Received 19 December 2022; revised 17 January 2023; accepted 5 February 2023. Date of publication 14 February 2023; date of current version 1 March 2023.

*Digital Object Identifier 10.1109/OJCOMS.2023.3244684*

# Determining Most Likely Links (MLL) for Network Fault Localization

**CHRISTOPHER MENDOZA AND MICHAEL P. MCGARRY (Senior Member, IEEE)**

Department of Electrical and Computer Engineering, University of Texas at El Paso, El Paso, TX 79968, USA

CORRESPONDING AUTHOR: M. P. MCGARRY (e-mail: mpmcgarry@utep.edu)

**ABSTRACT** We propose and evaluate a technique that learns the probability of a network transmission link experiencing a fault by using outlier flows (in the performance sense) as training data. This technique autonomously determines the *most likely links* causing performance degradation in a communications network; a critical feature of zero-touch network management. Our new Network Link Outlier Factor (NLOF) with most likely links (NLOF:MLL) is experimentally compared to the existing literature (including our original NLOF) using classification performance measures: recall, precision, $F_1$-score, and time-to-detection. We utilize inferential statistics and a wide set of Mininet experiments to determine statistically significant performance differences. We find that our NLOF:MLL outperforms the existing literature wrt the important $F_1$-score while exhibiting a competitive time-to-detection.

**INDEX TERMS** Zero-touch network management, clustering, outlier detection, data streaming, fault detection, fault localization.

## I. INTRODUCTION

NETWORK management can be quite overwhelming for network managers due to the large scale and heterogeneity of the networks they manage. The FCAPS model [1] adapted from ITU M.3400 partitions network management into five tasks:

1) **F**ault detection and correction
2) **C**onfiguration and operation
3) **A**ccounting and billing
4) **P**erformance assessment and optimization
5) **S**ecurity assurance and protection

To perform these tasks, network managers must consistently analyze network systems that are in many cases large scale and composed of heterogeneous devices. Enterprise networks consist of many hosts, switches and transmission channels; in many cases supplied by several vendors. Service provider networks are even larger with even more heterogeneity. When building networks, network managers purposefully select equipment from various vendors so they never become solely dependent on a single vendor. Software-defined networking (SDN) [2], [3] interfaces to network devices and software systems utilizing those interfaces can lead to the automation of network management tasks; relieving the burden on human network managers. Recent advances in machine learning algorithms/implementations coupled with instrumentation to collect voluminous data have made the application of machine learning quite fruitful. Indeed machine learning can be fruitfully applied to automate the tasks of network management; this is often referred to as zero-touch network management [4], [5], [6], [7].

We believe the network management task of fault detection and correction is quite stressful for network managers. These tasks are often preceded by user complaints and need to be resolved as quickly as possible. Automating this task will have a significant positive impact on the practice of network management. Specifically, the task of fault detection requires acquiring and analyzing diverse information spread across a network. That task can be overwhelming for a human network operator especially when subject to the stress of having to resolve the problem quickly. As a result, machine learning has a very important role to both reduce mistakes and reduce time-to-detection.

Our work seeks to apply machine learning to advance the automation of network fault localization. We formerly proposed and evaluated our Network Link Outlier Factor (NLOF) [8], [9], [10] for that purpose. NLOF utilizes network flow and topology data to localize network faults through the computation of outlier scores for every link in a network under management. Those outlier scores are a

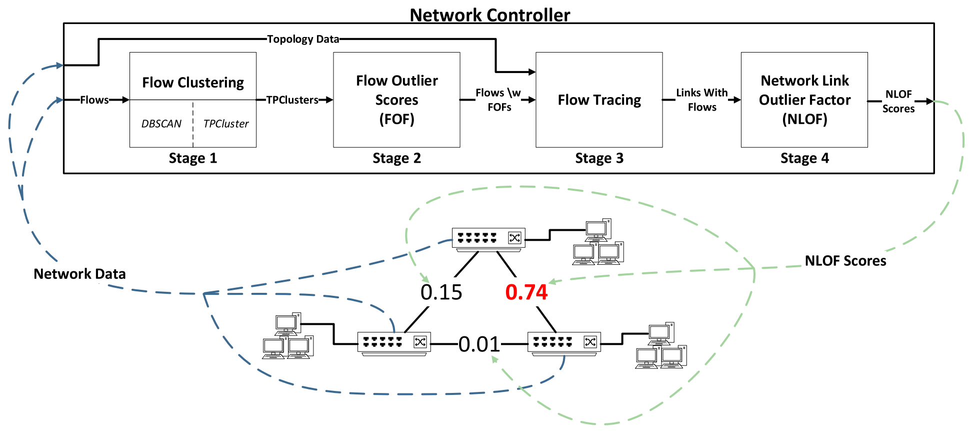


FIGURE 1. Our original Network Link Outlier Factor (NLOF) [10] consists of a 4-stage pipeline: clustering, flow tracing, and outlier scores (flows and links).

measure, from 0 to 1, indicating how a link is isolated from others (i.e., an outlier) in a performance context. Another view is that the outlier scores represent the probability that a link is experiencing a fault. When a link's outlier score exceeds a threshold, a link is classified as experiencing a fault. We discuss the selection of this threshold and other algorithm hyper-parameters in Section II-E.

NLOF consists of a 4-stage pipeline of unsupervised machine learning algorithms. In stage 1, flows are clustered into performance cohorts in two sub-stages using the DBSCAN algorithm and then our TPCluster algorithm. In stage 2, each flow is assigned an outlier score that measures its distance in feature space to a performance exemplar in its cohort cluster. In stage 3, flows and network links are correlated using the topology data. Finally, in stage 4 each network link is assigned an outlier score that is the ratio of outlier flows to all flows traversing that link. Figure 1 illustrates this 4-stage pipeline of clustering and outlier detection.

When detecting faults on links at the edge of the network we found that a simple threshold-based detection/classification rule on the outlier scores is sufficient. However, when a fault occurs on a link toward the core of the network the network links that are downstream (i.e., closer to the edge) have indistinguishable outlier scores from the actual link experiencing the fault, thereby resulting in false positives for those downstream links.

### A. LIMITATIONS OF ORIGINAL NLOF

If we consider topological relationships in our detection rule we can isolate the upstream link that is the origin of the fault [10]. However, we want to simplify the detection rule so, in this work, we are proposing a different approach to the final stage of the NLOF pipeline: the link outlier score computation. To understand the limitations of using the ratio of outlier flows to all flows traversing a link as the outlier score, we conducted a sensitivity analysis, see the Appendix for details. The objective is that the ratio of observed outlier flows to all observed flows (i.e., the outlier score) reflects the probability that a link is experiencing a fault that reduces the throughput of those outlier flows. From our sensitivity analysis, we can see that this ratio is not only affected by a fault on a link but is sensitive to where flows are sampled. This sensitivity limits the effectiveness of our original NLOF. We need an outlier score that is not derived from that ratio but rather learned through observing outlier and normal flows, in the performance context.

