. Cheng, X. Jia, Compressive traffic monitoring in hybrid SDN, IEEE J. Sel. Areas Commun. 36 (12) (2018) 2731–2743.

[7] C.-Y. Chu, K. Xi, M. Luo, H.J. Chao, Congestion-aware single link failure recovery in hybrid SDN networks, in: IEEE INFOCOM, 2015, pp. 1086–1094.

[8] X. Jia, Y. Jiang, J. Zhu, Link fault protection and traffic engineering in hybrid SDN networks, in: IEEE INFOCOM, 2018, pp. 853–858.

[9] M. Caria, A. Jukan, Link capacity planning for fault tolerant operation in hybrid SDN/OSPF networks, in: Global Communications Conference (GLOBECOM), IEEE, 2016, pp. 1–6.

[10] D.K. Hong, Y. Ma, S. Banerjee, Z.M. Mao, Incremental deployment of SDN in hybrid enterprise and ISP networks, in: Symposium on SDN Research, 2016, pp. 1–7.

[11] K. Poularakis, G. Iosifidis, G. Smaragdakis, L. Tassiulas, One step at a time: Optimizing SDN upgrades in ISP networks, in: IEEE INFOCOM, 2017, pp. 1–9.

[12] S. Agarwal, M. Kodialam, T. Lakshman, Traffic engineering in software defined networks, in: IEEE INFOCOM, 2013, pp. 2211–2219.

[13] R. Amin, N. Shah, B. Shah, O. Alfandi, Auto-configuration of ACL policy in case of topology change in hybrid SDN, IEEE Access 4 (2016) 9437–9450.

[14] R. Amin, N. Shah, W. Mehmood, Enforcing optimal ACL policies using K-Partite graph in hybrid SDN, Electronics 8 (6) (2019) 604.

[15] S.-Y. Wang, C.-C. Wu, C.-L. Chou, Constructing an optimal spanning tree over a hybrid network with SDN and legacy switches, in: IEEE Symposium on Computers and Communication (ISCC), 2015, pp. 502–507.

[16] M. Ibrar, L. Wang, G.M. Muntean, J. Chen, N. Shah, A. Akbar, IHSF: An intelligent solution for improved performance of reliable and time-sensitive flows in hybrid SDN-based FC IoT systems, IEEE Internet Things J. (2020) 1.

[17] B. Tian, X. Zhang, E. Zhai, H.H. Liu, Q. Ye, C. Wang, X. Wu, Z. Ji, Y. Sang, M. Zhang, et al., Safely and automatically updating in-network ACL configurations with intent language, in: Proceedings of the ACM Special Interest Group on Data Communication, 2019.

[18] N. Foster, A. Guha, M. Reitblatt, A. Story, M.J. Freedman, N.P. Katta, C. Monsanto, J. Reich, J. Rexford, C. Schlesinger, et al., Languages for software-defined networks, IEEE Commun. Mag. 51 (2) (2013) 128–134.

[19] C. Monsanto, J. Reich, N. Foster, J. Rexford, D. Walker, Composing software defined networks, in: Networked Systems Design and Implementation (NSDI), 2013, pp. 1–13.

[20] P. Kazemian, G. Varghese, N. McKeown, Header space analysis: Static checking for networks, in: Networked Systems Design and Implementation (NSDI), 2012, pp. 113–126.

[21] C. Prakash, J. Lee, Y. Turner, J.-M. Kang, A. Akella, S. Banerjee, C. Clark, Y. Ma, P. Sharma, Y. Zhang, Pga: Using graphs to express and automatically reconcile network policies, ACM SIGCOMM Comput. Commun. Rev. 45 (4) (2015) 29–42.

[22] K.R. Malik, T. Ahmad, M. Farhan, M. Alfawair, Enhancing SDN performance by enabling reasoning abilities in data traffic control, Peer-to-Peer Netw. Appl. 12 (2) (2019) 392–404.

[23] N. Sultana, N. Chilamkurti, W. Peng, R. Alhadad, Survey on SDN based network intrusion detection system using machine learning approaches, Peer-to-Peer Netw. Appl. 12 (2) (2019) 493–501.

