Machine Learning-Based Network Status Detection and Fault Localization

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

***Abstract*— Although the autonomous detection of network status and localization of network faults can be a valuable tool for network and service operators, very few works have investigated this subject. As a result in today's networks, fault detection and localization remains a mostly manual process. In this article, we propose a machine learning (ML) method that can automatically detect the status of a network and localize faults. Our method uses the decision tree, gradient boosting (GB), and extreme GB ML algorithms to detect the network status as normal, congestion, and network fault. In comparison, existing related work can at best classify the network status as faulty or nonfaulty. Experimental results show that our method yields accuracies of up to 99% on a dataset collected through an emulated network.**

***Index Terms*— Fault localization, imbalanced dataset, machine learning (ML), network automation, network measurement.**

I. Introduction

SERVICE reliability is important for network operators and internet service providers (ISPs), because there may be significant penalty costs associated with breaches of service level agreements (SLAs). Traditionally, ISPs have been performing service assurance and maintenance manually; i.e., if a part of a network or service fails; e.g., an alarm is triggered, a team of the Network Operation Center (NOC) technicians, who are typically on call 24/7, are alerted to identify, localize, and fix the problem [1]. This reactive process is expensive and takes time, especially if the failure is on a physical device and involves an in-person service call. Meanwhile, the quality of service and the customer experience suffer.

It would, therefore, be a significant contribution to create a system that would autonomously discover and localize potential network faults as early as possible, minimizing their negative impact on the customers' experience. Today's networks are massive interconnections of user devices, technologies, and applications, transporting petabytes of data every millisecond. This vast amount of data creates engineering challenges that can be addressed with modern big data techniques. At the same time, the vast amount of data also provides opportunities: if we could tap into this goldmine of information with the right tool, we could potentially detect and localize network faults with good precision.

Today, this tool has arrived in the form of machine learning (ML). Many of the network state classification and fault localization challenges are essentially big data problems that are difficult to solve analytically, especially for large and/or complex networks, but can be solved more efficiently by ML algorithms, improving the system performance while maintaining relative design simplicity [2], [3]. Tools like Ciena's Blue Planet Analytics (BPA) [4] are examples of the industry moving toward this direction. These tools use ML to help service providers and network operators gain deep insights into the network so that they can make intelligent data-driven business decisions, which lead to improved efficiency, lowered costs, and providing more personalized services. Such tools typically collect, normalize, and categorize a huge amount of data in near real-time and store it in big data clusters sitting in the cloud or on the premises. The data are then fed into applications that leverage ML to generate valuable insights. For example, BPA's network health predictor [5] can analyze historical and current data and employs predictive analysis to constantly assess the probability of each network element going into an abnormal state. If it identifies that a part on an end-to-end service path is at high risk, policy rules take effect. For instance, a policy rule would be that if there is greater than 80% risk of failure for a high-value customer and then adjust the service path. An orchestrator, such as the Blue Planet Multi-Domain Service Orchestration [6], can then take action based on this policy and, for example, provision a new service path or activate a backup system and fill a trouble ticket so the faulty equipment is automatically decommissioned. In essence, the network can be programed to self-monitor and self-heal.

In this article,[1] we propose an ML-based network status detection and fault localization system for general core IP networks with a specific focus on ISPs. The detector differentiates between three main network states: "normal," "congestion," and "network physical issue." In the latter

Manuscript received March 7, 2021; revised May 28, 2021; accepted June 19, 2021. Date of publication July 5, 2021; date of current version July 22, 2021. This work was supported in part by MITACS Accelerate Cluster under Grant IT12571 and in part by Ciena Corporation. The Associate Editor coordinating the review process was Dr. Datong Liu. *(Corresponding author: Shady A. Mohammed.)*

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

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

Digital Object Identifier 10.1109/TIM.2021.3094223

[1]The technology described in this article is part of the U.S. patent application number 16892594 titled "Action Recommendation Engine of a closed-loop ML system for controlling a telecommunications network" filed on June 4, 2020.

two cases, our system also localizes the fault, in essence distinguishing among eight classes in our testbed. We employ GNS3 to emulate a topology close to an ISP's real network to collect and build our dataset. We then evaluate the performance of four ML methods of decision tree (DT), gradient boosting (GB), extreme GB (XGB), and neural network (NN) as network state classifier with fault localization. Network state detection is a new research topic and we found only three other works [7]–[9] in this topic. To the best of our knowledge and unlike those three works that provide binary classification (e.g., normal or fault and congested or not congested), our work is the first one that provides multiclass classification. In addition, compared with the 92% accuracy of the only work out of the three that also localizes faults [9], we reach accuracies of 97%–99%.

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

II. Related Work

Network state detection can be considered a subset of the more general traffic classification field. To this end, ML-based traffic classification has been applied extensively against cyber attacks by diagnosing malicious traffic [10]–[14] and classifying encrypted traffic [15]–[19]. ML algorithms have also been used for comprehending the traffic flow [20], providing application-aware traffic classification [21], and classifying the network traffic via semisupervised learning [15] or supervised learning [22] methods. It has also been suggested in [23] that using especially collected network datasets, one could train supervised ML to identify different root causes for problems based on different raw performance monitoring patterns.

Despite the above-mentioned research, we could only find three other works covering ML-based classification of network status for IP networks [7]–[9], indicating that this topic is understudied despite its tremendous impact potential. Kumar *et al.* [7] investigated DT, support vector machine (SVM), and K-nearest neighbor (KNN) ML algorithms for classifying software-defined wireless mesh-network status as congested and noncongested. Their tests on two datasets with different observation time periods ranging from 5 s to 60 s shows accuracies of 90%–98%, with KNN achieving the best performance on both datasets with average accuracies of 98.5% and 97.5%. In [8], a naïve Bayesian classifier is used to categorize the network status as faulty or normal, while Gupta *et al.* [9] developed a hybrid framework of ML and DL to classify the network status for network function visualization (NFV) systems. In addition, it decides if the problem has already occurred or is predicted to happen with a certain probability. The framework works in two steps. First, it identifies if the network is faulty or not. Then, for a faulty network state, the framework localizes the fault and identifies the root cause, while for the predicted faults it localizes them and determines the severity. Their SVM method achieved 95.4% accuracy in the detection of the network status as faulty or normal, and 89.7% and 95.1% in classifying the faults as manifested or impending, respectively. Manifested problems are identified by multiclass classification with SVM. Localization of impending faults and prediction of their severity is done by stacked sparse autoencoder reaching 92% accuracy.