### B. OUR CONTRIBUTION

We redesign the outlier score computation of NLOF such that a simple threshold-based detection rule can be used in all instances and prevent the false positives discussed above. Specifically, we propose and evaluate a technique that learns the probability of a link experiencing a fault by using network flows as training data; performance outlier flows increase fault probability and normal flows decrease fault probability. Our technique also learns the specific links along a flow path most responsible for the performance degradation of a flow thereby improving the learning of the fault probabilities (i.e., outlier scores) that determine the fault locations. This approach to determining an outlier score is in contrast to our previous NLOF algorithm that assigned an outlier score as the ratio of outlier flows to total flows traversing a link. With our new machine learning technique we thereby learn the *most likely links* causing the performance degradation of the outlier flows. Our new NLOF with most likely links (NLOF:MLL) is experimentally compared to the existing literature (including our original NLOF) using classification performance measures: recall, precision, $F_1$-score, and time-to-detection. We employ inferential statistics to identify statistically significant differences in those performance measures.

### C. RELATED WORK

Early work on automated network fault detection in the 1990s and 2000s was driven mostly by a few research groups. A group led by F. Feather at Carnegie Mellon [11], [12] utilized the detection of abrupt changes in network performance measures to detect the occurrence of a fault. This group also used fault feature vectors to identify the specific cause of the fault. Similarly, a group led by C. Ji at RPI/Georgia Tech used abrupt changes in MIB variables (i.e., network switch event counters) to detect network faults. The abrupt change detection technique continued to be pursued throughout the 2000s; for some examples, see [13], [14].

Another group led by M. Schwartz at Columbia [15], [16], [17] used network alarm data to detect network faults using context-free grammars and probabilistic finite state machines.

Modeling network behavior and using predicted responses of the model to detect faults in the real network was another avenue of investigation in the 2000s; for some examples, see [18], [19].

Similar to our NLOF:MLL, the work presented in [20] proposes a method to assign a fault probability to each link in a network. The method uses *active measurements*, active measurements inject instrumentation traffic in the network to make the measurement, this is in contrast to *passive measurement* that makes a measurement by observing existing traffic. The active measurement of this technique is used to determine if a path between measurement points is meeting a service level agreement (SLA). Each active measurement triggers an update to the fault probability of a link. If the active measurement indicates an SLA violation (i.e., performance degradation), then the fault probability is increased for all links on the path and decreased for all links not on the path. If the active measurement does not indicate an SLA violation (i.e., there is no performance degradation), then the fault probability is decreased for all links (on the path and not on the path). In [21] the authors modify the probability update to decrease the fault probability of links on the active measurement path at a rate higher than those not on the path. The sum of the probabilities across all links in the network is 1; the sample space consists of all of the links in the network. If the probability assigned to a link exceeds a threshold, that link is classified as faulty. NLOF instead uses passive measurements, specifically flow and topology data; the flow data is existing traffic (i.e., no instrumentation traffic is inserted in the network). Further, the flow data only needs to be collected from a subset of switches whose transiting flows cover, by their paths in the network, a sufficient fraction of the links in the network for the desired fault coverage.

Also similar to our NLOF:MLL, 007 [22] assigns scores to links to localize faults. However, the manner in which 007 derives scores is quite different. 007 requires data about TCP flows (e.g., retransmissions) collected by agents running on each end host. It computes the scores using number of bad flow votes (i.e., those experiencing retransmissions). In a second round, votes can be redistributed to the link on the path most responsible (i.e., highest number of bad votes). NLOF uses flow data that can be collected at switches rather than hosts and that flow data does not need to be collected at every switch, as mentioned above. Our new outlier score technique, proposed here, derives scores in a way that does not assume that only one link on a path is responsible for the degraded performance. Lastly, we use flow throughput not retransmissions as the primary flow feature. Reduced throughput is indicative of performance degradation with less severity than is indicated by retransmissions; our technique should find more subtle failures.

### D. OUTLINE

The rest of this paper is organized as follows. In Section II we describe our algorithm that learns the probability that a link is experiencing a fault. In Section III we present our experimental plan designed to compare the performance of our NLOF:MLL with our original NLOF and two techniques from the literature: 007 and abrupt change detection (ACD). In Section IV we present the results of the experiments and discuss our findings across four classification performance measures: recall, precision, $F_1$-score, and time-to-detection (TTD). Finally, in Section V we summarize our findings and outline paths for future investigation.

## II. DETERMINING MOST LIKELY LINKS EXPERIENCING A FAULT

NLOF scores (ranging from 0 to 1) serve as a probability that a link is experiencing a fault. Let $y_i$ be a binary random variable describing the state of link $i$ as faulty when equal to 1 and normal when equal to 0. Let $\hat{\mathcal{P}}\{y_i = 1\}$ be the NLOF score or estimate of the probability that link $i$ is experiencing a fault, and $\gamma$ be a threshold on that probability estimate. We classify link $i$ as experiencing a fault when,

$$\hat{\mathcal{P}}\{y_i = 1\} > \gamma. \tag{1}$$

See Section II-E for a discussion regarding setting that threshold $\gamma$. The original NLOF scores (ratio of outlier flows to all flows traversing a link) were inspired by the relative frequency definition of probability. Let $\boldsymbol{a}_i$ be the set of outlier flows that traverse link $i$, $\boldsymbol{l}_i$ be the set of all flows that traverse link $i$. Our original NLOF scores were computed as,

$$\hat{\mathcal{P}}\{y_i = 1\} = \frac{|\boldsymbol{a}_i|}{|\boldsymbol{l}_i|}. \tag{2}$$

However, that approach to producing the NLOF scores has a shortcoming: each outlier flow has an equivalent effect on the NLOF score of each link it traverses. We now make the reasonable assumption that not every link traversed by a performance-degraded flow is experiencing a fault. Based on that assumption each outlier flow should rather have a weighted effect on the outlier score of each link it traverses. This weighting should utilize historical information to determine the most likely links responsible for the outlier (or performance-degraded) flow. Links that have historically been associated with more outlier flows should be considered more likely and therefore have their NLOF outlier scores increased more by the current outlier flow. The probability of a link being faulty is thereby learned from the outlier flows that traverse them. Figure 2 illustrates this learning algorithm replacing stage 4 of our NLOF pipeline.

