A Machine-Learning-Based Action Recommender for Network Operation Centers

Shady A. Mohammed, Ayşe Rumeysa Mohammed, David Côté, and Shervin Shirmohammadi, *Fellow, IEEE*

***Abstract*—Failure management and cost-aware traffic engineering are two important tasks done in Network Operation Centers (NOC). These are performed by expert technicians who must carefully analyze the network state and the flow of incoming alarms to decide how, where and when to take actions on the network. While based on implicit guiding principles, these network actions are very hard to automate with explicit rules due to the high complexity of the system; hence NOC action is essentially a manual process today. To automate part of that process, in this paper we introduce an Action Recommendation Engine (ARE) that can learn implicit NOC action rules with supervised machine learning from historical data. As a result, ARE can recommend suitable action(s) to remedy network faults and engineer the traffic to minimize costs, all while maximizing the users' Quality of Experience. To quantify the effectiveness of different NOC action scenarios, we introduce the QoE-OPEX metric which balances between users' quality of Experience and ISP's operational costs. After proper model training on 56,000 data points with 66 features, we demonstrate that ARE can effectively reproduce implicit action-taking logic of NOC technicians, thus moving us one step closer to reliable autonomous networks and fully-automated NOCs.**

***Index Terms*—Network automation, reliable networks, network failure management, traffic engineering, machine learning.**

I. Introduction

IT IS reported in [1] that Internet serves approximately 9 billion clients around the world. Major Internet Service Providers (ISPs) try to balance between guaranteeing their customers a secure and high-quality service in order to meet Service Level Agreements (SLA) and minimizing ISPs' operational costs (OPEX). To keep the network efficiently operating at any time, an ISP assembles a team of network professionals in a *Network Operation Centre* (*NOC*) to closely monitor their network's health [2], as shown in Figure 1. Essentially, the NOC is in charge of service assurance through meeting the customers' SLAs, though it needs to balance that

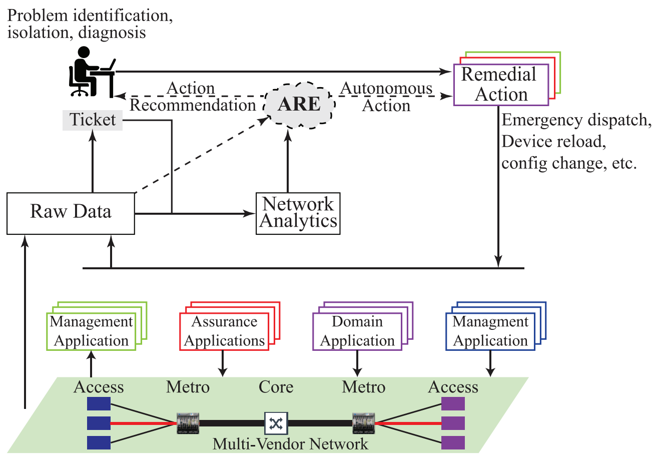


Fig. 1. A typical Network Operation Center (NOC). The ARE component is proposed by this work.

against the ISP's OPEX. As shown in the figure, the NOC collects performance monitoring (e.g., SNMP, ping, traceroute, etc.) and alarms data, which may be logged as tickets. Optionally, a Network Analytics module can analyze data and produce useful insights for NOC technicians (such as the state of network elements). Examples of such Network Analytics modules include Ciena's Unified Assurance and Analytics (UAA) [3], Route Optimization and Analysis (ROA) [4], and similar products by other vendors. In case of any identified network problem, the NOC technicians analyze the problem to find a suitable action that restores the network to its optimum condition. This process is known as *Failure Management*. At the same time, the NOC technicians plan the routing of important and bulky traffic through the ISP's infrastructure to ensure minimum OPEX. This process is known as *Traffic Engineering* (*TE*), which can be implemented by techniques such as MPLS tunnels [5].

In today's NOCs, most of the said failure management and traffic engineering actions are either done manually [6] or, for the simplest cases, with previously prepared expert rules [7]. NOC's operations rely on massive performance monitoring and alarm handling (aka "Network Assurance") systems used by large support teams. A team of NOC technicians works 24/7 to ensure the services are delivered at all times. To handle non-trivial issues, the NOC team relies on specialized support teams organized by technology domains, themselves relying on vendors' technical support on call. Operating with such pyramidal and manual support works: all important problems

Manuscript received November 19, 2020; revised April 11, 2021 and June 30, 2021; accepted July 4, 2021. Date of publication July 8, 2021; date of current version September 9, 2021. This research was financially supported by Mitacs Accelerate Cluster grant # IT12571 and by Ciena Corp. The associate editor coordinating the review of this article and approving it for publication was C. Mas. *(Corresponding author: Shady A. Mohammed.)*

Shady A. Mohammed, Ayşe Rumeysa Mohammed, and Shervin Shirmohammadi are with the Distributed and Collaborative Virtual Environments Research Lab, University of Ottawa, Ottawa, ON K1N 6N5, Canada (e-mail: smoha191@uottawa.ca; amus037@uottawa.ca; shervin.s@uottawa.ca).

David Côté is with the Blueplanet Division, Ciena Corporation, Ottawa, ON K2K 0L1, Canada (e-mail: dcote@ciena.com).

Digital Object Identifier 10.1109/TNSM.2021.3095463

eventually get resolved by taking the appropriate remediation actions. However, this model is getting increasingly expensive and inefficient as networks become more and more complex. To help with this, the networking industry wants to automate NOC operations as much as possible. However, attempts to automate network actions with explicit rules have not been very successful, generally, due to the intrinsic difficulty of defining expert rules that work reliably in such complex systems and the practical difficulty that no single expert knows about all the rules for different network technologies. The network action problem is therefore solved by "collective human intelligence" but not by a single human's intelligence. In our work, we ask: can this be solved by artificial intelligence (AI)? We ask this because even in complex networks, logical relationships still exist between problematic network events and their remediation actions. But it is difficult to codify these relationships from collective domain expertise for complex networks. Hence, we propose to measure them from big datasets with AI, using traditional network assurance data as features and NOC actions as labels. The results, shown in Section IV, are promising and show that AI can indeed find the said relationship and solve the network problem.

Specifically, we use supervised Machine Learning (ML) for this task. The recent advancement of ML has transformed almost every field in the industry, and NOC is no exception. Due to its data-driven nature, ML has the potential to manage network complexity and maximize QoS and QoE. The aforementioned tools like Ciena's Blue Planet UAA and ROA are examples of the industry moving towards ML-based network analytics. These tools use ML to help the NOC technicians gain deeper insights into the network, allowing those technicians to make intelligent data-driven decisions that lead to improved efficiency, lowered costs, and providing more personalized services. Every second, massive amounts of data is transported through networks. Therefore, rapid, accurate, and efficient decision making is no longer a luxury but has become a necessity in today's networks. Since ML is already being used in that process, it makes sense to leverage it one step further to help NOCs in automating much of their decision making too, so that the technicians only need to verify the ML system's recommendation rather than doing the hard analytics work themselves in details [8].

In this paper, we present a ML-based approach to automate two of the NOC's main tasks: failure management and OPEX-aware traffic engineering. We propose an OPEX-aware decision making system that can (1) recommend suitable action(s) to remedy network faults and (2) traffic engineer the ISP's traffic to minimize costs. We call our system Action Recommendation Engine (ARE), shown as such in Figure 1. ARE is a constantly running system that uses the network state, reported by the Network Analytics module, and the raw data, to continuously recommend actions to the NOC technicians (including the action of "do nothing"), shown in the figure with a dashed arrow labeled *Action recommendation*, or directly apply the recommended action in a human-out-of-the-loop fashion, shown in the figure with a dashed arrow labeled *Autonomous action*. The latter is useful for fast recovery. To the best of our knowledge, and as shown in Section II, ARE is the first work that explores the feasibility of using ML for automating the selection of the best action to take, which is valuable for complex networks where manual decision making becomes difficult. Performance results in Section IV show that when ARE is allowed to take actions autonomously, the results very closely follow that of actions taken by NOC technicians. As such, ARE can save both time and money for the tasks that it has been trained for.

The remainder of the paper is organized as follows. In Section II we cover the related work. In III we detail our system design and the rationale behind it. In Section IV introduce the selected ML algorithms and analyze their results. Finally, in Section V, we conclude our work and discuss future research venues.

II. Related Work

In this section, we present the related work done in both academia and industry in the domain of failure recovery and traffic engineering.