It is worth mentioning that Abbasi *et al.* [24] present a survey of deep learning (DL) methods for network traffic monitoring and analysis, which addresses, among many things, fault management. However, most of the presented works target radio access networks and cyber-physical networks and are, therefore, not applicable to our IP core network scenario.

In terms of non-ML-based network status classification, we could also only find two articles [25], [26]. Park and Kim [25] proposed an analytical approach in which the deviant RTT values are counted in a late time window to classify the traffic status into five classes indicating stability in the network: stable_strong, stable_weak, stable_transient, stable_biased, and stable_not. In [26], statistical information regarding present topology connection condition, port statuses, different types of packet counters, packet drop counters, link bandwidths, and latency values are acquired by the SDN controller to diagnose the network status as correct or faulty.

All the previously mentioned works classify the network status as faulty or nonfaulty. This binary classification combines all the issues that may happen to the network into one class. Our work aims to detect three high-level states: *normal* state; *network physical issue*, and *congestion*. The *normal* state is when the network is problem free. The *network physical issue* state represents any physical problem that can happen to the network, such as device (e.g., routers) failure and/or link disconnections. The *congestion* state represents traffic flood in any of the network paths that lead to QoE deterioration. In addition, the congestion and network physical issue states are further divided into substates so that our system identifies the exact faulty device or the congested link. This means that our system is a multiclass classifier, specifically eight classes for our testbed. Furthermore, we differ from existing works in terms of performances. We use ensemble algorithms to classify the network status achieving overall accuracies of 97%–99%.

Finally, in the specific field of instrumentation and measurement (I&M), network state detection is a new topic with no existing literature. Network measurement has been an I&M topic of interest for many years and includes, for example, measuring network delay [27]–[29], detecting faulty sensors in wireless sensor networks [30], [31], and detecting changes in the available bandwidth in video delivery networks [32]. But no I&M work has addressed network state detection and fault localization in general.

III. System Design

The design of our system is shown in Fig. 1. The system's inputs are QoS and QoE data metrics collected from our testbed. The QoS metrics consist of per-path and per-device metrics. For per-path metrics, we collect the end-to-end delay, jitter, packet losses, out-of-sequence cases, and discarded samples; i.e., SLA measurements of each of the three paths. For per-device metrics, we collect the ports' input and output packet rates and packet losses for each router in AS-1.

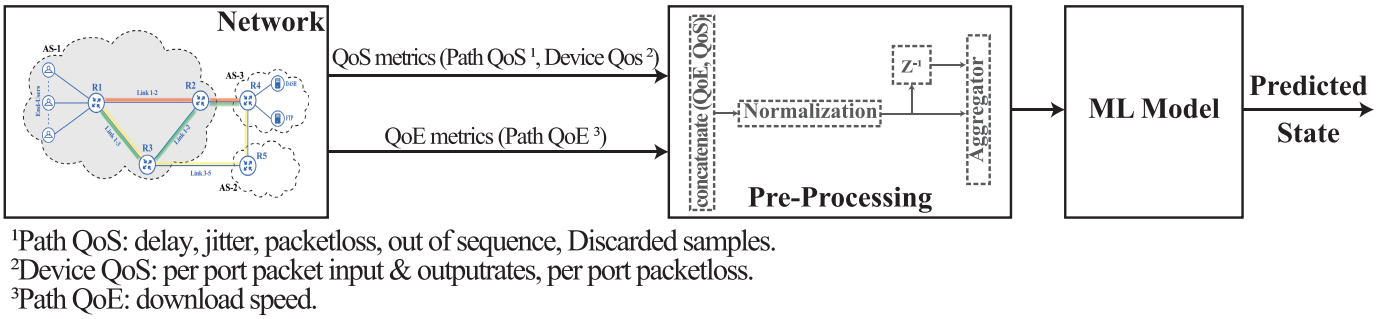

Fig. 1. System design.

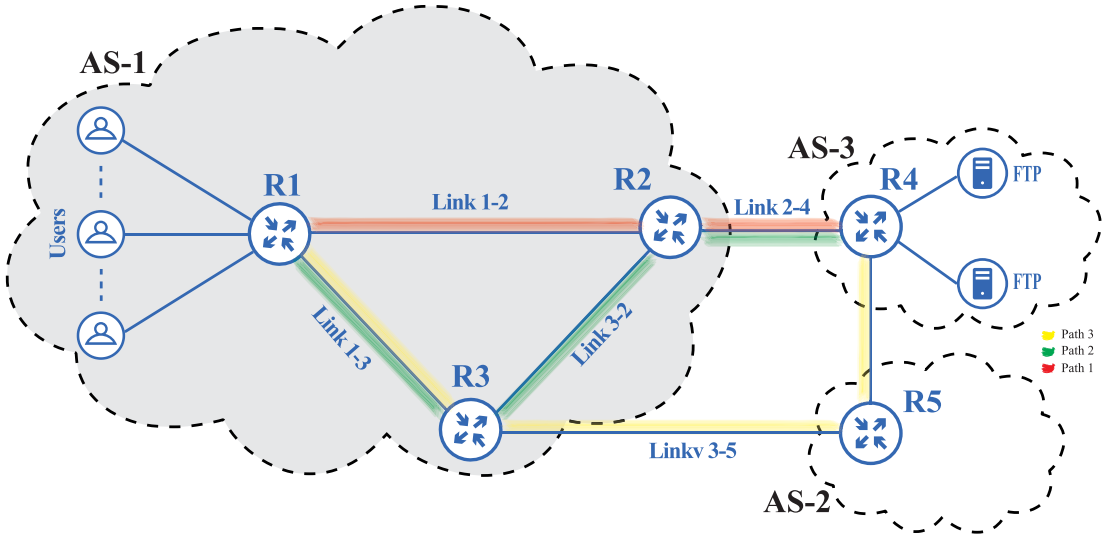

Fig. 2. Network topology.

Finally, QoE is indicated by the downloading speed in the file transfer application.

The input data pass by a preprocessing stage, including concatenation, normalization, and aggregation, before being split into three subportions for training, validation, and testing. Then, our ML models get trained via the training dataset and validated by the validation and test datasets. Finally, the trained models are able to predict the network state given new inputs.