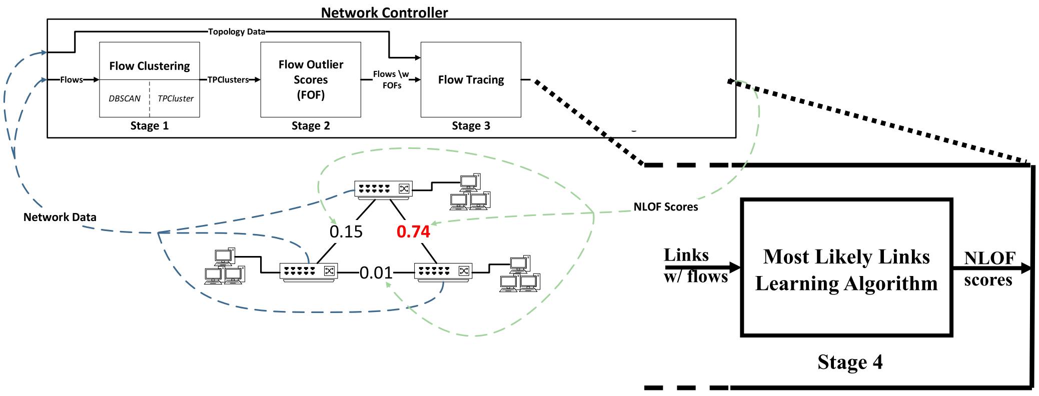


FIGURE 2. Our new NLOF:MLL replaces stage 4 of our original NLOF with our most likely links learning algorithm.

We use an intermediary weight for each link that is increased when an outlier flow traverses the link and decreased when a normal flow traverses the link. This weight is then normalized to become an outlier score for the link (i.e., a probability that the link is experiencing a fault). Let $\boldsymbol{w}$ be the vector of these intermediary weights, where each link in the network has a corresponding element in the vector and $N$ be the normalization factor; we will elaborate on this factor shortly. The NLOF:MLL score is computed as,

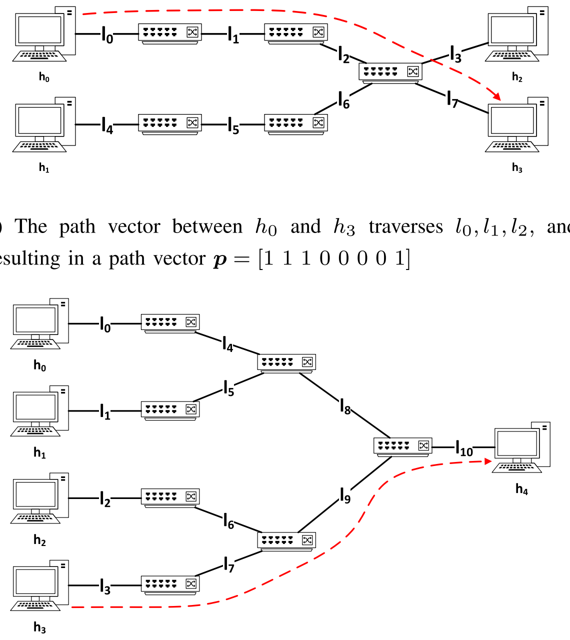
$$\hat{\mathcal{P}}\{y_i = 1\} = \frac{w_i}{N}. \tag{3}$$

### A. ALGORITHM OVERVIEW

Our algorithm executes periodically, each period being an epoch such as 1 minute. The algorithm is triggered at the conclusion of an epoch and uses the set of all flows that terminated during a time window that ends at the end of the epoch. We refer to that time window as the outlier window and its beginning can extend past the beginning of the epoch making it potentially larger than the epoch, see [10] for more detail. Let $\boldsymbol{w}^t$ be the weight vector for epoch $t$ and $\boldsymbol{\xi}^t$ be the initial value of the weight vector for epoch $t$ with an element in those two vectors for each link $i$, $w_i^t$ and $\xi_i^t$. For the first epoch, we set the initial weight vector ($\boldsymbol{\xi}^1$) to a vector with each element equal to a small non-zero value, $\boldsymbol{\epsilon}$. That small value represents no probability of a fault: $\boldsymbol{\xi}^1 = \boldsymbol{\epsilon}$. We discuss the setting of this initial weight vector for subsequent epochs ($\boldsymbol{\xi}^t$) in Section II-D.

For an epoch $t$, the link weight vector is initialized as $\boldsymbol{w}^t = \boldsymbol{\xi}^t$ and then updated for each flow $f$ in the set of flows that terminated during the outlier window. Each flow has an associated path vector expressing the path in the network that the flow traverses. Let $\boldsymbol{p}$ be the path vector for a flow with each element corresponding to a link in the network with the same ordering as the weight vectors. An element in the path vector of a flow is 1 if that link is traversed by the flow, and 0 if it is not.

Two examples of path vectors are visualized in Figure 3. In Figure 3a there are 4 hosts meaning that there would be 6 unique path vector values (4 choose 2) and in Figure 3b there are 5 hosts meaning there would be 10 unique path vector values (5 choose 2). Figure 3a shows the path taken by a flow between $h0$ and $h3$ which would traverse the links $l_0, l_1, l_2$, and $l_7$. The corresponding path vector would be $\boldsymbol{p}$ = [1 1 1 0 0 0 0 1]. Similarly, Figure 3b shows the path taken by a flow between hosts $h3$ and $h4$ which traverses the links $l_3, l_7, l_9$, and $l_{10}$ resulting in path vector $\boldsymbol{p}$ = [0 0 0 1 0 0 0 1 0 1 1]. The path vectors only need to be updated when the network topology changes or the path between two hosts changes; efficiently doing this is outside the scope of this work.



a) The path vector between $h_0$ and $h_3$ traverses $l_0, l_1, l_2$, and $l_7$ resulting in a path vector $\boldsymbol{p}$ = [1 1 1 0 0 0 0 1]

b) The path vector between $h_3$ and $h_4$ traverses $l_3, l_7, l_9$, and $l_{10}$ resulting in a path vector $\boldsymbol{p}$ = [0 0 0 1 0 0 0 1 0 1 1]

FIGURE 3. Path Vector Examples.

### B. UPDATING LINK WEIGHTS

For each epoch $t$ the link weights are updated for each flow in the set of flows in the outlier window. Let $f$ be the current flow used to update the link weights and $F_f$ be the outlier score of that flow (see [10] for how that flow outlier factor (FOF) score is determined), $\gamma$ be the threshold on the outlier score for declaring a flow as a performance-degraded outlier, $p_i$ be the $i^{th}$ element of the flow's path vector $\boldsymbol{p}$, $w_i^t$ be the $i^{th}$ element of the link weight vector for epoch $t$, and $\lambda$ be the link weight decrease factor. For the current flow $f$ the link weight vector $\boldsymbol{w}$ is updated element-by-element using,

$$w_i^t := w_i^t + \delta \tag{4}$$

where

$$\delta = \mathbb{1}\big(F_f > \gamma\big)\frac{w_i^t p_i}{(\boldsymbol{w}^t)^T\boldsymbol{p}} - \mathbb{1}\big(F_f \leq \gamma\big)\min\big(\lambda w_i^t p_i, 1\big). \tag{5}$$

Note that $\mathbb{1}(\cdot)$ is the indicator function; the value of the indicator function is 1 if the inequality in the parentheses is true and 0 otherwise. The *left hand term* in the update expression, Eq. (5), is non-zero for an outlier flow and increases the link weight in proportion of the link weight to all of the link weights along the path of the flow. The *right hand term* in the update expression is non-zero for a normal flow and decreases the link weight by a multiplicative factor of $\lambda$; but will not decrease by more than 1. We discuss the setting of the hyper-parameter $\lambda$ in Section II-E.