[24] A. Khurshid, X. Zou, W. Zhou, M. Caesar, P.B. Godfrey, Veriflow: Verifying network-wide invariants in real time, in: Networked Systems Design and Implementation (NSDI), 2013, pp. 15–27.

[25] I.I. Awan, N. Shah, M. Imran, M. Shoaib, N. Saeed, An improved mechanism for flow rule installation in-band SDN, J. Syst. Archit. 96 (2019) 1–19.

[26] H. Xu, X.-Y. Li, L. Huang, H. Deng, H. Huang, H. Wang, Incremental deployment and throughput maximization routing for a hybrid SDN, IEEE/ACM Trans. Netw. 25 (3) (2017) 1861–1875.

[27] T. Feng, J. Bi, Openrouteflow: Enable legacy router as a software-defined routing service for hybrid SDN, in: IEEE International Conference on Computer Communication and Networks (ICCCN), 2015, pp. 1–8.

[28] H. Lu, N. Arora, H. Zhang, C. Lumezanu, J. Rhee, G. Jiang, Hybnet: Network manager for a hybrid network infrastructure, in: Proceedings of the Industrial Track of the 13th ACM/IFIP/USENIX International Middleware Conference, 2013, pp. 1–6.

[29] Y.Y. Liu, M. Yang, M. Ramsay, X.S. Li, J.W. Coid, A comparison of logistic regression, classification and regression tree, and neural networks models in predicting violent re-offending, J. Quant. Criminol. 27 (4) (2011) 547–573.

[30] G. James, D. Witten, T. Hastie, R. Tibshirani, An Introduction to Statistical Learning: with Applications in R, seventh ed., Springer, New York, 2017.

[31] R. Boutaba, M.A. Salahuddin, N. Limam, S. Ayoubi, N. Shahriar, F. Estrada-Solano, O.M. Caicedo, A comprehensive survey on machine learning for networking: evolution, applications and research opportunities, J. Internet Serv. Appl. 9 (1) (2018) 16.

[32] M. Usama, J. Qadir, A. Raza, H. Arif, K.-L.A. Yau, Y. Elkhatib, A. Hussain, A. Al-Fuqaha, Unsupervised machine learning for networking: Techniques, applications and research challenges, IEEE Access 7 (2019) 65579–65615.

[33] Z. Wang, M. Zhang, D. Wang, C. Song, M. Liu, J. Li, L. Lou, Z. Liu, Failure prediction using machine learning and time series in optical network, Opt. Express 25 (16) (2017) 18553–18565.

[34] C. Yu, J. Lan, Z. Guo, Y. Hu, Drom: Optimizing the routing in software-defined networks with deep reinforcement learning, IEEE Access 6 (2018) 64533–64539.

[35] T. Holterbach, E.C. Molero, M. Apostolaki, A. Dainotti, S. Vissicchio, L. Vanbever, Blink: Fast connectivity recovery entirely in the data plane, in: Networked Systems Design and Implementation (NSDI), 2019, pp. 161–176.

[36] K. Qiu, J. Zhao, X. Wang, X. Fu, S. Secci, Efficient recovery path computation for fast reroute in large-scale software-defined networks, IEEE J. Sel. Areas Commun. 37 (8) (2019) 1755–1768.