To train the ML models, we need a sufficient amount of data. However, we could not find any open-source network dataset with the inputs our system needs. Therefore, we developed our own testbed to collect the needed data. We chose our testbed to be simple enough so that we could extract the ground truth manually, yet complex enough to not be able to determine optimal rules trivially. Sections III-A–III-C will explain in detail the testbed specifications and the data collection method and scenarios.

A. Testbed

We employ GNS3 [33] to emulate the network in Fig 2. This network is in a similar setup as the Internet's backbone network since it has multipaths between users and content servers [34]. In our case, the network contains three Autonomous systems (AS). AS-1 is the ISP network serving the users. AS-3 is where content servers such as FTP servers exist. We can see that AS-1 and AS-3 are connected through router R2. AS-2 symbolizes a neighboring ISP, where forwarding packets to it is considered to cost more compared to routing the packets internally within AS-1. AS-2 can be used as a backup in case Router R2 goes down or is congested. Our interest is in detecting AS-1's network status.

Cisco's IOSv images are used to model network devices, such as routers and switches. Moreover, we utilized containers running headless Ubuntu 18.04 OS images to model the traffic generated by users of the network. Containers are used due to their lightweight property [35]. In order to eliminate any unknown and external network conditions, the content servers are built on the same machine that runs the testbed. Furthermore, we placed nine users directly on AS-1. To ensure network connectivity, all routers run the open shortest path first (OSPF) routing protocol. Finally, we created three multiprotocol label switching (MPLS)/OSPF tunnels and utilized access control lists to control the paths between users and content servers.

Each user has three paths to the content servers. $PATH_1$ and $PATH_2$ are internal routes, whereas $PATH_3$ is an external path that forwards the traffic from AS-3 to AS-2 and then to AS-1.

Cisco's IOSv images can support around 2 Mb/s. To mitigate this bandwidth restraint, we apply layer-3 policies, as shown in Table I, to limit MPLS tunnels' bandwidth to

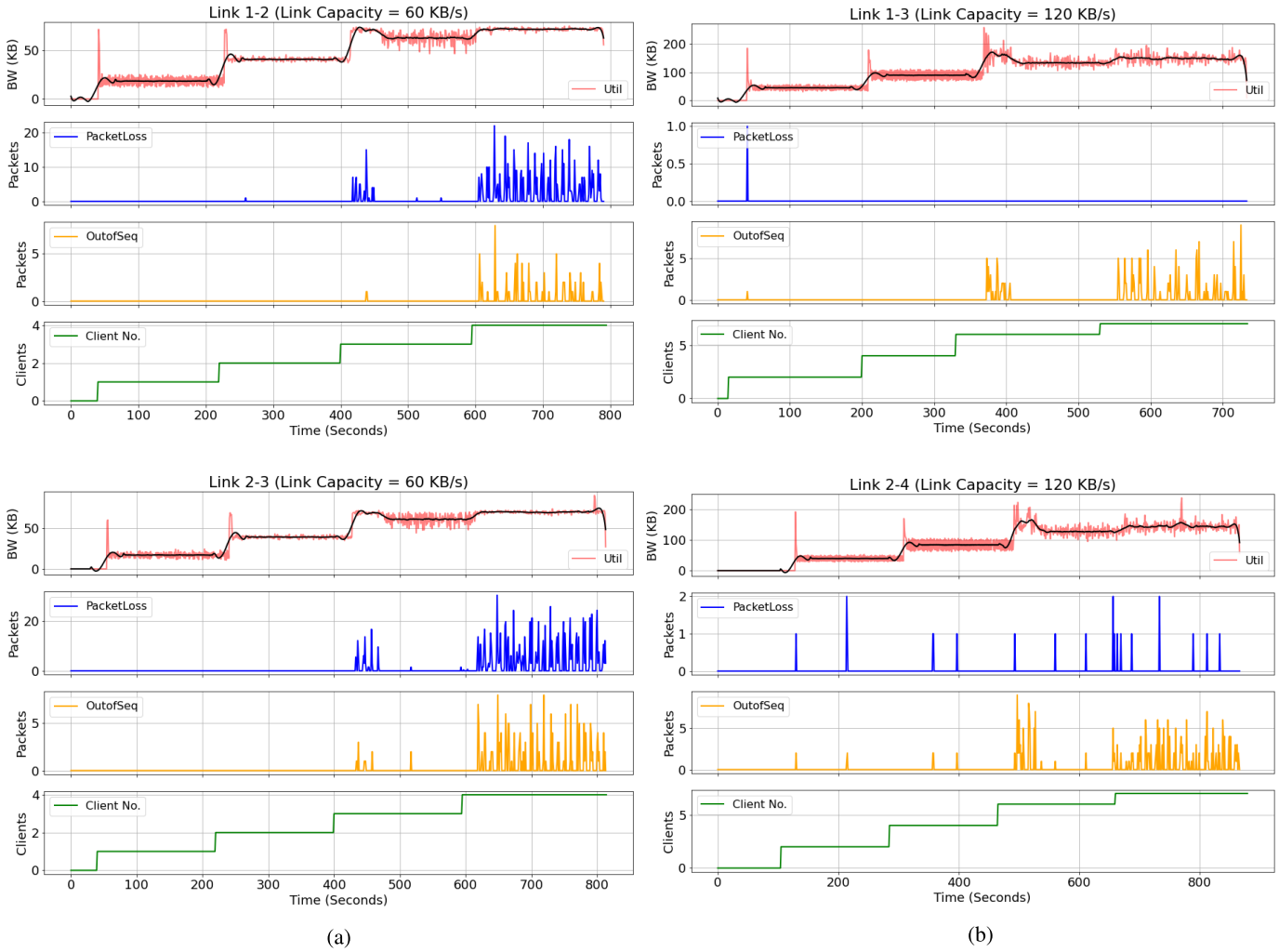

Fig. 3. MPLS link utilization. (a) Unique links. (b) Shared links.

TABLE I
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

support up to three simultaneous users transferring files up to 20 KB/s each. Wireshark is used to ensure the effectiveness of the applied policies versus the number of users per path [36], as shown in Fig. 3. In Fig. 3, there are two types of links: unique and shared links. Unique links are a member of only one path. Therefore, a unique links' maximum bandwidth is 60 KB/s; i.e., the bandwidth of three users, while a shared link aggregates two or more paths, and its maximum bandwidth is double the unique link.