## C. NORMALIZING LINK WEIGHTS TO PRODUCE AN OUTLIER SCORE

Let $\boldsymbol{a}_i^t$ be the set of outlier flows that traverse link $i$ during the outlier window for epoch $t$, $\boldsymbol{l}_i^t$ be the set of all flows that traverse link $i$ during the outlier window for epoch $t$ and $y_i$ be a binary random variable describing the state of link $i$ as faulty when equal to 1 and normal when equal to 0. The final link weights ($w_i^t$) can have a maximum value of the number of outlier flows that traverse that link plus its initial weight ($|\boldsymbol{a}_i^t| + \xi_i^t$) if that link received all of the probability for all of the outlier flows that traversed it. Thus, to produce a normalized weight value ($\acute{w}_i^t$) that is restricted to the range of 0 and 1 to represent the probability that link $\boldsymbol{l}_i^t$ is experiencing a fault we divide the weight value by $|\boldsymbol{a}_i^t| + \xi_i^t$. The normalized link weights are:

$$\acute{w}_i^t = \frac{w_i^t}{N} = \frac{w_i^t}{|\boldsymbol{a}_i^t| + \xi_i^t}. \tag{6}$$

An issue of using the number of outlier flows to normalize the weights for each link is that it ignores the ratio of outlier flows to normal flows ($\frac{|\boldsymbol{a}_i^t|}{|\boldsymbol{l}_i^t|}$). Meaning that if out of many flows only one was outlier but link $i$ received all of the probability then the estimation of the probability of it being the source of a fault would be 1. For a more accurate estimation of the probability that a link is experiencing a failure then it needs to have both a large proportion of the probability (i.e., a high value for $\frac{w_i^t}{|\boldsymbol{a}_i^t|}$), as well as a large proportion of the flows that traverse it to be outliers (i.e., a high value for $\frac{|\boldsymbol{a}_i^t|}{|\boldsymbol{l}_i^t|}$). Since we are bootstrapping the weight values for epoch $t$ using $\boldsymbol{\xi}^t$, that value must be added to all the terms that do not already include it (i.e., $|\boldsymbol{a}_i^t|$ and $|\boldsymbol{l}_i^t|$). Thus, the final normalized weights ($\acute{w}_i^t$) and subsequently the estimation for the probability that a link is a source of a fault ($\hat{\mathcal{P}}\{y_i = 1\}$) is given by,

$$\acute{w}_i^t = \frac{w_i^t}{\big(|\boldsymbol{a}_i^t| + \xi_i^t\big)}\frac{\big(|\boldsymbol{a}_i^t| + \xi_i^t\big)}{\big(|\boldsymbol{l}_i^t| + \xi_i^t\big)} = \frac{w_i^t}{|\boldsymbol{l}_i^t| + \xi_i^t} \tag{7}$$

and

$$\hat{\mathcal{P}}\{y_i = 1\} = \acute{w}_i^t = \frac{w_i^t}{|\boldsymbol{l}_i^t| + \xi_i^t}. \tag{8}$$

## D. LEARNING FROM PRIOR EPOCHS

Initializing all of the link weights to a vector of small values, $\boldsymbol{\epsilon}$ is a reasonable initial assumption as without any prior information it cannot be concluded that any link is experiencing a fault. However, since NLOF executes periodically on an epoch-by-epoch basis we can use the learning in prior epochs to initialize the weights for the current epoch. Let the superscript $t$ denote the current epoch, the superscript $t-1$ denote the previous epoch, and so on. We use the average of the normalized weights for a link from the last $k$ epochs to initialize the weights for the current epoch,

$$\xi_i^t = \frac{\sum_{j=1}^{k} \acute{w}_i^{(t-j)}}{k}. \tag{9}$$

## E. SELECTION OF HYPER-PARAMETERS: $\gamma$, $\lambda$, AND $k$

Our learning algorithm has three hyper-parameters: 1) $\gamma$, the outlier score threshold for classifying a flow as an outlier, 2) $\lambda$, the multiplicative factor used to decrease the link weight for a normal flow, and 3) $k$, the number of previous epochs to learn the initial link weights from.

Choosing a value for $\gamma$ can be a difficult task as there is not a single value that can be used in all situations. Due to the nature of how the flow outlier factor (FOF) score is computed there can be a large variation in FOF scores for outlier flows. Rather than select a static threshold, it is best to use the FOF scores that have already been computed to select a value for $\gamma$. If the assumption is made that most of the traffic that will be observed is normal then a reasonable value can be selected for $\gamma$ by taking the $n$-th percentile of the FOF scores in the clustering window, where $n$ is a high value. In our experiments $n = 95$ worked well.

We found that selecting a small positive non-zero value for $\lambda$ worked well; in our experiments 0.1 provided good performance. A small positive value for $\lambda$ sufficiently prevents weight values from growing too quickly when there are consistent normal flows while not excessively diminishing the weight values.

Although leveraging insight from learning in prior epochs is useful, it is important to adapt quickly to changing conditions; so we recommend setting $k$ to either 1 or 2. This recommendation matches guidance in the general forecasting literature whereby auto-regressive models of order 2 or less are sufficient [23]. In our experiments we used $k = 1$.

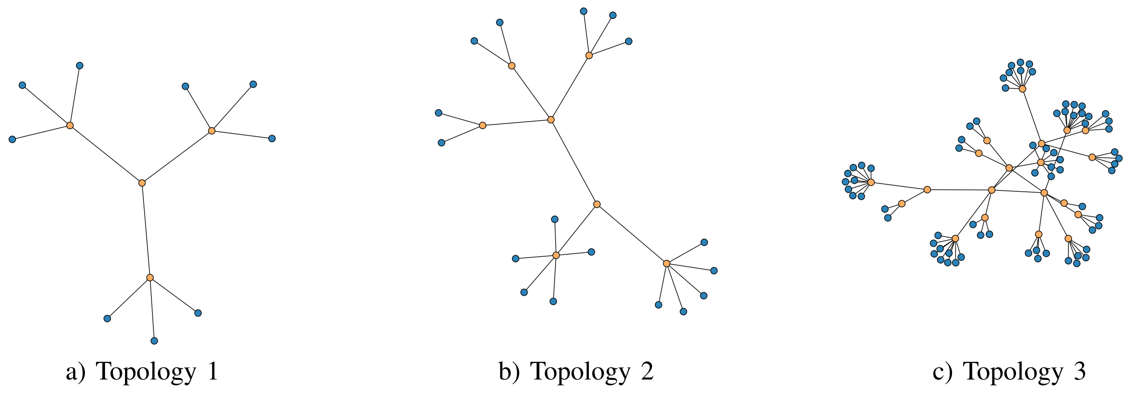


FIGURE 4. Topologies used in experiments. A blue node denotes a host and orange node denotes a switch.