[37] G. Iannaccone, Analysis of link failures in a IP backbone, in: Internet Measurement Workshop, 2002, http://www.icir.org/vern/imw-2002/imw2002-papers/202.pdf.
[38] D. Turner, K. Levchenko, A.C. Snoeren, S. Savage, California fault lines: understanding the causes and impact of network failures, in: ACM SIGCOMM, 2010, pp. 315–326.
[39] G. Bankhamer, R. Elsasser, S. Schmid, Local fast rerouting with low congestion: A randomized approach, in: IEEE International Conference on Network Protocols (ICNP), 2019, pp. 1–11.
[40] B. Yang, J. Liu, S. Shenker, J. Li, K. Zheng, Keep forwarding: Towards k-link failure resilient routing, in: IEEE INFOCOM, 2014, pp. 1617–1625.
[41] V. Liu, D. Halperin, A. Krishnamurthy, T. Anderson, F10: A fault-tolerant engineered network, in: Networked Systems Design and Implementation (NSDI), 2013, pp. 399–412.
[42] A. Greenberg, J.R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D.A. Maltz, P. Patel, S. Sengupta, VL2: a scalable and flexible data center network, Commun. ACM 54 (3) (2011) 95–104.
[43] H.H. Liu, S. Kandula, R. Mahajan, M. Zhang, D. Gelernter, Traffic engineering with forward fault correction, in: ACM Conference on SIGCOMM, 2014, pp. 527–538.
[44] T. Holterbach, S. Vissicchio, A. Dainotti, L. Vanbever, Swift: Predictive fast reroute, in: ACM Special Interest Group on Data Communication, 2017, pp. 460–473.
[45] P. Gill, N. Jain, N. Nagappan, Understanding network failures in data centers: measurement, analysis, and implications, in: Proceedings of the ACM SIGCOMM 2011 Conference, 2011, pp. 350–361.
[46] J.J. LaViola, Double exponential smoothing: an alternative to Kalman filter-based predictive tracking, in: Proceedings of the Workshop on Virtual Environments 2003, 2003, pp. 199–206.
[47] M. Casado, M.J. Freedman, J. Pettit, J. Luo, N. Gude, N. McKeown, S. Shenker, Rethinking enterprise network control, IEEE/ACM Trans. Netw. 17 (4) (2009) 1270–1283.
[48] D. Levin, M. Canini, S. Schmid, F. Schaffert, A. Feldmann, Panopticon: Reaping the benefits of incremental SDN deployment in enterprise networks, in: Annual Technical Conference USENIX, 2014, pp. 333–345.
[49] X. Huang, S. Cheng, K. Cao, P. Cong, T. Wei, S. Hu, A survey of deployment solutions and optimization strategies for hybrid SDN networks, IEEE Commun. Surv. Tutor. 21 (2) (2018) 1483–1507.
[50] M. Markovitch, S. Schmid, SHEAR: A highly available and flexible network architecture marrying distributed and logically centralized control planes, in: IEEE International Conference on Network Protocols (ICNP), 2015, pp. 78–89.
[51] Y.-W.E. Sung, X. Sun, S.G. Rao, G.G. Xie, D.A. Maltz, Towards systematic design of enterprise networks, IEEE/ACM Trans. Netw. 19 (3) (2010) 695–708.
[52] K. Jayaraman, N. Bjørner, G. Outhred, C. Kaufman, Automated analysis and debugging of network connectivity policies, Microsoft Res. (2014) 1–11.
[53] L. Akoglu, H. Tong, D. Koutra, Graph based anomaly detection and description: a survey, Data Min. Knowl. Discov. 29 (3) (2015) 626–688.
[54] M. Rzepka, P. Borylo, A. Lason, A. Szymanski, PARD: Hybrid proactive and reactive method eliminating flow setup latency in SDN, J. Netw. Syst. Manage. (2020) 1–28.
[55] H. Zeng, P. Kazemian, G. Varghese, N. McKeown, Automatic test packet generation, in: International Conference on Emerging Networking Experiments and Technologies, 2012, pp. 241–252.
[56] M. Hussain, N. Shah, Automatic rule installation in case of policy change in software defined networks, Telecommun. Syst. 68 (3) (2018) 461–477.
[57] M. Hussain, N. Shah, A. Tahir, Graph-based policy change detection and implementation in SDN, Electronics 8 (10) (2019) 1136.
[58] V. Zaborovsky, V. Mulukha, A. Silinenko, S. Kupreenko, Dynamic firewall configuration: Security system architecture and algebra of the filtering rules, in: International Conference on Evolving Internet–INTERNET, 2011, pp. 19–24.
[59] J. Xie, F.R. Yu, T. Huang, R. Xie, J. Liu, C. Wang, Y. Liu, A survey of machine learning techniques applied to software defined networking (SDN): Research issues and challenges, IEEE Commun. Surv. Tutor. 21 (1) (2018) 393–430.