The layer-3 policies help in defining the network states. The normal state is when any link has a maximum of three users. Congestion state is when there are more than three users connected to a path. In addition, we define network physical issues, such as any network device failure or path disconnectivity, by a composition of delay and jitter ranging between 80–350 and 5–60 ms, respectively.

B. Data Collection and Preparation

A python script is written to interact with GNS3's REST API. The API can turn ON and OFF any network device. In addition, it can impose delay and/or jitter on any of the network's links. The python script defines the scenario of data collection by adding the ability to do the live configuration of any network device and control the operation of the nine users. The script decides randomly the path that each user connects to, and also collects QoS and QoE metrics every 30 s. The QoS metric consists of: 1) end-to-end delay, jitter, and packet loss values; i.e., SLA measurements of the three paths and 2) input and output rates, and packet loss values of the router ports in AS-1. File transfer downloading speed is the only QoE metric we gather. Afterward, QoS and QoE metrics are merged and aligned with the aid of timestamps. Table II summarizes the collected features and labels.

We collected over 20 days of data and a dataset of 56 000 data points. The rules utilized in data collection are presented in Algorithm 1. It was fairly easy for us to

TABLE II
FEATURES AND LABELS SUMMARY

Features
QoS features: • *Delay*: End-to-end Path delay. • *Jitter*: End-to-end Path jitter. • *Packet loss*: End-to-end Path packet loss. • *Out of sequence*: Number of out of sequence packets. • *Discarded samples*: Number of discarded packets due to full buffer at the receiver. • *Port statistics loss*: average packet loss of AS-1's routers ports. • *Port statistics rates*: average input and output packet rates of AS-1's routers ports. **QoE features:** • Download speed: FTP transfer speed.
Labels
Path Congestion: • *Cong:P1*: Congestion is detected on $Path_1$. • *Cong:P2*: Congestion is detected on $Path_2$. **Network Physical Issue:** • *NPI:1→3*: Physical issue is detected between router 1 & 3. • *NPI:2→4*: Physical issue is detected between router 2 & 4. • *NPI:2→1*: Physical issue is detected between router 2 & 1. • *NPI:3→2*: Physical issue is detected between router 3 & 2. • *NPI:3→5*: Physical issue is detected between router 3 & 5.

Algorithm 1 Operational Labeling Rules for This Experiment

```
while True do
    choice = random(add_client, physical_prob)
    if choice == physical_prob then
        label = NPI
    end
    else
        if client_len(PATH1)>3 then
            label = Congest:P1
        end
        else if client_len(PATH1)>3 then
            label = Congest:P2
        end
        else
            label = Normal
        end
    end
end
```

define these rules since our testbed network is not complex. However, in a more realistic network, these rules become more difficult to define, and this is why NOCs employ expert technicians. The collected dataset has 66 different features. The labeled network status distribution is shown in Fig. 4(a). In Fig. 4(a), it is obvious that the normal state is predominant, which is the case in the real world, so that our data are realistic. We achieved this condition by resetting our network to the normal state before introducing another abnormal state. For instance, if the current state is to congest $PATH_1$, after three iterations, one of the $PATH_1$'s clients is moved either to $PATH_2$ or $PATH_3$ to reset the network to its normal state.

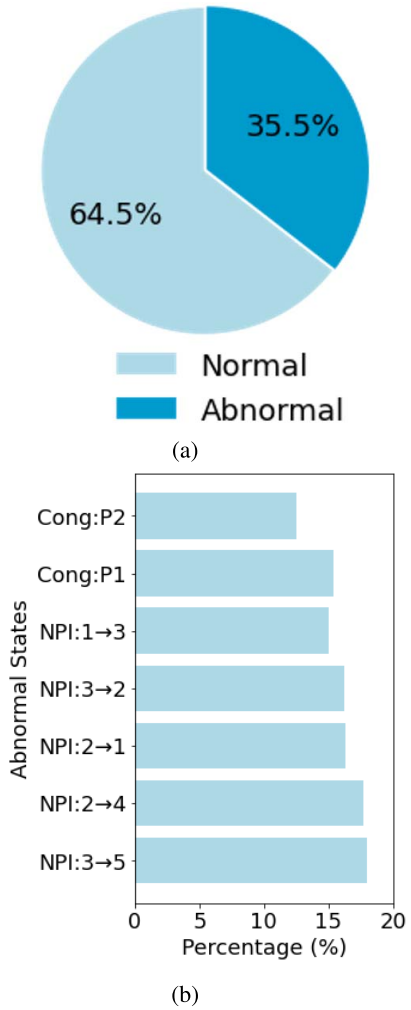

Fig. 4. Actions space distribution. (a) Class distribution. (b) Minority class breakdown.

Likewise, if the current state is a physical issue realized by enforcing a delay and jitter filter, after three iterations, the filter is removed to reset the network to its normal state. As a result, we encounter the problem of an unbalanced classification problem; i.e., one class, including more data points than other classes. Again, this problem is a realistic reflection of what happens in the real world; therefore, creating an imbalanced dataset with the Normal state having more instances than the other classes is intentional. Fig. 4(b) shows the distribution of the abnormal states labeled by rules that mimic the NOC technicians' decisions. The abnormal network states can be mainly classified into either a network physical issue, denoted by "NPI", or a congested path, denoted by "Cong."

C. Model Selection

As described in Sections III-A and III-B, the dataset consists of classes with uneven distribution of instances. This phenomenon creates two issues: 1) the AI model acts greedy and, therefore, becomes more biased toward the majority class to maximize the overall accuracy and 2) the data points of the minority classes are not representative enough to describe the classes. These two issues negatively affect the model

TABLE III
MODELS' PARAMETERS

DT	Split criterion: gini, Splitter: best, Depth:19, Leaves:149 Class Weight: 'Balanced'
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'
NN	Learning rate: 0.01, Hidden layers: 2, Neurons/Layer: 128, Optimizer: Adam

TABLE IV
OVERALL MODEL ACCURACIES

Model	**DT**	**GB**	**XGB**	**NN**
Accuracy	97.1%	98.1%	99.0%	98.6%

performance because the model favors the predominant class. But, in fact, it is the minority classes that are the points of interest; i.e., they are the classes that we are keen to classify: congestion and physical fault.

To overcome this problem, we utilize ensemble methods, which use the unification of weak learners to advance the predictive performance. When new data are introduced, weak learners provide their votes for classes. After that, the class with the most votes is chosen for that data point. In our case, we tested a simple DT [37] algorithm and two ensemble methods: GB [38] and XGB [39] algorithms.