## III. EXPERIMENTAL PLAN

Our experimental plan is designed to reveal the classification performance differences between two fault detection techniques from the existing literature (007 [22], and abrupt change detection (ACD) [24]), our original NLOF, and our NLOF:MLL proposed in this manuscript. Our NLOF:MLL contains an algorithm that learns the probability that a link is experiencing a fault. We have designed experiments to provide a high-level of realism and the ability to check for statistically significant performance differences. This is achieved with four strategies.

*First*, we use Mininet [25] as our experimental platform. Mininet uses process virtualization and network namespaces that are available in most recent Linux kernels to emulate network hardware utilizing real Linux network protocol stacks. Each host is a bash process that is created inside a network namespace so that each host has its own network interfaces and only has visibility of its own processes. Switches are implemented using software-based switches such as Open vSwitch that connect to hosts and other switches using virtual Ethernet pairs.

*Second*, we utilize network topologies derived from guidance for enterprise networks from the two largest equipment manufacturers: Cisco and Juniper. Three different network topologies representing three different sizes of enterprise networks are used. Three different loads were presented to the network: 0.1 Gbps, 0.5 Gbps, and 1 Gbps. We provide details on these topologies in Section III-A.

*Third*, we utilize generally accepted synthetic traffic models. We generate two types of flows, the first type are short-life constant bit rate (CBR) flows and the second type are long-life variable bit rate (VBR) flows with the self-similarity property. Flow data are captured using PMACCT [26] by monitoring all the traffic that traverses the switch with the highest centrality in the network. We provide more detail on our traffic models in Section III-C.

*Lastly*, we utilize repeated experiments whereby we reseed our pseudo-random number generators, use the same random network scenario for each of the four fault detection techniques, and apply inferential statistics to check for statistically significant performance differences. Six repeated experiments were conducted for each combination of topology and network load for a total of 54 experiments. During each of the repeated experiments, network flows and faults were randomly generated on the selected topology with the selected network load. Each of these experiments were 2 hours. During each repeated experiment between 1 and 4 link faults are created at random. Faults are emulated by adjusting the packet error rate of a link between 5% and 15%. The start of the fault is randomly selected and the duration of each fault is randomly selected to be between 500 and 1500 seconds. Shorter fault durations or a lower packet rate seem too inconsequential to consider in our experiments. In summary, pseudo-random number generation is used in each repeated experiment for the selection of:

1) Number, start time, duration, and location of faults
2) Start time, and duration of flows
3) Source and destination of flows
4) Burst lengths and individual packet sizes in a packet burst for a long-life VBR flow
5) Off period duration for a long-life VBR flow

The four fault detection techniques are compared using the generally accepted classification metrics of recall, precision, $F_1$-score, and time-to-detection (TTD) over the 54 experiments. Recall provides a measure of the fraction of true positives detected and precision provides a measure of false positives; both are important so the $F_1$-score is the harmonic mean of both those measures. Note that for the abrupt change detection technique its performance will be evaluated based on detection rather than localization as it was only designed to detect faults and not localize them. Confidence intervals are computed to determine statistically significant differences. A confidence interval is an interval with which we are confident (e.g., with 95% confidence) that the true measure is contained. Pairwise t-tests are utilized when confidence intervals are insufficient to test for statistical significance.

### A. TOPOLOGY SELECTION

The three topologies used in our experiments are shown in Figure 4. These topologies were designed using the medium-size enterprise design guides from Cisco [27] and

Juniper [28]. Both guides suggest a 3-layer architecture comprising of core, distribution, and access layers. The guides suggest collapsing the core and distribution layers for smaller networks. Topology 1 has collapsed core and distribution layers as the center switch; the other switches act as the access layer. Topology 2 has two center switches as the core and distribution layer. Topology 3 has 3 central switches as the core and distribution layer. While NLOF may be suitable for localizing faults on other topologies such as a mesh topology, that is reserved for future work. Due to computational restrictions larger topologies were not considered.

As a network grows in size, there is an increasing number of transmission links for which we need flows to transit to have the effect of faults on those links affect the flow data. The collected flows need to transit a fraction of the links that provides the fault coverage we desire (let's say 95%). All of those flows may be observable from a rather small subset of switches in the network regardless of its size depending on the structure of the network. As an example, if a gateway switch carries flows that traverse the desired coverage fraction of the network, then flows only need to be observed at that single switch. The computational complexity of our algorithm will increase with network size. The size of the weight and path vectors are proportional to the size of the network. The time complexity of the weight updating method of NLOF:MLL is linear, i.e., $\mathcal{O}(n)$; it will scale favorably to large networks. However, NLOF:MLL includes flow outlier detection, which currently uses clustering methods that have higher time complexity, typically $\mathcal{O}(n \log n)$. Note, the input size $n$ is relative to the number of flows observed which is likely to grow with the physical size of the network.

A divide and conquer approach is the best method to detect faults in a very large network. Divide the network into subnetworks and have an instance of our algorithm execute to detect faults on each subnetwork. The computational complexity primarily depends on the number of flows within the clustering and outlier windows. The number of flows in those windows is not easily related to the physical size of the network and could change significantly over time. For that reason, we recommend a trial and error method to determine when to employ the divide and conquer strategy.

### *B. ALGORITHM HYPER-PARAMETERS*

In our experiments we used the following algorithm hyper-parameters: 95th percentile of FOF scores for $\gamma$, 0.1 for $\lambda$, and 1 for $k$.

### *C. TRAFFIC GENERATION*

To generate traffic a total load is presented to the network by generating flows between hosts in the network. The total load presented to the network is constant and is defined by the emulation parameter that controls the load presented to the network ($\beta$) in bits per second (bps). To maintain a constant load a number of active flows will be generated that will supply that load. The number of active flows at any given time during an emulation is 2/3 the number of hosts ($|\boldsymbol{h}|$) on the network, e.g., if there are 9 hosts on the network then there will be 6 active flows at any given time. Each of the active flows will be one of two different types of flows; a short-life CBR flow, or a long-life VBR flow. VBR flows are generated using a self-similar model derived from the one presented in [29]; packet sizes follow the IMIX distribution. Self-similar models are shown to accurately model Ethernet traffic [30]. Half of the active flows are CBR flows and the other half are VBR flows.

#### 1) SHORT-LIFE CONSTANT BIT RATE (CBR) FLOWS

CBR flows are implemented using a Python socket connection between two hosts in Mininet. The duration is randomly selected between a range of 1 and 15 seconds. The total number of bytes to be sent are split into equal-sized packets sent out with a deterministic inter-packet delay.