[60] P. Soucy, G.W. Mineau, A simple KNN algorithm for text categorization, in: IEEE International Conference on Data Mining, 2001, pp. 647–648.
[61] Á. López-Raventós, F. Wilhelmi, S. Barrachina-Muñoz, B. Bellalta, Combining software defined networks and machine learning to enable self organizing WLANs, in: IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), 2019, pp. 1–8.
[62] R. Alvizu, S. Troia, G. Maier, A. Pattavina, Matheuristic with machine-learning-based prediction for software-defined mobile metro-core networks, J. Opt. Commun. Netw. 9 (9) (2017) D19–D30.
[63] C. Chen-Xiao, X. Ya-Bin, Research on load balance method in SDN, Int. J. Grid Distrib. Comput. 9 (1) (2016) 25–36.
[64] J. Carner, A. Mestres, E. Alarcón, A. Cabellos, Machine learning-based network modeling: An artificial neural network model vs a theoretical inspired model, in: 2017 Ninth International Conference on Ubiquitous and Future Networks (ICUFN), IEEE, 2017, pp. 522–524.
[65] R. Pasquini, R. Stadler, Learning end-to-end application QoS from openflow switch statistics, in: IEEE Conference on Network Softwarization (NetSoft), 2017, pp. 1–9.
[66] A. Pellegrini, P. Di Sanzo, D.R. Avresky, A machine learning-based framework for building application failure prediction models, in: IEEE International Parallel and Distributed Processing Symposium Workshop, 2015, pp. 1072–1081.
[67] D. Didona, F. Quaglia, P. Romano, E. Torre, Enhancing performance prediction robustness by combining analytical modeling and machine learning, in: Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering, 2015, pp. 145–156.
[68] L. Liao, J. Chen, H. Li, Y. Zeng, W. Shang, J. Guo, C. Sporea, A. Toma, S. Sajedi, Using black-box performance models to detect performance regressions under varying workloads: an empirical study, Empir. Softw. Eng. (2020) 1–31.
[69] A. Mahgoub, P. Wood, S. Ganesh, S. Mitra, W. Gerlach, T. Harrison, F. Meyer, A. Grama, S. Bagchi, S. Chaterji, Rafiki: a middleware for parameter tuning of NOSQL datastores for dynamic metagenomics workloads, in: ACM/IFIP/USENIX Middleware Conference, 2017, pp. 28–40.
[70] P. Di Sanzo, D. Rughetti, B. Ciciani, F. Quaglia, Auto-tuning of cloud-based in-memory transactional data grids via machine learning, in: IEEE Second Symposium on Network Cloud Computing and Applications, 2012, pp. 9–16.
[71] P. Di Sanzo, F. Molfese, D. Rughetti, B. Ciciani, Providing transaction class-based QoS in in-memory data grids via machine learning, in: IEEE Network Cloud Computing and Applications (Ncca 2014), 2014, pp. 46–53.
[72] A. Pellegrini, P. Di Sanzo, D.R. Avresky, Proactive cloud management for highly heterogeneous multi-cloud infrastructures, in: IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2016, pp. 1311–1318.
[73] A. Greenberg, G. Hjalmtysson, D.A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, H. Zhang, A clean slate 4D approach to network control and management, ACM SIGCOMM Comput. Commun. Rev. 35 (5) (2005) 41–54.
[74] S. Uhlig, B. Quoitin, J. Lepropre, S. Balon, Providing public intradomain traffic matrices to the research community, ACM SIGCOMM Comput. Commun. Rev. 36 (1) (2006) 83–86.
[75] S. Misra, N. Saha, Detour: Dynamic task offloading in software-defined fog for IoT applications, IEEE J. Sel. Areas Commun. 37 (5) (2019) 1159–1166.
[76] C. Cortes, V. Vapnik, Soft Margin Classifier, Google Patents, 1997, US Patent 5, 640, 492.
[77] C. Jin, Q. Chen, S. Jamin, Inet: Internet topology generator http://topology.eecs.umich.edu/inet/.
[78] C. Jin, C. Lumezanu, Q. Xu, Z.-L. Zhang, G. Jiang, Telekinesis: Controlling legacy switch routing with openflow in hybrid networks, in: Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research, 2015, pp. 1–7.
[79] S.k. Lee, P. Kang, S. Cho, Probabilistic local reconstruction for k-NN regression and its application to virtual metrology in semiconductor manufacturing, Neurocomputing 131 (2014) 427–439.