We also use another solution for the unbalanced data problem: employing a fully connected neural network with weight balancing. Weight balancing gives more weight to the more important classes by multiplying the loss of each data point by a certain factor depending on their class, leading to an overall balance in the data. The results for all four are reported in Section IV.

During training, we noticed that the 30-s period was creating a dataset, where congestion states are not distinguished clearly. This was overcome by a two-factor aggregation strategy: we took the average of the SLA and QoE measurements, and we also set the abnormal network state as the dominant action; i.e., the abnormal label is always selected if the two aggregated labels are normal and abnormal.

IV. EVALUATION AND RESULTS

As mentioned earlier, our goal is to not only classify the network state but also localize a fault if it exists; in fact, five of our model classes indicate the specific location of the physical problems in the network. Given the moderate amount of training data (60% of the dataset) and the complexity of the problem, we find that XGB provides optimal performance. DT is too simple and provides the least good performance, while the NN model is too complex and requires more training data to perform better. Table III lists the four models' parameters. Table IV summarizes the overall accuracies for the four classification models. As can be seen, all of the classifiers performed well with a minimum accuracy of 97.1% and with XGB having the highest accuracy at 99.0%.

A similar trend is observed for the precision and recall values shown in Tables V and VI, which also show the fault localization performance. Compared with the other models, XGB performed better for almost every class in the dataset. Precision values for the Normal class are over 0.97 and for the NPI classes are either 0.99 or 1. However, model performances degraded for the congestion classes, where precision varies between 0.73 and 0.93 for Congest:P1, and between 0.70 and 0.95 for Congest:P2. The recall follows a similar pattern: recall values for the normal class are over 0.97 and for the NPI classes are between 0.98 and 1, while performance degrades again for congestion classes, with recall values being between 0.76 and 0.90 for Congest:P1 and between 0.78 and 0.91 for Congest:P2. While precision and recall are low for the Congestion classes, XGB still performed the best in identifying the congestion in $PATH_1$ and $PATH_2$ with values over 0.90. In addition, all the models yielded higher scores for Congest:P2 than Congest:P1, the only exception being the precision value of the GB model. In addition, for NPI:3→5, all models have the value 1 except DT in precision, which is 0.99.

We can, therefore, see that XGB is the best performer, DT performs poorly and, GB and NN yields similar performance. As stated earlier, NN underperforms due to a lack of sufficient training data. However, these relatively poor classification performances in congestion classes did not significantly affect the overall accuracy of the models since the normal class had considerably more instances than others combined, see Fig. 4.

We believe that the confusion between the normal and congestion states is due to three reasons: 1) the strict labeling technique presented in Algorithm 1 sets the network state as congestion immediately if one of the paths has more than three clients. 2) The fact that congestion needs time to build up and to be shown in the collected metrics due to the routers' buffer. Therefore, the collected metrics start reflecting congestion after the buffer floods. 3) ML-model sampling frequency, i.e., how often the model takes a decision. If the model takes its decision before the metrics reflect congestion, the model's decision will be "normal," while the label is "congestion". That is the cause of the confusion. But if the model takes its decision after the metrics reflect congestion, the model's decision will be "congestion" and the label is also "congestion."

To further assess the classification models, we opted to utilize confusion matrices. The normalized confusion matrices for the classifiers are shown in Fig. 5, again including fault localization (NPI). We can see that all models are able to make perfect detection of NPI classes. The confusion happens between the normal and congestion classes. In fact, it can be said that this is the only considerable confusion causing the models to deteriorate. DT and GB fail drastically to distinguish congestion in $PATH_1$ and $PATH_2$ from the normal state, while NN is slightly better at separating the two congestion classes from the normal state. Unlike its counterparts, XGB's performance is not weakened by this task. Table VII shows a comparison of this work with state-of-the-art.

In addition, the ROC curves are shown in Fig. 6. These graphs summarize the performances of the classifiers over each probable threshold for every class. As can be seen,