A. Failure Recovery

Continuous fault management involves four stages: detection, diagnosis, alerting, and resolving/recovering from fault. Our work falls into the latter stage. In the industry, traditional approaches in failure recovery are offered by MPLS fast reroute which creates a band-aid tunnel between the start and the end points of the failed link to avoid the broken path and provide a localized fast reroute. Examples of this approach are [9] and [10].

In the academia, various approaches have been proposed. Saldamli *et al.* [11] developed a proactive method that predicts link failure and congestion occurrences, and uses a Software Defined Networking (SDN) controller to push rules to the switches before such events happen. In this work, congestion is detected by continuously monitoring the traffic at the OpenFlow switch. When any port's traffic surpasses a certain threshold value, that port is tagged as congested. Then new rules are established by the proactive controller. Additionally, in case of link failure(s), the calculated active and backup paths are presented to route the traffic. Authors in [12] mitigate multi-path failure by proposing *FUSO* which is an algorithm that detects the low-utilized links and routs the traffic through them. It is reported that *FUSO* is suitable for general data center networking.

Liu *et al.* [13] developed a congestion aware and adaptive failure recovery mechanism in data center networks that is resilient to topology changes. This system, combined with the SDN controller, utilizes asymmetrical routing to determine rerouting paths. In [14], shortest-path spanning tree algorithm routes traffic through less congested paths in SDN.

Wang *et al.* [15] proposed a proactive restoration algorithm that chooses the most convenient backup path in the occurrence of single link failure in SDN. In [16], failure recovery is achieved in SDN via a system called *BOND*, which combines proactive and reactive modes to establish backup paths to satisfy the necessary requirements of every application. In [17], with the aid of prioritization of the flow tables in the switch,

TABLE I
COMPARISON SUMMARY OF RELATED WORK

Method	Paper	Contribution	Approach	Platform	AI	Metrics
Failure Recovery	[11]	Traffic level is compared to a threshold to judge port congestion	Pre-Planned rules	SDN	No	Link Utilization
	[12]	Low utilized links detection by FUSO algorithm	Dynamic rules	IP/Data Center	No	Link Utilization
	[13]	A congestion-aware algorithm, utilizes asymmetrical routing	Dynamic rules	Data Center/SDN	No	LLDP packets
	[14]	Elastic-aware routing tree to achieve fast recovery	Dynamic rules	SDN + Wireless	No	No. of Nodes
	[15]	A proactive scheme to achieve link failures recovery via segment routing	Pre-Planned rules	SDN	No	Network Topology
	[16]	A proactive and reactive failure recovery algorithm based on flow demand collection	Pre-Planned rules	SDN	No	Bandwidth
	[17]	An OpenFlow-based scheme based on two timers to measure traffic propagation	Pre-Planned rules	IP	No	Delay
	[18]	A proactive and adaptive mechanism which computes candidate paths between source-destination pair	Dynamic rules	Data Center/SDN	No	Network Topology
Traffic Engineering	[19]	A cross-layer optimized geographic node-disjoint algorithm based on multiple layers' interactions	Multi-path routing	Wireless Networks	No	2-Hop Neighbor Info
	[20]	An algorithm to allocate paths based on source node	Source routing	SDN	No	Network Topology
	[21]	A semi-supervised deep network to select shortest path and max-ming routing	Graph-query NN	IP	Yes	N/A
	[22]	An architecture to predict the traffic paths by using traffic patterns	Deep Neural Network	IP	Yes	Inbound Packet No.
	[23]	An architecture to construct routing tables for a GPU-accelerated Software Defined Routers	Deep Belif Network	SDN	Yes	Inbound Packet No.
	[24]	A path determination and traffic scheduling strategy based on Dijkstra algorithm	Path scheduling	SDN	No	QoS
	[25]	An energy-aware routing mechanism	Path scheduling	SDN	No	Network Topology
	[27]	A load-balancing based elevator scheduling algorithm	Path scheduling	SDN	No	Delay
	[28]	A closed-loop traffic engineering application in the SDN framework	Closed-loop TE App.	SDN	No	N/A
	[29]	A deep learning path planning architecture utilizing Deep Learning	RNN + Attention	SDN	Yes	Sequence of Nodes
	[30]	An ML based traffic-aware path computation architecture	Unsupervised ML	SDN	Yes	Network Topology

a backup path forwarding rule is executed upon a link failure. In [18], after link failures, failure recovery is rapidly and pro-actively obtained thorough pre-configured backup paths.

Despite their advantages, none of the existing works take into account the ISP's OPEX while restoring networks. In addition, some use pre-planned expert rules which can be difficult to determine for larger or more complex networks, such as multi-layer and multi-vendor networks. Finally, none attempt to explicitly maximize the user's QoE by acting on network infrastructure. As we will see in Section III, our proposed ARE system overcomes these shortcomings.

B. Traffic Engineering

For better traffic control, better traffic operation, and better traffic management, several solutions are proposed in the domain of traffic engineering: multi-path routing [19], source routing [20], constructing a new routing protocol design [21], creating a new routing technique [22], optimizing the routing table calculations [23], QoS-aware routing [24], energy-aware routing and flow scheduling [25], flow-based routing [26], and load balancing [27]. In general, traffic engineering revolves around evaluating and optimizing network performance. In SDN, this can be achieved by the controller [28]. In [29], a sequential deep learning model - seq2seq - is implemented in the SDN's controller to provide network level path-planning under restrictive forwarding conditions; for example, the forwarding path has a restriction to include a predefined number of nodes which also requires path connectivity. In that work, the network paths are examined as a serializable data. However, the nature of the seq2seq model includes outputs with variant sizes which might result in unconnected paths. In order to overcome this issue, the seq2seq model is enhanced by (i) an attention mechanism which extracts the sequential characteristics of the nodes in the network and (ii) a beam search method which excludes path non-connectivity. In [30], for given network states, an unsupervised ML model - k-means - is deployed in the controller and learns the past routing behavior and then predicts the best routing path according to the current network state. Due to the characteristics of unsupervised algorithms, the proposed model can adapt to network topology changes. In [28], first, network status information such as latency, packet counters, and packet drop values are collected to investigate whether the network status is correct. If it is identified as not being correct, then by predicting the future traffic behavior and creating a traffic scheduling algorithm, the congestion states are avoided. Table I summarizes the related work.

Our work is closely related to the path-planning task in TE, and similar to the works in [28]–[30], we also apply path-planning to obtain an optimized network. However, we also include in our calculations the ISP's OPEX, which is ignored by the existing works. In addition, in [29], [30] ML is used for path computation. In our work, paths are already decided and our proposed system only decides which ones get the traffic. Our proposed system is therefore different and complementary to [29], [30]. However, we note that our framework should theoretically be able to utilize a "path computation" application if it was available. The authors in [28], present four possible criteria to optimize traffic, namely: load balancing, QoS guarantee, energy saving and Hybrid IP/SDN. Their work differs from our system in two main ways: (1) they use neither OPEX nor end-users' QoE as possible figure of merit (2) they use heuristics to optimize for one criteria (e.g., load balancing or energy saving, but not both), while we use supervised ML with the distinct advantage of allowing multi-variate optimization (OPEX and QoE simultaneously) and of being able to learn directly from NOC actions in the field.

III. SYSTEM DESIGN

ARE aims to automate NOC operations in terms of recommending actions, or performing actions if the operator decides to make it fully autonomous, to counter network failure, maximize clients' QoE, and minimize the ISP's cost (QoE-OPEX). ARE utilizes the Extreme Gradient Boost (XGB) algorithm as the main action recommender, taking as inputs the SLA measurements collected from an autonomous system (AS) such as router states, QoE metrics collected from AS's end-users, and the predicted network state provided by the Network Analytics module. To compare ARE against the actions that NOC technicians would take, we create some expert rules to mimic the technician's actions and use them as labels for our ML model, as will be shown in Section III-B.

To train ARE, we need a sufficient amount of data. However, we did not find any public network dataset that includes details

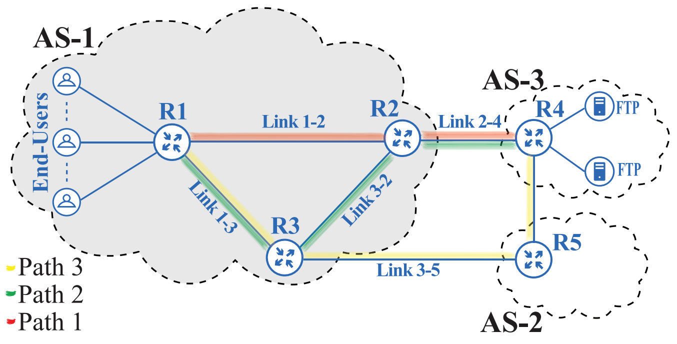


Fig. 2. Network Topology © IEEE reused with permission from [31].