#### 2) LONG-LIFE VARIABLE BIT RATE (VBR) FLOWS

VBR flows are also implemented using Python socket connections between two hosts in Mininet. The self-similarity of each flow is determined by the Hurst parameter, which is set to 0.75. Each flow has a queue thread that is fed by 32 streams. Each of the streams has its own thread which generates bursts of packets that are fed into a queue for the flow. The number of packets in a burst are sampled from a Pareto distribution with shape $\alpha = 3$ and scale $m = 10$. The delay between bursts of packets is also sampled from a Pareto distribution with $\alpha = 3 - (2 * 0.75)$ which means the delay exhibits the property of self-similarity. The scale parameter ($m$) is then computed given all the other information to achieve the desired average bit rate over a long period. A separate thread handles sending the data in the queue over a Python socket connection, which will check the queue and send the data in the queue for the duration of the flow. The duration of the VBR flows is selected randomly between 60 and 600 seconds. The size of each packet sent is randomly selected from the IMIX packet size distribution.

## IV. EXPERIMENTAL PERFORMANCE ANALYSIS

We compared the four fault detection techniques using the classification performance measures of recall, precision, $F_1$-score, and time-to-detection (TTD). We discuss each of these performance measures in the next four sub-sections.

### *A. RECALL*

Figure 5 shows the average recall for each technique over the 54 experiments. The results also include the 95% confidence interval computed over the 54 experiments. The average recall of 007, abrupt change detection (ACD), NLOF, and NLOF:MLL were 0.986, 0.666, 0.504, and 0.823, respectively. 007 has a much higher recall than any of the other techniques as it is able to localize almost all of the faults that occurred. The recall performance of 007 does have a caveat, it declares most links as faulty resulting in many true positives but also many false positives. When only considering the recall 007 performs deceptively well, but the overall

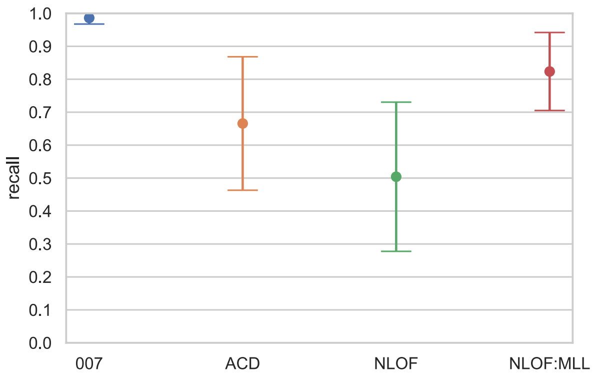


FIGURE 5. Average recall of each technique over all 54 experiments, with 95% confidence intervals.

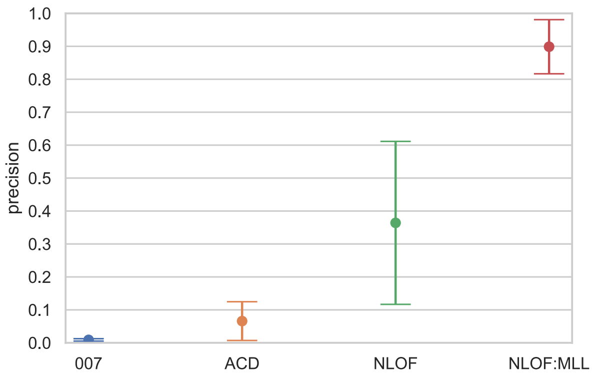


FIGURE 6. Average precision of each technique over all 54 experiments, with 95% confidence intervals.

TABLE 1. P-values for two-sided pairwise t-tests to compare the average recall between each pair of techniques, each value in bold indicates a statistically significant difference with over 95% confidence.

| | **NLOF** | **NLOF:MLL** | **007** | **ACD** |
|---|---|---|---|---|
| **NLOF** | NaN | **1.86e-05** | **1.17e-08** | 5.22e-02 |
| **NLOF:MLL** | **1.86e-05** | NaN | **3.51e-04** | 4.84e-02 |
| **007** | **1.17e-08** | **3.51e-04** | NaN | **4.17e-05** |
| **ACD** | 5.22e-02 | 4.84e-02 | **4.17e-05** | NaN |

TABLE 2. P-values for two-sided pairwise t-tests to compare the average precision between each pair of techniques, each value in bold indicates a statistically significant difference with over 95% confidence.

| | **NLOF** | **NLOF:MLL** | **007** | **ACD** |
|---|---|---|---|---|
| **NLOF** | NaN | **1.34e-09** | **3.32e-10** | **7.54e-08** |
| **NLOF:MLL** | **1.34e-09** | NaN | **2.74e-44** | **1.20e-39** |
| **007** | **3.32e-10** | **2.74e-44** | NaN | **1.91e-07** |
| **ACD** | **7.54e-08** | **1.20e-39** | **1.91e-07** | NaN |

performance will be made clear in the following subsections. NLOF:MLL has a slightly lower recall but is still able to localize most faults that occurred and provides a significant improvement compared to NLOF. The abrupt change detection method performed poorly even when only evaluated on its ability to detect faults rather than localize them.

A two-sided pairwise t-test was conducted to uncover statistically significant differences in average recall between each pair of techniques. The results are shown in Table 1. The t-tests show that most of the techniques are statistically significantly different with over 99% confidence. The exception is when comparing abrupt change detection with NLOF and NLOF:MLL; p-values were 5.22e-02 and 4.84e-02, respectively.

### B. PRECISION

Figure 6 shows the average precision for each technique over the 54 experiments. The results also include the 95% confidence interval computed over the 54 experiments. The average precision of 007, abrupt change detection (ACD), NLOF, and NLOF:MLL were 0.009, 0.066, 0.364, and 0.899, respectively. While 007 had a nearly perfect recall, its precision was near 0. This indicates that 007 is producing many false positives since 007 declares any link with more than 1% of the votes as faulty resulting in very high recall but also very low precision. The approach taken by 007 is not robust to noise; i.e., when a retransmission is caused by something other than a faulty link. This observation is corroborated by the Omnimon performance comparison with 007, see [31]. The abrupt change detection has the same problem generating many false positives likely due to abrupt changes in the traffic behavior when there is no fault, which could happen in the case of congestion or large bursts of traffic. NLOF:MLL has significantly higher precision than all of the other techniques, meaning that NLOF:MLL can localize most of the faults while producing much fewer false positives than the other techniques.

A two-sided pairwise t-test was conducted to check for statistically significant differences in average precision between each pair of techniques. The results are shown in Table 2. The results of the t-tests show that there is a statistically significant difference between each pair of techniques with well over 99% confidence. Note that there is a statistically significant difference in the average precision between NLOF and NLOF:MLL indicating that NLOF:MLL did address the issue of NLOF generating false positives when a faulty core link is present.

### C. $F_1$-SCORE

Figure 7 shows the average $F_1$-score for each technique over the 54 experiments. The results also include the 95% confidence interval computed over the 54 experiments. Given that the $F_1$-score is the harmonic mean of precision and recall the results show exactly what is expected with 007, abrupt change detection (ACD), NLOF, and NLOF:MLL having average $F_1$-scores of 0.018, 0.110, 0.350, and 0.837, respectively.