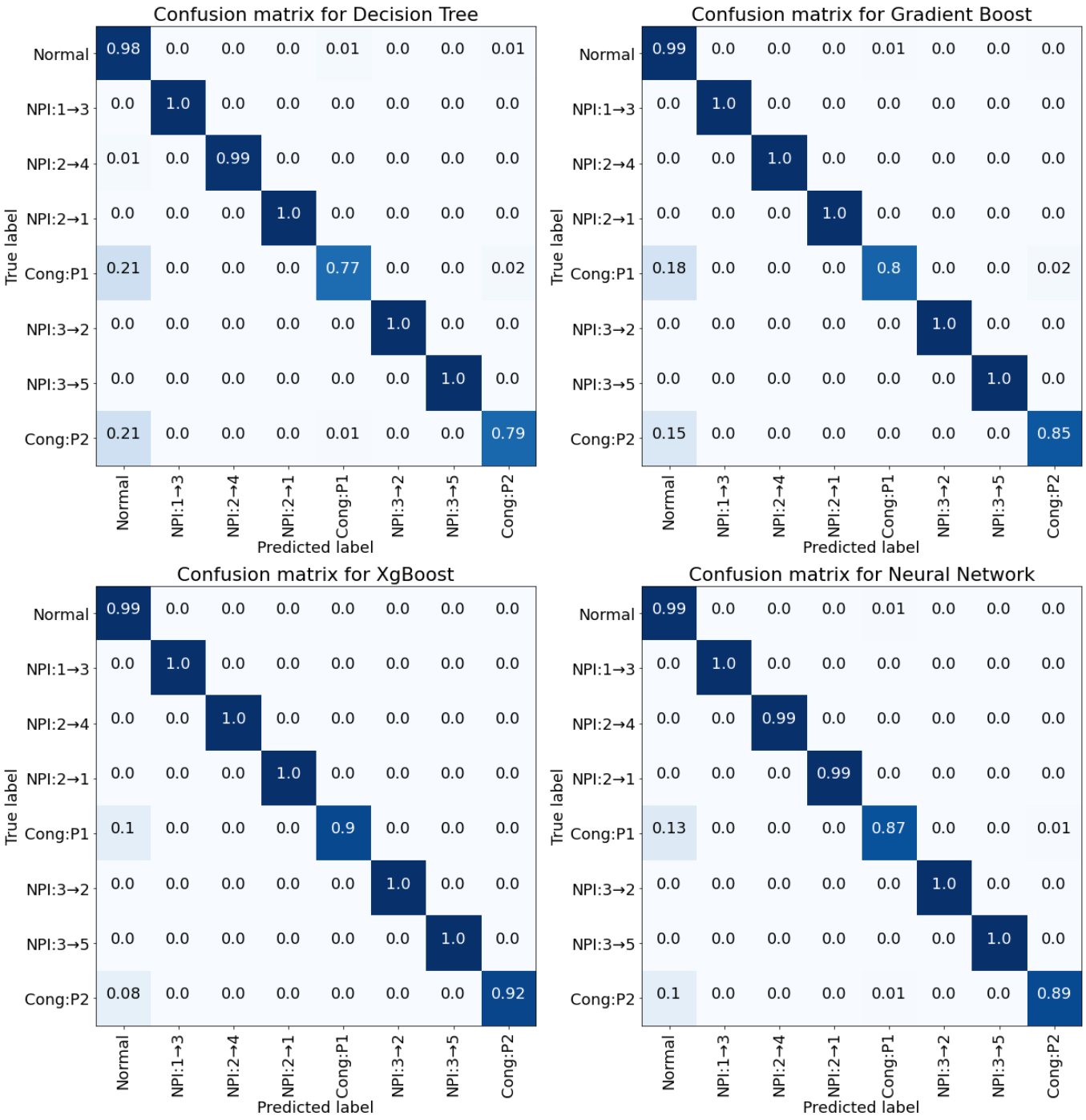

Fig. 5. Confusion matrices for DT, GB, XGB, and NN.

TABLE V
PRECISION FOR EACH CLASS

State / Model	Normal	NPI:3→5	NPI:2→4	NPI:2→1	NPI:3→2	NPI:1→3	Congest:P1	Congest:P2
DT	0.97	0.99	0.99	0.99	0.99	0.99	0.73	0.79
GB	0.98	1	0.99	0.99	1	1	0.87	0.86
XGB	0.99	1	1	0.99	0.99	1	0.93	0.95
NN	0.98	1	0.99	1	0.99	0.99	0.87	0.90

the classifiers have a high predictive ability with the area under curve (AUC) values ranging between approximately 0.80 and 1.00, and the latter symbolizing a perfect test. Almost all the classifiers can easily identify the first six classes shown in Fig. 6. However, some deterioration in AUC values occurs in identifying Cong:P1 and Cong:P2 classes. DT classifier gives

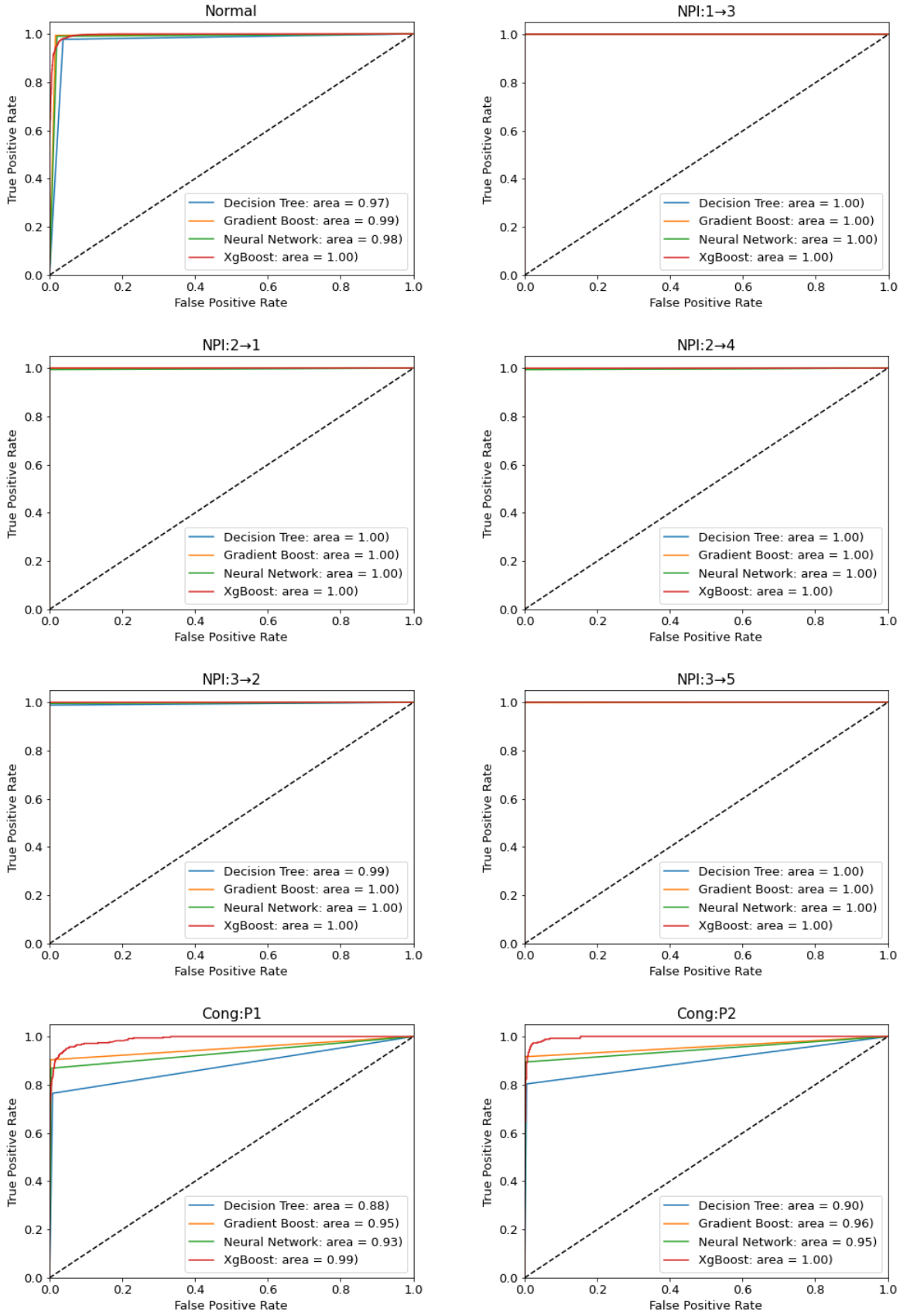

Fig. 6. ROC curves for DT, GB, XGB, and NN.