TABLE II
COSTS AND BENEFITS

Each hop	-1	**TE (re-routing)**	-2
Fix Network	-5	**External AS**	-5
Bad QoE	-5	**High QoE**	5
Medium QoE	0		

TABLE III
LAYER-3 POLICIES SUMMARY

Link	Type	Max BW	Member	Policy
Link 1-2	Unique	60	Path1	R2: G0/0 peak 60
Link 2-4	Shared	120	Path1&2	R4: G0/1 peak 120
Link 1-3	Shared	120	Path2&3	R3: G0/0 peak 120
Link 2-3	Unique	60	Path2	R2: G0/1 peak 60
Link 3-5	Unique	60	Path3	R5:G0/0 peak 60

like SLA measurements and QoE levels. As a result, we developed our own testbed to collect the data needed to train our systems. Our testbed is chosen to be simple enough to extract good rules from common sense, yet complex enough to not be able to determine optimal rules trivially. The following subsections will explain in detail the testbed specifications and the data collection method and scenarios.

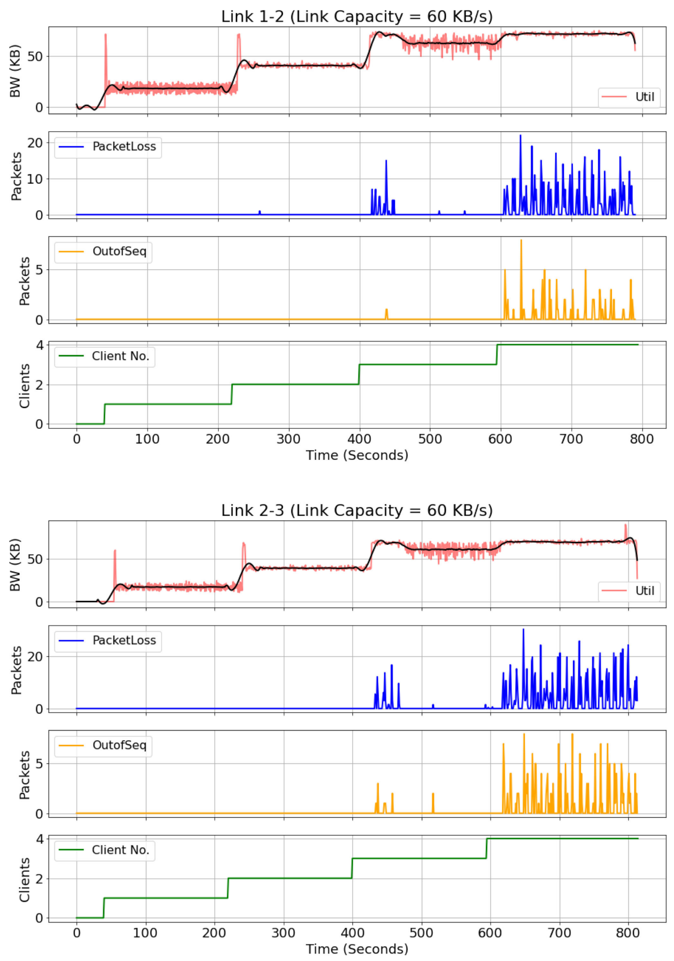
A. Testbed

We use GNS3 [32] to emulate the network in Fig 2. GNS3 is an open-source software used by major organizations like AT&T, Google, and NASA to emulate their systems. This network is chosen to be similar to the Internet's backbone network, where it has multi-paths between end-users and content servers [33]. The network topology of the testbed must be neither too simple such that it doesn't sufficiently reflect the real world nor too complicated such that it's difficult to determine the aforementioned expert rules, which are needed to compare the performance of ARE against that of the expert rules. The topology we use here is a compromise between the two, and consists of 3 Autonomous Systems (AS) representing consumer-side and its ISP (AS-1), an external ISP (AS-2) which is used as backup in case of failure and is considered more expensive, and the service provider (AS-3), which in this case is an FTP file server. Cisco's IOSv images are used to model network devices such as routers and switches. Also, the end-users are modelled as containers running a headless Ubuntu 18.04 OS image. We use containers due to their lightweight property [34]. The content servers are built on the same machine that runs the testbed to ensure eliminating unknown and external network conditions. We placed nine end-users directly on AS-1. To ensure network connectivity, all routers run the OSPF routing protocol. Finally, we created three MPLS/OSPF tunnels and utilized Access Control Lists (ACL) to control the paths between end-users and content servers.

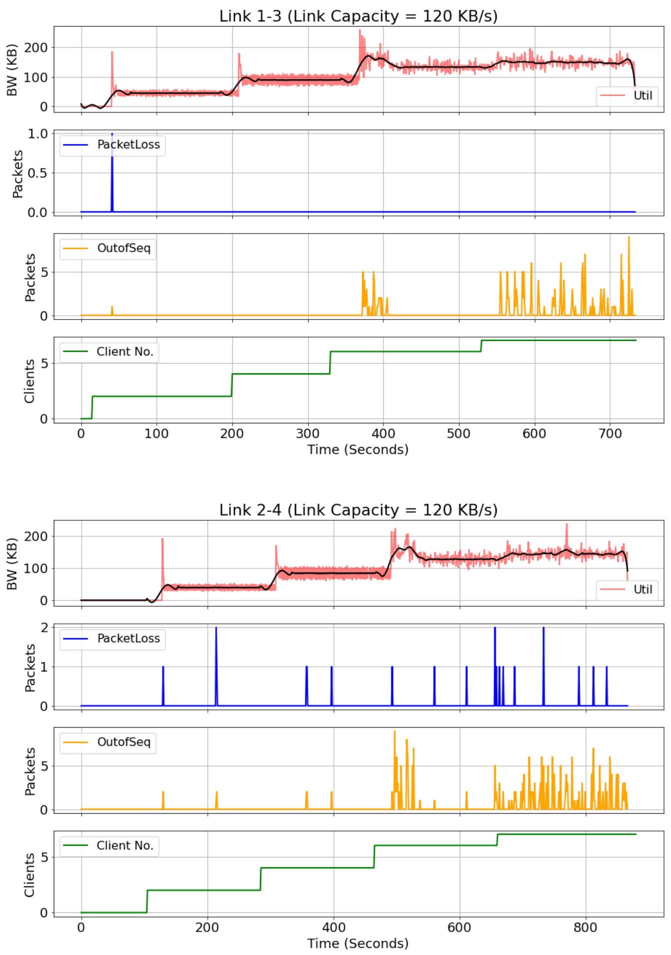
Each end-user has at least three paths to the content servers. PATH_1 and PATH_2 are internal routes within AS-1, and their costs depend on the number of hops the traffic passes through. In contrast, PATH_3 is an external path that forwards the traffic to AS-2 and then to the content servers in AS-3. To match the real situation of an ISP, we implemented a billing system to characterize the cost of each path and each action. Internal paths will cost less compared to the external ones. As a result, PATH_1 is considered to be the optimum path in terms of cost since it is an internal route, and it has the least number of hops; i.e., OPEX $= 2$ IGP hops $\times -1 = -2$. In contrast, PATH_3 is the most expensive path since AS-2 owner charges for allowing AS-1 to use its infrastructure; i.e., OPEX $= 2$ IGP hops $\times -1 + -5$ (External AS) $= -7$. PATH_2 is more expensive than PATH_1 but cheaper than PATH_3 since it is an internal route though with more hops than PATH_1; i.e., OPEX $= 3$ IGP hops $\times -1 = -3$. Also, the system takes into consideration that fixing network issues requires time and money, so OPEX $= -5$. Furthermore, the system either awards or charges the AS based on the end-users' QoE per path. All costs and benefits are summarized in Table II and used in calculating the QoE-OPEX metric explained in Section III-B.