A two-sided pairwise t-test was conducted to check for statistically significant differences in the average $F_1$-score between each pair of techniques. The results are shown in Table 3. The results of the t-tests show that there is a statistically significant difference between each pair of techniques with well over 99% confidence. Given that NLOF:MLL has a much higher average $F_1$-score than the other techniques in

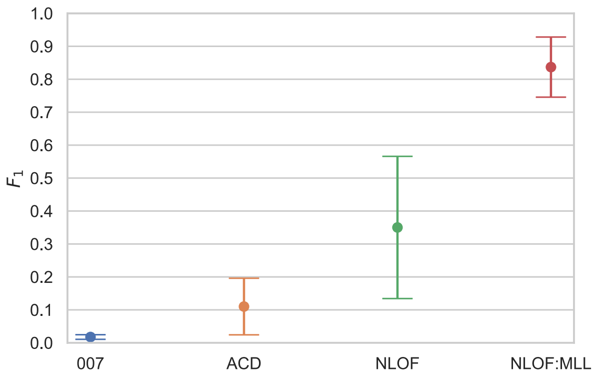


FIGURE 7. Average $F_1$-score of each technique over all 54 experiments, with 95% confidence intervals.

TABLE 3. P-values for two-sided pairwise t-tests to compare the average $F_1$-score between each pair of techniques, each value in bold indicates a statistically significant difference with over 95% confidence.

| | NLOF | NLOF:MLL | 007 | ACD |
|---|---|---|---|---|
| **NLOF** | NaN | **2.43e-11** | **1.03e-10** | **2.62e-06** |
| **NLOF:MLL** | **2.43e-11** | NaN | **2.87e-39** | **7.58e-27** |
| **007** | **1.03e-10** | **2.87e-39** | NaN | **3.35e-08** |
| **ACD** | **2.62e-06** | **7.58e-27** | **3.35e-08** | NaN |

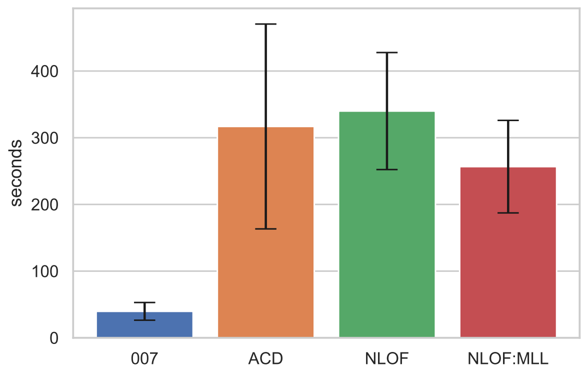


FIGURE 8. Average Time To Detection (TTD), in seconds, of each technique over all 54 experiments.

this wide set of experiments, we conclude that NLOF:MLL can localize most faults without producing too many false positives.

### D. TIME TO DETECTION (TTD)

Figure 8 shows the average time to detection over all of the correctly localized/detected faults for each technique. 007, abrupt change detection (ACD), NLOF, and NLOF:MLL had an average Time To Detection (TTD), in seconds, of 40, 317, 340 and 257, respectively. From the results it is clear that 007 has the advantage when it comes to how quickly it can localize a fault. This is due to the data that is used by each technique. 007 uses retransmissions that are reported immediately when they occur. Comparatively, NLOF:MLL uses flow records that are not observable until a flow has terminated and its record is written to persistent storage such as a database. The abrupt change detection technique, NLOF, and NLOF:MLL all have reasonable average TTD values as

TABLE 4. P-values for two-sided t-tests to compare the average time to detection (TTD) between each pair of techniques, each value in bold indicates a statistically significant difference with over 95% confidence.

| | NLOF | NLOF:MLL | 007 | ACD |
|---|---|---|---|---|
| **NLOF** | 1.00e+00 | **3.53e-03** | **6.66e-22** | 6.05e-01 |
| **NLOF:MLL** | **3.53e-03** | 1.00e+00 | **1.79e-22** | 1.58e-01 |
| **007** | **6.66e-22** | **1.79e-22** | 1.00e+00 | **1.42e-10** |
| **ACD** | 6.05e-01 | 1.58e-01 | **1.42e-10** | 1.00e+00 |

the fault can be localized within a few minutes of the fault beginning to affect traffic.

A two-sided t-test was performed between each pair of techniques to check for statistically significant differences in average TTD. The p-values resulting from the t-tests are shown in Table 4. There is a statistically significant difference in the average TTD for most pairs of techniques. The exceptions are abrupt change detection compared to either NLOF or NLOF:MLL. An important observation is that the average TTD of NLOF:MLL is smaller than that of NLOF. The smaller TTD provided by NLOF:MLL is a result of the running average method that is used by NLOF:MLL allowing NLOF scores to increase more quickly.

## V. CONCLUSION

In conclusion, our NLOF:MLL technique outperforms our original NLOF, abrupt change detection, and 007 on precision and $F_1$-score in our wide ranging Mininet experiments; the performance differences are statistically significant. 007 had higher recall but at the expense of very poor precision as a result of excessive false positives. The time-to-detection of NLOF:MLL was competitive with abrupt change detection but higher than 007. NLOF:MLL improved the time-to-detection over our original NLOF.

In future work we can explore strategies to reduce the time-to-detection. Other paths for future work can explore the use of additional flow features, and techniques other than averaging to incorporate learning from prior epochs.

## APPENDIX

NLOF uses clustering and outlier detection in its first two stages to identify flows that are performance outliers; performance-degraded flows. The performance of those two stages has sensitivities that we do not analyze in this work. In the third stage, NLOF associates flows to the links they traverse. In the fourth and final stage, NLOF assigns an outlier score to each link as a ratio of outlier flows to all flows traversing that link. That ratio is sensitive to the flows that are observed (due to flow export locations). We analyze this particular sensitivity in this Appendix.

Recall from Section II, $\boldsymbol{a}_i$ is the set of outlier flows that traverse link $i$, $\boldsymbol{l}_i$ is the set of all flows that traverse link $i$. Therefore, our original NLOF scores are computed as,

$$\hat{\mathcal{P}}\{y_i = 1\} = \frac{|\boldsymbol{a}_i|}{|\boldsymbol{l}_i|}. \tag{10}$$

To facilitate the sensitivity analysis, we will assume all flows traversing a faulty link will be properly labeled as

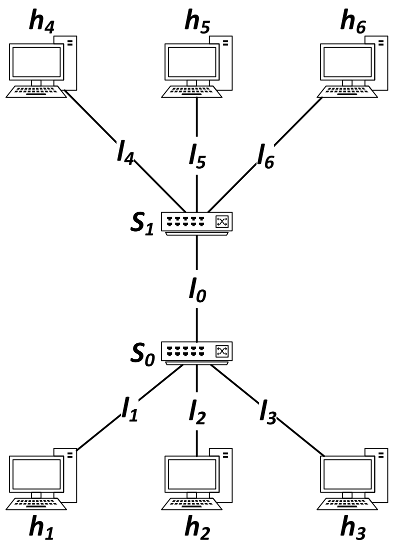


FIGURE 9. **Topology used to describe the sensitivity of NLOF to probe placement and to the traffic matrix.**

a performance outlier (i.e., performance-degraded). We will also compute NLOF scores by considering flows traversing the same path as an aggregate: flow-producing host-pairs. Specifically, we assume every pair of hosts produces flows that traverse the same path; a reasonable assumption in the absence of route flapping. We further assume a balanced traffic matrix: pairs of hosts produce the same number of flows over time. Those assumptions allow computing the NLOF scores using numbers of host-pairs. Host-pairs whose flows would traverse a faulty link would increment the count of $\boldsymbol{a}_i$ by 1 for each link $i$ they traverse; whether the link has a fault or not. All host pairs whose flows would traverse a link $i$ would be counted in $\boldsymbol{l}_i$, the NLOF score for link $i$ is the ratio of those two values as can be seen in the equation above. Using the granularity of flow-producing host-pairs simplifies the sensitivity analysis.