Muhammad Ibrar is pursuing his Ph.D. degree at the School of Software, Dalian University of Technology, China. He has completed his MS degree in Telecommunication and Networking from Bahria University, Islamabad, Pakistan, in 2014. He received his BS degree in Telecommunication and Networking from COMSATS University Islamabad, Abbottabad Campus, Pakistan, in 2010. His research interests include hybrid Software-Defined Networking (SDN), fog computing, wireless ad-hoc, and sensor networks.

Lei Wang is currently a full professor of the School of Software, Dalian University of Technology, China. He received his B.S., M.S., and Ph.D. from Tianjin University, China, in 1995, 1998, and 2001, respectively. He was a Member of Technical Staff with Bell Labs Research China (2001-2004), a senior researcher with Samsung, South Korea (2004-2006), a research scientist with Seoul National University (2006-2007), and a research associate with Washington State University, Vancouver, WA, USA (2007-2008). His research interests involve wireless ad hoc networks, sensor networks, social networks, and network security. He has published 160+ papers, and the papers have 2700+ citations. He is a member of the IEEE, ACM and a senior member of the China Computer Federation (CCF).

Gabriel-Miro Muntean (M'04, SM'17) is Professor with the School of Electronic Engineering, Dublin City University (DCU), Ireland, and Co-Director of the DCU Performance Engineering Laboratory. He has published over 400 papers in top-level international journals and conferences, including 4 authored and 6 edited books. His research interests include quality, performance, and energy saving issues related to rich media delivery, technology-enhanced learning, and

other data communications over heterogeneous networks. He is an Associate Editor of the IEEE Transactions on Broadcasting, the Multimedia Communications Area Editor of the IEEE Communications Surveys and Tutorials, and chair and reviewer for important international journals, conferences, and funding agencies.

Aamir Akbar is currently a Lecturer in the Computer Science Department at Abdul Wali khan University in Pakistan. His research focuses on energy-efficient, self-aware, and self-adaptive mobile cloud computing. He is particularly interested in using multi-objective optimization techniques and evolutionary algorithms in IoT, Robotics, and cloud/fog/edge computing. He obtained his Ph.D. at Aston University in the UK. Previously, he worked as a lecturer in the COMSATS Institute of IT in Pakistan. He received an M.Sc. degree from Oxford Brookes University (UK). Before that, he received a B.Sc. degree from COMSATS University Islamabad, Abbottabad Campus, Pakistan.

Nadir Shah received the B.Sc. and M.Sc. degrees from Peshawar University, Peshawar, Pakistan, in 2002 and 2005, respectively, the M.S. degree from International Islamic University, Islamabad, Pakistan, in 2007, all in computer science, and the Ph.D. degree from Sino-German Joint Software Institute, Beihang University, Beijing, China. He was a Lecturer with the Department of Computer Science, COMSATS Institute of Information Technology, Abbottabad, Pakistan, from 2007 to 2008. He is currently an Associate Professor with the COMSATS University Islamabad, Wah Campus. He has authored several research papers in international journals/conferences, such as the ACM Computing Surveys and the IEEE Communication Letters. His current research interests include computer networks, distributed systems, and network security. He is serving in the editorial board of IEEE Softwarizations, AHWSN, IJCS, and MJCS. He has been serving as a Reviewer for several journals/conferences, including ICC, INFOCOM, WCNC, Computer Networks (Elsevier), IEEE Communications Letters, IEEE Communication Magazine, IEEE Transactions on Industrial Informatics, and The Computer Journal.

Kaleem Razzaq Malik received a M.Sc. degree in computer science from the National University of Computer and Emerging Sciences, Lahore, Pakistan in 2008, and a Ph.D. degree in computer science from the University of Engineering and Technology, Lahore, Pakistan in 2018. He is an Associate Professor with Air University, Multan, Pakistan from March 2018. He was an Assistant Professor in COMSATS Institute of Information Technology, Sahiwal, Pakistan between 2015 and 2018 and a Lecturer with Department of Software Engineering, Government College University Faisalabad, Pakistan between 2013 and 2015. He also worked as an instructor of computer science in the Virtual University of Pakistan. He has published various articles in top national and international journals and a book chapter. Dr. Malik has performed reviews for many top-ranking international journals published by Springer, IEEE, and Elsevier and was a member of technical program committees of international conferences. His research interests include *Bigdata, Cloud Computing, Data Sciences, and Semantic Web*.