Cisco's IOSv images suffer from bandwidth limitation since their main purpose is to test and analyze recently proposed network algorithms and configurations, which does not require high bandwidth utilization. As a consequence, the maximum bandwidth IOSv images can support is around 2Mb/sec. On the contrary, the end-users in our experiments are transferring files based on the TCP windowing algorithm [35] and a single end-user is expected to fully occupy this limited bandwidth. To alleviate this bandwidth restraint, we limit the maximum transfer speed to 20 KB/sec. Moreover, we applied layer-3 policies to limit MPLS tunnels' bandwidth to support up to three simultaneous end-users, see Table III. Wireshark is used to ensure the effect of the applied policies versus the number of end-users per path [36]. In other words, we want to make sure that links are congested by four clients, and we can verify this in Figure 3(e). In the figure, there are two types of links: unique and shared links. Unique links are a member of only one path. Therefore, the unique links' maximum bandwidth is 60 KB/sec; i.e., bandwidth of three end-users, while a shared link aggregates two or more paths and its maximum bandwidth is double the unique path.

The layer-3 policies help in determining current network states. The normal state is when any link has a maximum of three end-users. Congestion state is when there are more than three end-users connected to a path. Additionally, we defined network issues such as any network device failure or path disconnectivity by a composition of delay and jitter ranging



(a) Unique Links.



(b) Shared Links.

Fig. 3. MPLS Link Utilization © IEEE reused with permission from [31].

between 80-350 ms and 5-60 ms, respectively. To summarize our action space: reroute due to congestion TE-P1→P2, TE-P1→P3, TE-P2→P3, reroute to optimize cost TE-P2→P1, TE-P3→P1, TE-P3→P2, and device issue Fix LR-1-3, Fix LR-2-4, Fix LR-2-1, Fix LR-3-2, Fix LR-3-5.

B. Data Collection and Preparation

We developed an API in Python to interact with the GNS3 emulator to control the operation of the network devices such as routers and docker containers (end-users). The API has the ability to enforce delay and/or jitter on any available link in the network. Also, the API can do live configuration of any network device. We started the data collection by randomly assigning each end-user to a path and we created a problem by randomly choosing between disconnecting a link or intentionally having congestion on a specific link. We measured the network status via the Blue Planet state classifier [31], collected QoS and QoE metrics every 30 seconds, and labelled the network actions via Algorithm 1. The QoS metric consists of (i) end-to-end delay, jitter, packet loss values, etc.; i.e., SLA measurements of the three paths (ii) packet drops, ingress rates, and egress rates values of the router ports in AS-1. File transfer downloading speed is the only QoE metric we gather. Afterwards, QoS and QoE metrics are merged and aligned with the aid of timestamps. Table IV lists all the collected features and labels.

We collected over twenty days of data and a dataset of 56,000 data points. The NOC expert rules utilized in data collection are presented in Algorithm 1. In the algorithm, ARE always tries to route clients' traffic internally before externally. In other words, ARE favors internal routes over external routes since they are cheaper from the OPEX point of view. Because our testbed network is not complex, it was relatively easy for us to define these rules. But, in a more realistic network, these rules become more complicated to define, and this is why NOCs employ expert technicians. The collected dataset has 66 different data features. The labeled action distribution is shown in Figure 5(a). It is clear that the *Do Nothing* action is predominant. This situation stemmed from the strategy developed during the data collection process. The network is forced to reset to the normal state before introducing another abnormal state. Consequently, we encounter the problem of unbalanced classes in ML; i.e., one class including more data points than other classes. Creating an imbalanced dataset with the Normal state having more instances than the other classes is intentional

TABLE IV
FEATURES AND LABELS SUMMARY

Features
QoS features (57 features): • Per-Path-QoS (30 features; 3 Paths × 10 features): – Delay: End-to-end Path delay (min/avg/max). – Jitter: End-to-end Path jitter (min/avg/max). – Packet loss: End-to-end Path packet loss (min/avg/max). – Out of Sequence: In-ordered packets. • Port statistics (27 features; 3 routers × 3 Ports × 3 Features): – Drops: Packet Drop count for ports 0, 1, and 2. – Input Rate: Ingress packet rate for ports 0, 1, and 2. – Output Rate: Egress packet rate for ports 0, 1, and 2. QoE features (9 features;3 Paths × 3 features): • Download speed: FTP transfer speed (min/avg/max).
Labels
Reroute due to congestion: • TE-P1→P2: Reroute clients from path 1 to path 2. • TE-P1→P3: Reroute clients from path 1 to path 3. • TE-P2→P3: Reroute clients from path 2 to path 3. Reroute to optimize cost: • TE-P2→P1: Reroute clients from path 2 to path 1. • TE-P3→P1: Reroute clients from path 3 to path 1. • TE-P3→P2: Reroute clients from path 3 to path 2. Fix device or link issues: • Fix LR-1-3: Fix issues between router 1 and router 3. • Fix LR-2-4: Fix issues between router 2 and router 4. • Fix LR-2-1: Fix issues between router 2 and router 1. • Fix LR-3-2: Fix issues between router 3 and router 2. • Fix LR-3-5: Fix issues between router 3 and router 5.

since Normal is the dominant network condition in reality and problems occur less often than Normal. Figure 5(b) shows the distribution of the recommended actions by the expert rules (which mimic the NOC technicians' decisions) when there is network abnormality. The recommended actions can be mainly classified into either fixing the faulty device by restarting or re-configuring, denoted as *FIX* or rerouting the end-users who are experiencing bad QoE due to this abnormality, denoted as *TE*. The NOC chooses among these actions according to the cost of each action.

C. ARE: Action Recommendation Engine

As mentioned earlier, the dataset collected is imbalanced, introducing two different problems: (1) the ML algorithm tends to be more biased towards the predominant class as an attempt of maximizing its accuracy, and (2) the instances in the minority class are insufficient such that they do not carry adequate information to represent the class. These two situations are challenging because it makes the algorithm either favor the majority class or not learn the distinctive features of the minority class even though it is usually the minority class that we are interested in classifying.

To solve these problems, we can use ensemble methods which are powerful ML models that combine several learning algorithms to boost their ability to generalize and predict. With every new data point, classifiers vote for the classes. After that, the data is assigned to the most voted class [37]. This allows the ensemble model to tackle the problem of efficiently

Algorithm 1: Operational Rules Mimicking NOC Actions for This Experiment

```
Input: Network State
Output: Action
while True do
    if Network State == Congestion then
        if Internal Path Available then
            Action=Reroute Internally
        end
        else
            Action=Reroute Externally
        end
    end
    else if Network State == Network Failure then
        Action=Fix the physical device
    end
    else
        Action=Do Nothing
    end
end
```

classifying minority classes in imbalanced datasets [38], [39]. Therefore, we apply ensemble methods to exploit their ability in imbalanced classification tasks. We tried two ensemble methods: Gradient Boosting [40] and Extreme Gradient Boosting (XGBoost) [41] algorithms.In addition, we also use a Decision Tree [42]. Furthermore, we intentionally biased the three chosen ML models by introducing class weights 1:18 towards the rare classes, to slightly balance them against the majority class. We used the Sklearn *compute_class_weight* utility function, that has been designed specifically for this purpose, to calculate the weights. The results for all 3 are reported later in the paper.

During training the ML models, we found out that the wait period of 30 seconds was not enough for some states, especially congestion states, to be clearly seen. To tackle this issue, we aggregated the dataset by a factor of two. The aggregation is done by taking average, maximum, or minimum of the metric depending on its nature. For example, for RTT, we collect RTT_{min}, RTT_{avg}, and RTT_{max}. For RTT_{min}, we take the minimum of the previous measurement:

$$RTTmin_{agg}[T] = min(RTT_{min}[T-1], RTT_{min}[T])$$

Similarly, for RTT_{max} the aggregation becomes:

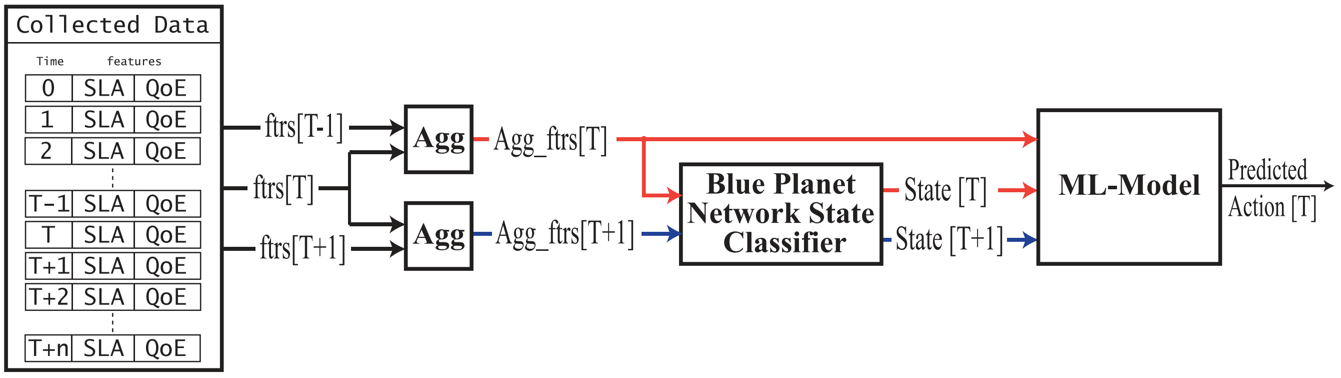
$$RTTmax_{agg}[T] = max(RTT_{max}[T-1], RTT_{max}[T])$$

while for RTT_{avg} the aggregation becomes:

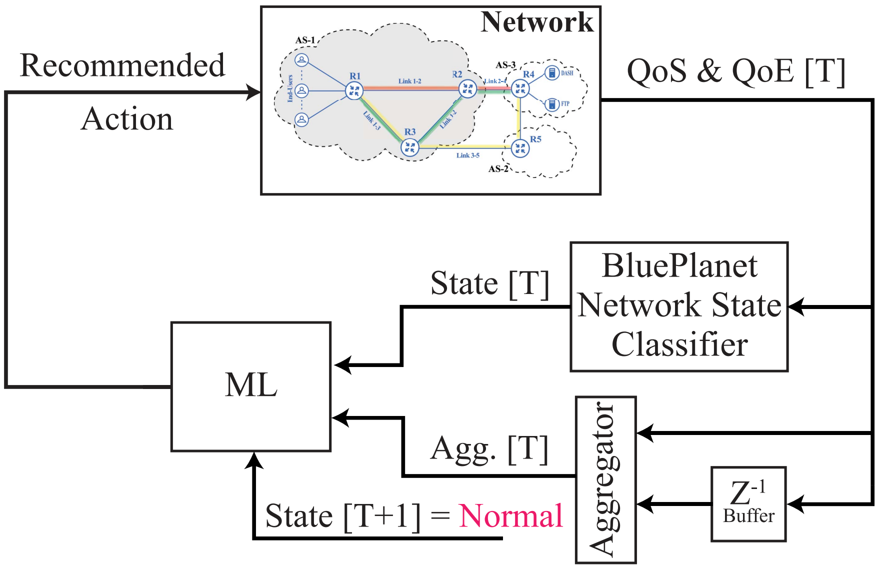
$$RTTavg_{agg}[T] = avg\big(RTT_{avg}[T-1], RTT_{avg}[T]\big)$$

The action is chosen to be the dominant action. In other words, if the two actions are *Do Nothing* and any other action; e.g., reroute, or fix, we chose the other action to be our label.

On the one hand, we fed the ML models with the aggregated features at time [T] as well as the state probabilities resulting from Ciena's BluePlanet's network state classifier at



(a) Training procedure of ARE.



(b) ARE's Closed-loop diagram.

Fig. 4.

time [T] and [T+1]. State probabilities is a list of probabilities for all available states. So the Blue Planet state classifier tells us what is the probability that we are currently in state normal, congestion, and physical network problem. For example, it gives us something like {normal 78%, congestion 13%, physical problem 9%}. On the other hand, the label is the action at time [T]. Thereby, the ML models can predict the root cause action that changed the network state from *state*[*T*] to *state*[*T+1*]. The training procedure is illustrated in Figure 4a.

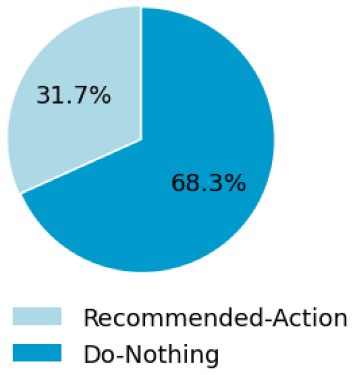
After training, the ML model is integrated with GNS3 to form a live closed-loop system and configured to automatically execute its predicted actions on the network, shown in Figure 4b. At every time [T], SLA and QoE measurements are fed to BluePlanet's state classifier to generate a state probabilities vector. Also, SLA and QoE measurements are aggregated with the measurements generated at time [T−1], represented by Z^{-1} in the figure. Afterwards, the state probabilities vector and the aggregated measurements, at time [T], are fed to the ML model. Also, the state probabilities vector of [T+1] is always set to the Normal state to instruct ARE to use its training and recommend an action that will keep or lead the network into the Normal state. The predicted action is applied automatically to the network via the aforementioned python-based script.

IV. Evaluation and Results

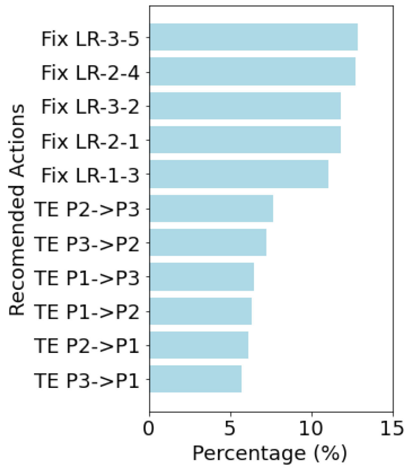
In this section, we present the results such as accuracy, F1-score, and confusion matrix. Moreover, we present the results collected from the live emulation.

A. AI-Based Models

The dataset was randomly partitioned into 75% for training and 25% for testing the models. We also employed 10-fold cross validation on the training dataset. The parameters of the three models and feature importance are listed in Tables V and VI, respectively. To calculate feature importance, we used Sklearn's default *feature_importances* method which is based on Mean Decrease Impurity (MDI) to calculate feature importance for each fold's model and then average them to get



(a) Class Distribution.



(b) Minority Class Breakdown.

Fig. 5. Actions space distribution.

TABLE V
MODELS' PARAMETERS: ALL ARE HYPER-PARAMETERS UNLESS STATED OTHERWISE

DT	Split criterion: gini, Splitter: best, Max depth:19, Class Weight: 'Balanced', Leaves: 149 (learned)
GB	Learning rate: 0.1, Max depth: 3, No. of estimator: 100 Class Weight: 'Balanced'
XGB	Learning rate: 0.3, Max depth: 6, booster: gbtree, No. of estimator: 100, Class Weight: 'Balanced'

Note: For more information about sklearn terminologies, please see [40], [41], and [42] for GB, XGB, and DT, respectively.

TABLE VI
TOP-3 IMPORTANT FEATURES

DT	**GB**	**XGB**
Path1_RTT_Min:29%	State_IsNormal:28%	Path1_Jitter_Avg:35%
Path1_Jitter_Avg:9%	Path2_RTT_Min:4%	State_IsNormal:19%
State_IsNormal:3%	Path1_PacketLoss:7%	Router_Port0_Drop:2%

the Top-3 features. On the test set, classification models DT, GB, and XGB produced the accuracies 99.48%, 99.70%, and 99.87%, respectively. The overall accuracies for the models are high with all of them being greater than 99%. However, these results are tricky since the dataset doesn't include homogeneously distributed classes. Around 68% of the instances belong to one class: Do-Nothing, see Figure 5(a) and 32% belong to the rest of the classes, see Figure 5(b).

TABLE VII
MODELS F1-SCORE

Model	**TE-Action**					
	P1-P2	P3-P2	P1-P3	P2-P3	P2-1	P3-1
DT	0.88	0.94	0.9	0.93	1	0.81
GB	0.910	0.98	0.93	0.92	0.84	0.88
XGB	0.92	1	0.96	0.97	1	0.96
Model	**Fix-Action**					
	DoNoth	**LR-1-3**	**LR-2-4**	**LR-2-1**	**LR-3-2**	**LR-1-3**
DT	1	0.99	0.99	0.97	1	1
GB	1	1	0.99	1	1	1
XGB	1	1	1	1	1	1

To better comprehend the models' efficiencies, we focus on model performances on the minority classes. Table VII presents the F1 scores for each class. It is evident that all the models are able to clearly detect the majority class Do-Nothing with F1 score of almost 100% and any link issues with F1 scores ranging from 0.97 to 1.00. Nevertheless, we observe a performance degradation in detecting rerouting actions, especially the ones related to congestion: TE P1 → P2, TE P1 → P3, and TE P2 → P3. In addition, the lowest f1 scores belong to the 2 classes with the least instances: TE P2 → P1 with GB having 0.84 and TE P3 → P1 with DT 0.81 and GB 0.88 f1 scores.