TABLE VI
RECALL FOR EACH CLASS

State / Model	Normal	NPI:3→5	NPI:2→4	NPI:2→1	NPI:3→2	NPI:1→3	Congest:P1	Congest:P2
DT	0.97	1	0.98	0.99	1	0.99	0.76	0.78
GB	0.98	1	0.99	1	0.99	0.99	0.80	0.85
XGB	0.99	1	0.99	1	0.99	1	0.90	0.91
NN	0.98	1	0.99	0.99	0.99	1	0.86	0.89

TABLE VII
PERFORMANCE AND SUMMARY COMPARISON

Paper	Platform	ML-Type	Model	Pro/Re-Active	Contribution	Reported Results
[7]	SDN	Unsupervised	K-Nearest Neighbour	Reactive	Classify network status into congested or not-congested	Acc: 98%
[8]	Standard	Supervised	Naive Bayes	Reactive	Categorize the network status as faulty or normal and localize faults	Acc: 95%
[9]	NFV	Supervised	Support Vector Machine	Both	classify network status and predict if the problem happened or not	Acc:95%
Our work	Standard	Supervised	Extreme Gradient Boosting	Reactive	Classify network status into 3 classes and localize faults	Acc:99%

the lowest performance, whereas XGB provides the highest, almost reaching 1.00. GB and NN yield very similar results.

Although NNs, such as DL, have great potential, they come with a significant drawback: the limited availability of labeled data decreases the accuracy in classification and limits the choices of algorithms since DL techniques often require a large amount of data for training.

V. CONCLUSION

In this work, we proposed an ML-based system that distinguishes between congestion, network physical issue, and normal states. Then, we studied the performance of three ML methods to classify the network status using a dataset generated in GNS3. We showed the feasibility of the ensemble models, with the XGB algorithm having the highest overall accuracy of 99%.

Several research venues from this work remain open. In future work, first, we will explore the scalability of our approach and study how the selected ML models perform in real time in a diverse and complex network with more path combinations and routers. Our selected ML model is network specific. Today, the domain of taking an ML method from the laboratory to an actual environment and ensuring that it can operate smoothly in the real world is called machine learning operations (MLOps) [40]. It is well-known that ML models heavily depend on the quality of data and are tailored to the targeted applications (in our case, a specific network configuration). MLOps has proven that trained models can be migrated to other applications and still function after fine tuning, transfer learning, or (in the worst case scenario) retraining. Furthermore, we will investigate the emergence of new network states. With the discovery of new states, we will consider more advanced ML algorithms, such as a deep neural network to find hidden patterns and perfect the classification performance. We will especially focus on attention models to study the dataset in a sequential format since every new data in our dataset depends on the previous data point.

Finding a way to combine unsupervised learning with supervised learning to teach NNs how to learn with less data is a promising area of research. Furthermore, teaching NNs to accumulate their knowledge will make them more effective and efficient in learning new things, and thus, fewer data will be required for training.

Although our system is not a human life-critical system, like a medical or military system or self-driving cars, we would like to explore the realm of explainable AI to study how the AI model takes its decisions.

Finally, network state classification paves the path toward identifying the cause of traffic congestion. Based on such predictions, potential actions can be suggested to resolve the problem autonomously.

REFERENCES

[1] M. Benatia, A. Louis, and D. Baudry, "Alarm correlation to improve industrial fault management," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 10485–10492, 2020.

[2] A. R. Mohammed, S. A. Mohammed, and S. Shirmohammadi, "Machine learning and deep learning based traffic classification and prediction in software defined networking," in *Proc. IEEE Int. Symp. Meas. Netw. (M N)*, Jul. 2019, pp. 1–6.

[3] B. Ma, H. Zhang, Y. Guo, Z. Liu, and Y. Zeng, "A summary of traffic identification method depended on machine learning," in *Proc. Int. Conf. Sensor Netw. Signal Process. (SNSP)*, Oct. 2018, pp. 469–474.

[4] *Blue Planet Unified Assurance and Analytics*. Accessed: Feb. 20, 2021. [Online]. Available: https://www.blueplanet.com/products/uaa.html

[5] *Blue Planet Network Health Predictor*. Accessed: Feb. 20, 2021. [Online]. Available: https://www.blueplanet.com/resources/Blue-Planet-Network-Health-Predictor.html

[6] *Blue Planet Multi-Domain Service Orchestration*. Accessed: Feb. 20, 2021. [Online]. Available: https://www.blueplanet.com/products/multi-domain-service-orchestration.html

[7] R. Kumar, U. Venkanna, and V. Tiwari, "A binary classification approach for time granular traffic in SDWMN based IoT networks," in *Proc. Int. Conf. Commun. Syst. Netw. (COMSNETS)*, Jan. 2020, pp. 531–534.

[8] A. Saple and L. Yilmaz, "Agent-based simulation study of behavioral anticipation: Anticipatory fault management in computer networks," in *Proc. 44th Annu. Southeast Regional Conf.*, 2006, pp. 383–388.

[9] L. Gupta, T. Salman, M. Zolanvari, A. Erbad, and R. Jain, "Fault and performance management in multi-cloud virtual network services using AI: A tutorial and a case study," *Comput. Netw.*, vol. 165, Dec. 2019, Art. no. 106950.

[10] Y. Jin, M. Tomoishi, and S. Matsuura, "Detection of hijacked authoritative dns servers by name resolution traffic classification," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Dec. 2019, pp. 6084–6085.

[11] S. Naseer *et al.*, "Enhanced network anomaly detection based on deep neural networks," *IEEE Access*, vol. 6, pp. 48231–48246, 2018.

[12] T. Ongun, T. Sakharaov, S. Boboila, A. Oprea, and T. Eliassi-Rad, "On designing machine learning models for malicious network traffic classification," 2019, *arXiv:1907.04846*. [Online]. Available: http://arxiv.org/abs/1907.04846

[13] Y. Zuo, Y. Wu, G. Min, C. Huang, and K. Pei, "An intelligent anomaly detection scheme for micro-services architectures with temporal and spatial data analysis," *IEEE Trans. Cognit. Commun. Netw.*, vol. 6, no. 2, pp. 548–561, Jun. 2020.

[14] Z. Tian *et al.*, "Deep learning and Dempster-Shafer theory based insider threat detection," *Mobile Netw. Appl.*, pp. 1–10, Oct. 2020. [Online]. Available: https://link.springer.com/article/10.1007/s11036-020-01656-7

[15] A. S. Iliyasu and H. Deng, "Semi-supervised encrypted traffic classification with deep convolutional generative adversarial networks," *IEEE Access*, vol. 8, pp. 118–126, 2019.