Moreover, DT yields almost always the lowest scores, with the exception of rerouting actions TE P2 → P3 for congestion and TE P2 → P1 for cost optimization policy where GB yield lower results than DT. On the other hand, XGB and DL either produce the same or better results when compared to other models, even with the two actions related to the cost optimization policies where DT and GB struggled.

However, when we examine the confusion matrices closely, we see a dissimilar trend to the F1 scores, see Figure 6. Compared to the other models, XGB still performed the best with the exception of action TE P1 → TE P2 where GB performed slightly better, and action TE P3 → P1 where DT yielded the best outcome with significant improvement compared to XGB. Additionally, XGB only confused the ground truth action with Do-Nothing while DT and GB experienced confusion with other action classes. Furthermore, DT and GB produced comparable results. For instance; GB yielded better results for the following actions: Fix LR-1-3, Fix LR-2-1, TE P1 → P2, and TE P3 → P2, whereas DT performed well for these actions: Fix LR-3-5, TE P1 → P3, TE P2 → P3, TE P2 → P1, and TE P3 → P1.

B. Live OPEX Performance Comparison

To further asses our work, we employed a live emulation that is quite similar to real-life situations faced by NOCs. The emulation employs the incremental QoE-OPEX as the main metric of comparison. We compare ARE with the *NOC* (expert rules) taking into account OPEX, *static* that represents a network not managed by any person or algorithm, and an *Anchor* method which represents existing works in that it is a congestion-aware traffic engineering algorithm without taking into consideration OPEX. To implement the Anchor method, we used the same expert rules but considered the cost of all paths to be equal, in essence not taking into account OPEX.

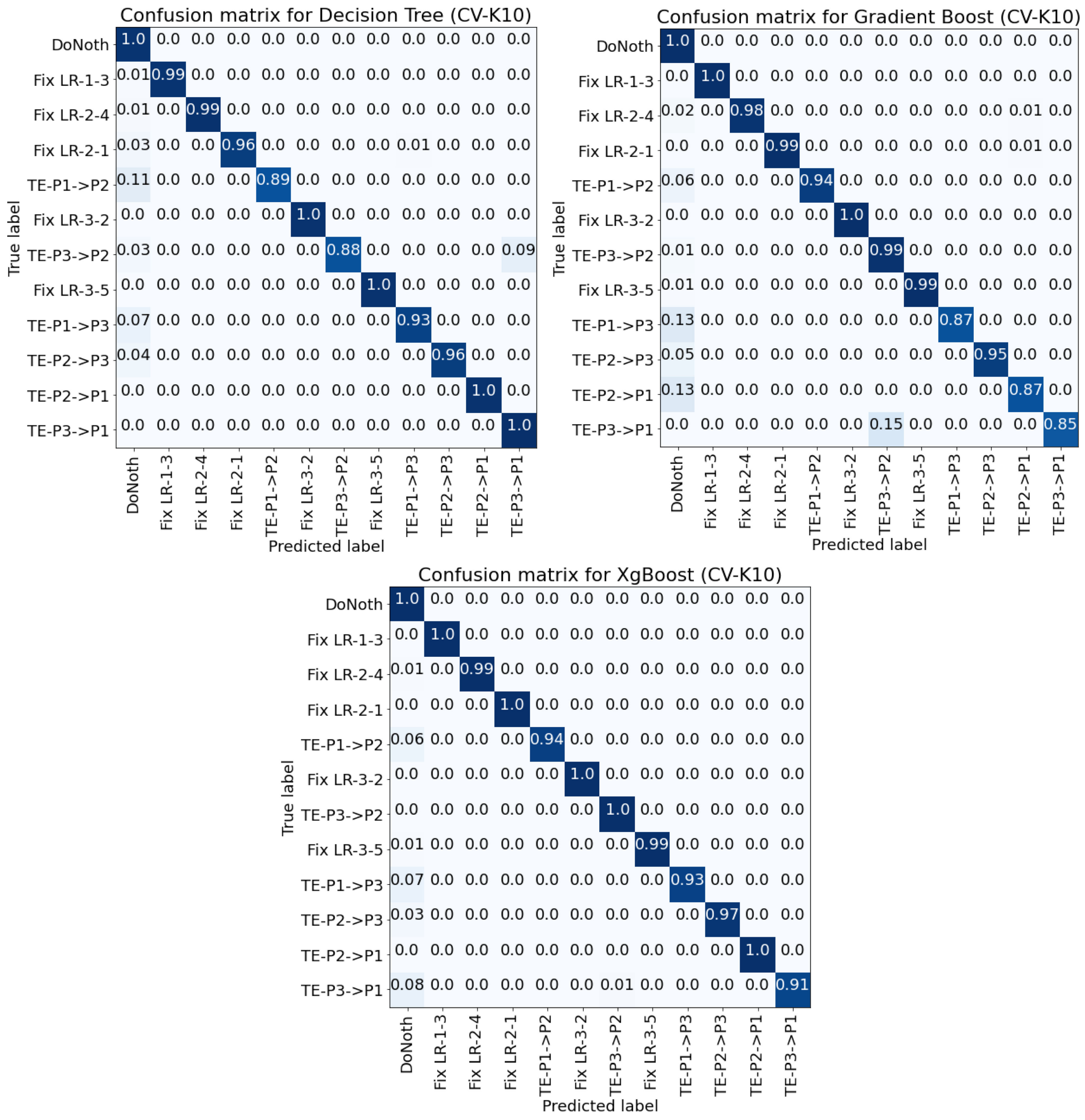


Fig. 6. Confusion Matrices for DT, GB, and XGB.

We did not compare against deep learning methods such as [29] because (1) their application is different, as explained in Section II, and (2) they use deep learning, and if we try to apply their algorithm to our dataset, which is relatively small compered to theirs, their performance will be low because deep learning models are data hungry and typically need millions of data points to achieve their potential. So, running their algorithms on our dataset is clearly unfair to them. We utilized a scenario similar to the one we used in generating the training dataset, but we randomized the operation of the nine clients and paths they use. We ran this experiment for more than 1200 steps; i.e., 10 hours. Figure 7 shows the cumulative QoE-OPEX over 1200 iterations. We can see that the static network is outperformed by all other methods. The second-worst performing method is the Anchor method, which is expected since it does not consider OPEX. Among the remaining methods, the XGB model closely follows the expert rules, proving that the automated recommendations by ARE are valid. The only time expert rules lead to noticeably better results than ARE is at approximately iteration 770. When we checked that instance, we noticed that the network state classifier has incorrectly classified the network state as "Normal". Since this state

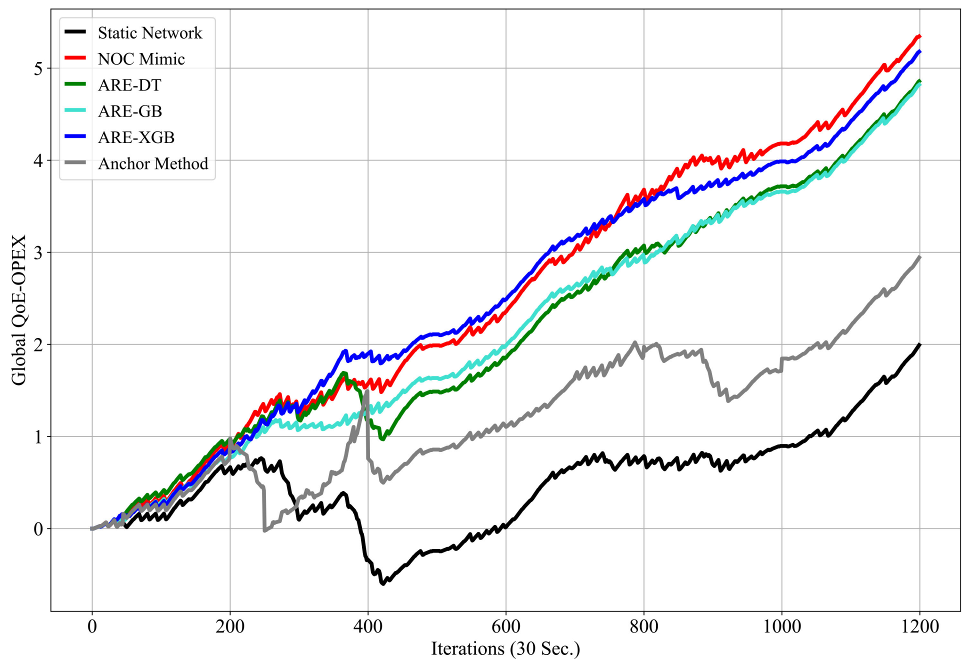


Fig. 7. Cumulative QoE-OPEX comparison between ARE models, static network, Anchor method, and NOC rules.

is one input to ARE, it misled ARE into taking an action (namely "Do-Nothing") that was not the best. It should be noted that in the real world, NOC technicians might also be misled by the data and/or make mistakes. Also, NOC technicians have the ability to override the ARE recommendations. But overall, what the figure shows is that letting ARE take autonomous actions will lead to results that are quite comparable to the manual decisions taken by NOC technicians, despite ARE's mistakes, and this is a positive step towards NOC automation.

V. Conclusion and Future Work

We introduced the first ML-based action recommender system for NOCs. The system, called ARE, can recommend actions for failure management and OPEX-aware traffic engineering. These tasks are usually performed manually by expert NOC technicians in complex cases, or through automated rules in simple scenarios. ARE automates the selection of the best action to take for both simple and complex scenarios. Performance evaluations showed that the actions automatically recommended by ARE lead to results that closely follow, and sometime exceed, those of the NOC technicians. In that sense, ARE moves us one step closer towards autonomous NOCs.

For future work, we plan to address the following challenges.

1) ML model scalability: The model needs to be adjusted every time the ISP's network expands. We plan to tackle this challenge by examining the feasibility of applying Graphical Neural Networks (GNN) which are a class of deep learning methods designed to perform inference on data described by graphs/networks. In GNN, the network is decomposed according to each node (router) location and connectivity with others.
2) Lab to production: Similar to any AI model, ARE is prone to working in a real environment where the data might be somewhat different from ARE's trained-with data. To mitigate this, we plan to apply the ML-Ops workflow to deploy ARE to a real-world environment and trigger automatic transfer learning, or even retrain in case of noticeable performance degradation.
3) Superhuman performance: We showed that ARE can automate what NOCs do manually nowadays. This by itself is an important finding and contribution for moving closer towards autonomous/self-healing networks. But, it also gives rise to the question that if supervised learning can mimic the performance of NOC operators, can more advanced machine learning methods such as Reinforcement Learning outperform humans, as they do on computer gaming? If so, that could potentially revolutionize the field of network management and operation.
4) Explainable AI: For this work, we did not enter the realm of explainable AI, because our system is not a human life-critical system like a medical or military system or self-driving cars. But explaining the path the model takes in order to derive a decision can be useful and subject to future work.

Acknowledgment

The technology described in this paper is part of U.S. patent application number 16892594 titled "Action Recommendation Engine (ARE) of a closed-loop Machine Learning (ML) system for controlling a telecommunications network" filed June 4 2020.

References

[1] A. Malik, B. Aziz, M. Adda, and C.-H. Ke, "Smart routing: Towards proactive fault handling of software-defined networks," *Comput. Netw.*, vol. 170, Apr. 2020, Art. no. 107104.

[2] R. Blum and J. Kaplan, *Network Operations Centers*, Lucent Technol., Murray Hill, NJ, USA, Mar. 2004. [Online]. Available: http://penta2.ufrgs.br/tutorials/qos/noc01.pdf

[3] *Cienaś Unified Assurance and Analytics*. Accessed: Nov. 15, 2020. [Online]. Available: https://www.blueplanet.com/products/uaa.html

[4] *Cienaś Route Optimization and Analysis*. Accessed: Nov. 15, 2020. [Online]. Available: https://www.blueplanet.com/products/route-optimization.html

[5] E. D. Osborne and A. Simha, *Traffic Engineering With MPLS*. Indianapolis, IN, USA: Cisco Press, 2003.

[6] P. Gill, N. Jain, and N. Nagappan, "Understanding network failures in data centers: Measurement, analysis, and implications," in *Proc. ACM SIGCOMM Conf.*, 2011, pp. 350–361.

[7] B. Raeisi and A. Giorgetti, "Software-based fast failure recovery in load balanced sdn-based datacenter networks," in *Proc. IEEE 6th Int. Conf. Inf. Commun. Manag. (ICICM)*, 2016, pp. 95–99.

[8] D. Côté, "Using machine learning in communication networks," *J. Opt. Commun. Netw.*, vol. 10, no. 10, pp. D100–D109, 2018.

[9] *Cisco MPLS*. Accessed: Nov. 15, 2020. [Online]. Available: https://www.cisco.com/c/en/us/products/ios-nx-os-software/multiprotocol-label-switching-mpls/index.html

[10] *Juniper MPLS*. Accessed: Nov. 15, 2020. [Online]. Available: https://www.juniper.net/documentation/en_US/junos /topics/concept/mpls-security-overview.html

[11] G. Saldamli *et al.*, "Improving link failure recovery and congestion control in SDNs," in *Proc. 10th Int. Conf. Inf. Commun. Syst. (ICICS)*, 2019, pp. 30–35.

[12] G. Chen *et al.*, "FUSO: Fast multi-path loss recovery for data center networks," *IEEE/ACM Trans. Netw.*, vol. 26, no. 3, pp. 1376–1389, Jun. 2018.

[13] Y. Liu, H. Gu, K. Wang, X. Yu, and Y. Wang, "An adaptive failure recovery mechanism based on asymmetric routing for data center networks," *J. Supercomput.*, vol. 77, pp. 2103–2123, Jun. 2020.

[14] Y. Peng, X. Gong, L. Guo, and D. Kong, "A survivability routing mechanism in SDN enabled wireless mesh networks: Design and evaluation," *China Commun.*, vol. 13, no. 7, pp. 32–38, 2016.

[15] S. Wang, H. Xu, L. Huang, X. Yang, and J. Liu, "Fast recovery for single link failure with segment routing in SDNs," in *Proc. IEEE 21st Int. Conf. High Perform. Comput. Commun. IEEE 17th Int. Conf. Smart City IEEE 5th Int. Conf. Data Sci. Syst. (HPCC/SmartCity/DSS)*, 2019, pp. 2013–2018.

[16] Q. Li, Y. Liu, Z. Zhu, H. Li, and Y. Jiang, "BOND: Flexible failure recovery in software defined networks," *Comput. Netw.*, vol. 149, pp. 1–12, Feb. 2019.

[17] A. Sgambelluri, A. Giorgetti, F. Cugini, F. Paolucci, and P. Castoldi, "OpenFlow-based segment protection in Ethernet networks," *J. Opt. Commun. Netw.*, vol. 5, no. 9, pp. 1066–1075, 2013.

[18] R. Kanagavelu and Y. Zhu, "A pro-active and adaptive mechanism for fast failure recovery in SDN data centers," in *Proc. Future Inf. Commun. Conf.*, 2018, pp. 239–257.

[19] G. Han, Y. Dong, H. Guo, L. Shu, and D. Wu, "Cross-layer optimized routing in wireless sensor networks with duty cycle and energy harvesting," *Wireless Commun. Mobile Comput.*, vol. 15, no. 16, pp. 1957–1981, 2015.

[20] Q. Dong, J. Li, Y. Ma, and S. Han, "A path allocation method based on source routing in SDN traffic engineering," in *Proc. IEEE Int. Conf. Smart Cloud (SmartCloud)*, 2019, pp. 163–168.

[21] F. Geyer and G. Carle, "Learning and generating distributed routing protocols using graph-based deep learning," in *Proc. Workshop Big Data Anal. Mach. Learn. Data Commun. Netw.*, 2018, pp. 40–45.

[22] B. Mao *et al.*, "A tensor based deep learning technique for intelligent packet routing," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, 2017, pp. 1–6.

[23] B. Mao *et al.*, "Routing or computing? The paradigm shift towards intelligent computer network packet transmission based on deep learning," *IEEE Trans. Comput.*, vol. 66, no. 11, pp. 1946–1960, Nov. 2017.

[24] J. Xiao, S. Chen, and M. Sui, "The strategy of path determination and traffic scheduling in private campus networks based on SDN," *Peer-to-Peer Netw. Appl.*, vol. 12, no. 2, pp. 430–439, 2019.