[16] A. S. Khatouni and N. Zincir-Heywood, "Integrating machine learning with off-the-shelf traffic flow features for http/https traffic classification," in *Proc. IEEE Symp. Comput. Commun. (ISCC)*, Jun. 2019, pp. 1–7.

[17] M. Lotfollahi, M. J. Siavoshani, R. S. H. Zade, and M. Saberian, "Deep packet: A novel approach for encrypted traffic classification using deep learning," *Soft Comput.*, vol. 24, no. 3, pp. 1999–2012, 2020.

[18] S. Soleymanpour, H. Sadr, and H. Beheshti, "An efficient deep learning method for encrypted traffic classification on the web," in *Proc. 6th Int. Conf. Web Res. (ICWR)*, 2020, pp. 209–216.

[19] X. Wang, S. Chen, and J. Su, "App-Net: A hybrid neural network for encrypted mobile traffic classification," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Jul. 2020, pp. 424–429.

[20] S. Patel *et al.*, "Network traffic classification analysis using machine learning algorithms," in *Proc. Int. Conf. Adv. Comput., Commun. Control Netw. (ICACCCN)*, Oct. 2018, pp. 1182–1187.

[21] M. Amiri, H. Al Osman, and S. Shirmohammadi, "Game-aware and sdn-assisted bandwidth allocation for data center networks," in *Proc. IEEE Conf. Multimedia Inf. Process. Retr. (MIPR)*, Apr. 2018, pp. 86–91.

[22] I. L. Cherif and A. Kortebi, "On using extreme gradient boosting (XGBoost) machine learning algorithm for home network traffic classification," in *Proc. Wireless Days (WD)*, 2019, pp. 1–6.

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

[24] M. Abbasi, A. Shahraki, and A. Taherkordi, "Deep learning for network traffic monitoring and analysis (NTMA): A survey," *Comput. Commun.*, vol. 170, pp. 19–41, Feb. 2021.

[25] J. Park and J. Kim, "A classification of network traffic status for various scale networks," in *Proc. Int. Conf. Inf. Netw. (ICOIN)*, 2013, pp. 595–599.

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

[27] S. A. Mohammed *et al.*, "A multimodal deep learning-based distributed network latency measurement system," *IEEE Trans. Instrum. Meas.*, vol. 69, no. 5, pp. 2487–2494, Jan. 2020.

[28] J. Fabini and M. Abmayer, "Delay measurement methodology revisited: Time-slotted randomness cancellation," *IEEE Trans. Instrum. Meas.*, vol. 62, no. 10, pp. 2839–2848, Oct. 2013.

[29] T. Eylen and C. F. Bazlamaçci, "One-way active delay measurement with error bounds," *IEEE Trans. Instrum. Meas.*, vol. 64, no. 12, pp. 3476–3489, Dec. 2015.

[30] A. I. Moustapha and R. R. Selmic, "Wireless sensor network modeling using modified recurrent neural networks: Application to fault detection," *IEEE Trans. Instrum. Meas.*, vol. 57, no. 5, pp. 981–988, May 2008.

[31] S. C. Chan, H. C. Wu, and K. M. Tsui, "Robust recursive eigendecomposition and subspace-based algorithms with application to fault detection in wireless sensor networks," *IEEE Trans. Instrum. Meas.*, vol. 61, no. 6, pp. 1703–1718, Jun. 2012.

[32] A. Javadtalab, M. Semsarzadeh, A. Khanchi, S. Shirmohammadi, and A. Yassine, "Continuous one-way detection of available bandwidth changes for video streaming over best-effort networks," *IEEE Trans. Instrum. Meas.*, vol. 64, no. 1, pp. 190–203, Feb. 2013.

[33] *GNS3*. Accessed: Feb. 20, 2021. [Online]. Available: https://www.gns3.com/

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

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

[36] *Wireshark, Go Deep*. Accessed: Feb. 20, 2021. [Online]. Available: https://www.wireshark.org/

[37] J. Quinlan, "Induction of decision trees," *Mach. Learn.*, vol. 1, no. 1, pp. 81–106, 1986.

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

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

[40] Y. Zhou, Y. Yu, and B. Ding, "Towards MLOps: A case study of ml pipeline platform," in *Proc. Int. Conf. Artif. Intell. Comput. Eng. (ICAICE)*, 2020, pp. 494–500.

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

Shady A. Mohammed received the M.Sc. degree in electronics and computer engineering from Istanbul Sehir University, Istanbul, Turkey, in 2016. He is currently pursuing the Ph.D. degree in electrical and computer engineering with the University of Ottawa, Ottawa, ON, Canada.

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

David Côté received the Ph.D. degree in physics from the Université de Montréal, Montreal, QC, Canada, in 2007.

He was a Particle Physicist with the Stanford Linear Accelerator Center, Menlo Park, CA, USA, and CERN, Meyrin, Switzerland, notably, for 14 years, where he gained substantial hands-on experience with big data engineering, data science, and world-class research. He is currently the Chief Data Scientist with the Blue Planet Division, Ciena Corporation, Ottawa, ON, Canada, where he is also a Co-Founder of the Blue Planet Analytics program. 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.

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

He is currently a Professor with the School of Electrical Engineering and Computer Science, University of Ottawa. He is also the Director of the Distributed and Collaborative Virtual Environments Research Laboratory, 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 post-doctoral, Ph.D., and master's levels, 30 patents and technology transfers to the private sector, and a number of awards.

Dr. Shirmohammadi served as the Vice President of the Membership Development Committee, IEEE Instrumentation and Measurement Society, from 2014 to 2017. He 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. He has been an IEEE Instrumentation and Measurement Society AdCom member since 2014. He is currently on the Editorial Board of the *IEEE Instrumentation and Measurement Magazine*. He received the 2019 George S. Glinski Award for Excellence in Research, a Senior Member of the ACM. He received the University of Ottawa Gold Medalist. He was the Associate Editor-in-Chief of the *IEEE Instrumentation and Measurement Magazine* from 2014 to 2015. He is the Editor-in-Chief of the IEEE TRANSACTIONS ON INSTRUMENTATION AND MEASUREMENT. He is a licensed Professional Engineer in Ontario.