[25] D. Li, Y. Shang, and C. Chen, "Software defined green data center network with exclusive routing," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, 2014, pp. 1743–1751.

[26] A. Amokrane, R. Langar, R. Boutabayz, and G. Pujolle, "Online flow-based energy efficient management in wireless mesh networks," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, 2013, pp. 329–335.

[27] Q. Li, J. Cui, H. Zhong, Y. Du, Y. Luo, and L. Liu, "LBBESA: An efficient software-defined networking load-balancing scheme based on elevator scheduling algorithm," *Concurrency Comput. Pract. Exp.*, vol. 32, no. 16, 2019, Art. no. e5222.

[28] Z. Shu *et al.*, "Traffic engineering in software-defined networking: Measurement and management," *IEEE Access*, vol. 4, pp. 3246–3256, 2016.

[29] Y. Zuo, Y. Wu, G. Min, and L. Cui, "Learning-based network path planning for traffic engineering," *Future Gener. Comput. Syst.*, vol. 92, pp. 59–67, Mar. 2019.

[30] S. Kumar, G. Bansal, and V. S. Shekhawat, "A machine learning approach for traffic flow provisioning in software defined networks," in *Proc. IEEE Int. Conf. Inf. Netw. (ICOIN)*, 2020, pp. 602–607.

[31] A. R. Mohammed, S. A. Mohammed, D. Côté, and S. Shirmohammadi, "Machine learning based network status detection and fault localization," in *Proc. IEEE Int. Symp. Meas. Netw. (M&N)*, 2019. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8805044

[32] *GNs3*. Accessed: Nov. 15, 2020. [Online]. Available: https://www.gns3.com/

[33] M. Yuksel, K. Ramakrishnan, and R. D. Doverspike, "Cross-layer failure restoration techniques for a robust IPTV service," in *Proc. 16th IEEE Workshop Local Metropolitan Area Netw.*, 2008, pp. 49–54.

[34] D. Merkel, "DOCKER: Lightweight Linux containers for consistent development and deployment," *Linux J.*, vol. 2014, no. 239, p. 2, 2014.

[35] N. Patel and R. Patel, "Performance analysis of network with different queuing mechanisms in TCP/FTP and UDP/FTP scenario," in *Smart Systems and IoT: Innovations in Computing*. Heidelberg, Germany: Springer, 2020, pp. 13–21.

[36] *Wireshark, Go Deep*. Accessed: Nov. 15, 2020. [Online]. Available: https://www.wireshark.org/

[37] J. Zerouaoui, "Using ensemble methods to solve the problem of pulsar search," in *Big Data and Networks Technologies*. Cham, Switzerland: Springer, 2019, p. 183.

[38] T. M. Khoshgoftaar, A. Fazelpour, D. J. Dittman, and A. Napolitano, "Ensemble vs. data sampling: Which option is best suited to improve classification performance of imbalanced bioinformatics data?" in *Proc. IEEE 27th Int. Conf. Tools Artif. Intell. (ICTAI)*, 2015, pp. 705–712.

[39] M. Galar, A. Fernandez, E. Barrenechea, H. Bustince, and F. Herrera, "A review on ensembles for the class imbalance problem: Bagging-, boosting-, and hybrid-based approaches," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 42, no. 4, pp. 463–484, Jun. 2012.

[40] J. H. Friedman, "Stochastic gradient boosting," *Comput. Stat. Data Anal.*, vol. 38, no. 4, pp. 367–378, 2002.

[41] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Disc. Data Min.*, 2016, pp. 785–794.

[42] J. Quinlan, "Induction of decision trees," *Mach. Learn.*, vol. 1, pp. 81–106, Mar. 1986.

[43] N. Kitsuwan, S. McGettrick, F. Slyne, D. B. Payne, and M. Ruffini, "Independent transient plane design for protection in OpenFlow-based networks," *IEEE/OSA J. Opt. Commun. Netw.*, vol. 7, no. 4, pp. 264–275, Apr. 2015.

[44] H. Mliki, L. Kamoun, and L. Chaari, "Ethernet congestion manager characteristics, calibration and analysis," in *Proc. 2nd Int. Conf. Commun. Netw.*, 2010, pp. 1–8.

[45] O. Tilmans and S. Vissicchio, "IGP-as-a-backup for robust SDN networks," in *Proc. IEEE 10th Int. Conf. Netw. Service Manag. (CNSM) Workshop*, 2014, pp. 127–135.

[46] Y.-D. Lin, H.-Y. Teng, C.-R. Hsu, C.-C. Liao, and Y.-C. Lai, "Fast failover and switchover for link failures and congestion in software defined networks," in *Proc. IEEE Int. Conf. Commun. (ICC)*, 2016, pp. 1–6.

[47] S. Jain *et al.*, "B4: Experience with a globally-deployed software defined WAN," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 3–14, 2013.

[48] X. Wu *et al.*, "NetPilot: Automating datacenter network failure mitigation," in *Proc. ACM SIGCOMM Conf. Appl. Technol. Architect. Protocols Comput. Commun.*, 2012, pp. 419–430.

[49] C.-Y. Hong *et al.*, "Achieving high utilization with software-driven WAN," in *Proc. ACM SIGCOMM Conf. SIGCOMM*, 2013, pp. 15–26.

[50] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "Enabling fast failure recovery in OpenFlow networks," in *Proc. 8th Int. Workshop Design Rel. Commun. Netw. (DRCN)*, 2011, pp. 164–171.

Shady A. Mohammed received the M.Sc. degree in electronics and computer engineering from Istanbul Sehir University, Istanbul, Turkey. He is currently pursuing the Ph.D. degree in electrical and computer engineering with the University of Ottawa, Ottawa, ON, Canada.

He is currently working as a Machine Learning Developer with Transport Canada. Prior joining Transport Canada, he collaborated with Ciena Corporation, on Applied AI for network analytics and automation. His research interests include machine learning operations, big data, cloud computing, and multimedia systems & communication.

Ayşe Rumeysa Mohammed received the B.Sc. degree in computational and medical physics from Bogazici University, Istanbul, Turkey, in 2014, and the M.Sc. degree in data science from Istanbul Sehir University, Istanbul, in 2017. She is currently pursuing the Ph.D. degree in electrical and computer engineering with the University of Ottawa, Ottawa, ON, Canada.

She collaborated with Ciena Corporation, Ottawa Branch, on Applied AI for network traffic rehabilitation. Her research interests include computer networks measurement, multimedia systems and communication, and computer vision.

David Côté received the Ph.D. degree in physics from Université de Montréal in 2007. He is currently the Chief Data Scientist with the Blue Planet Division, Ciena and a Co-Founder of the Blue Planet Analytics Program with Ciena. His group has developed and productized several machine learning applications recognized by industry innovation awards and commercial success. He also participates in several research projects with industrial and academic partners and holds numerous patents and publications in the field. Prior to joining Ciena, he worked as a Particle Physicist for 14 years with the Stanford Linear Accelerator Center, USA, and CERN, Switzerland, notably, where he gained substantial hands-on experience with big data engineering, data science, and world-class research.

Shervin Shirmohammadi (Fellow, IEEE) received the Ph.D. degree in electrical engineering from the University of Ottawa, Canada, in 2000.

He is currently a Professor with the School of Electrical Engineering and Computer Science, University of Ottawa. He is the Director of the DISCOVER Lab, doing research in measurement methods and Applied AI for multimedia and networking systems. The results of his research, funded by more than $26 million from public and private sectors, have led to 400 publications, three Best Paper awards, over 70 researchers trained at the postdoctoral, Ph.D., and master's levels, 30 patents and technology transfers to the private sector, and a number of awards.

Prof. Shirmohammadi is the Editor-in-Chief of the IEEE TRANSACTIONS ON INSTRUMENTATION AND MEASUREMENT, and was also the Associate Editor-in-Chief of *IEEE Instrumentation and Measurement Magazine* in 2014 and 2015, and is currently on its editorial board. He has been an IEEE Instrumentation and Measurement Society AdCom Member since 2014, served as the Vice President of its Membership Development Committee from 2014 to 2017, and was a member of the IEEE I2MTC Board of Directors from 2014 to 2016. He is an IEEE Fellow for contributions to multimedia systems and network measurements, winner of the 2019 George S. Glinski Award for Excellence in Research, a Senior Member of the ACM, a University of Ottawa Gold Medalist, and a licensed Professional Engineer in Ontario.