Figure 9 shows the simple network topology we are considering in the following NLOF outlier score sensitivity analysis. This topology consists of 2 switches, 6 hosts, and 7 links; there are 15 combinations of pairs of hosts. We explore two fault examples: *a*) a fault on edge link $l_1$, and *b*) a fault on core link $l_0$.

With the topology in Figure 9 we have two switches; so we consider three possibilities: *i*) exporting from both switches, *ii*) exporting only from switch $S_0$, and *iii*) exporting only from switch $S_1$.

### A. EXAMPLE A: FAULT ON EDGE LINK $L_1$

With a fault on edge link $l_1$, all host-pairs including $h1$ (five total) will be performance-degraded the other ten host-pairs will not be. Full flow visibility (i.e., flow export from both Switches $S_0$ and $S_1$) means the NLOF score for link $l_1$ is 1 and less than 1 for the other 5 links as they will have host-pairs other than those that traverse the faulty link. Specifically, the scores will be 0.2 for the other edge links and 0.33 for the core link $l_0$; thereby clearly distinguishing the faulty link with the NLOF score. As an example, for link $l_2$ there are 5 observable flow-producing host-pairs but only one is performance degraded $h_1$:$h_2$ for a ratio of 0.2.

Exporting flows only from Switch $S_0$ permits observing all of the flows for host-pairs that include $h_1$ and results in NLOF scores similar to full visibility.

If flows are only exported from Switch $S_1$ not all of the flows for host-pairs including $h_1$ are visible. NLOF scores are 0 for links $l_2$ and $l_3$ because the performance-degraded host-pairs $h_1$:$h_2$ and $h_1$:$h_3$ are not observable, otherwise the results are the same as the other two flow export scenarios. Importantly, the NLOF score of the faulty link is still clearly distinguished.

This analysis suggests faults on edge links are robust to flow export location(s). This matches our experimental findings reported in [10]. In practice, not all flows traversing a faulty link will be labeled as an outlier either due to the fault not consistently degrading flow performance or the clustering and outlier detection not properly labeling the performance-degraded flow. As a result, we found that setting a lower threshold (i.e., 0.66) worked well in our experiments.

### B. EXAMPLE B: FAULT ON CORE LINK $L_0$

With a fault on the core link $l_0$, all host-pairs that cross through the two switches in the topology of Figure 9 will be performance-degraded (nine total), the other six host-pairs will not be. Full flow visibility results in the NLOF score being 1 for link $l_0$ but 0.6 for the other links; all links have high NLOF scores. However, a threshold of 0.66 isolates the faulty link $l_0$. See Table 5 for details on the NLOF score computations.

If flows are only exported from one of the two switches (either Switch $S_0$ or $S_1$) then several links will have NLOF scores of 1 in addition to the faulty link. Thereby causing the faulty link to be indistinguishable from those other links without considering additional criteria. In our original NLOF we utilized topological relationships to make the distinction making the detection rule quite complicated. Tables 6 and 7 show the details of the NLOF score computations.

### C. GENERALIZATION

Let's assume a single faulty link, let $\boldsymbol{l}_f$ be the set of flows traversing that faulty link, and $\boldsymbol{a}_f$ be the set of outlier flows traversing that faulty link, then the NLOF score of the faulty link ($N_f$) from Eq. (10) is,

$$N_f = \frac{|\boldsymbol{a}_f|}{|\boldsymbol{l}_f|}. \tag{11}$$

Eq. (11) by its definition is the proportion of flows traversing the faulty link that are performance-degraded.

The intersection of the set of flows traversing the faulty link and a non-faulty link $i$, (i.e., $\boldsymbol{l}_i \cap \boldsymbol{l}_f$), are the flows that traverse both the faulty link and link $i$. We will refer to these as the "common flows" among the faulty link and link $i$.

TABLE 5. Core link error ($l_0$): Flow export from both switches (i.e., full visibility).

| link | total observable flow-producing host-pairs | performance-degraded observable flow-producing host-pairs | NLOF score |
|---|---|---|---|
| $l_0$ | $(h_1, h_4)$, $(h_1, h_5)$, $(h_1, h_6)$,$(h_2, h_4)$, $(h_2, h_5)$, $(h_2, h_6)$ $(h_3, h_4)$, $(h_3, h_5)$, $(h_3, h_6)$ | $(h_1, h_4)$, $(h_1, h_5)$, $(h_1, h_6)$,$(h_2, h_4)$, $(h_2, h_5)$, $(h_2, h_6)$ $(h_3, h_4)$, $(h_3, h_5)$, $(h_3, h_6)$ | 9/9 = 1 |
| $l_1$ | $(h_1, h_2)$, $(h_1, h_3)$, $(h_1, h_4)$, $(h_1, h_5)$, $(h_1, h_6)$ | $(h_1, h_4)$, $(h_1, h_5)$, $(h_1, h_6)$ | 3/5 = 0.6 |
| $l_2$ | $(h_2, h_1)$, $(h_2, h_3)$, $(h_2, h_4)$, $(h_2, h_5)$, $(h_2, h_6)$ | $(h_2, h_4)$, $(h_2, h_5)$, $(h_2, h_6)$ | 3/5 = 0.6 |
| $l_3$ | $(h_3, h_1)$, $(h_3, h_2)$, $(h_3, h_4)$, $(h_3, h_5)$, $(h_3, h_6)$ | $(h_3, h_4)$, $(h_3, h_5)$, $(h_3, h_6)$ | 3/5 = 0.6 |
| $l_4$ | $(h_4, h_1)$, $(h_4, h_2)$, $(h_4, h_3)$, $(h_4, h_5)$, $(h_4, h_6)$ | $(h_4, h_1)$, $(h_4, h_2)$, $(h_4, h_3)$ | 3/5 = 0.6 |
| $l_5$ | $(h_5, h_1)$, $(h_5, h_2)$, $(h_5, h_3)$, $(h_5, h_4)$, $(h_5, h_6)$ | $(h_5, h_1)$, $(h_5, h_2)$, $(h_5, h_3)$ | 3/5 = 0.6 |
| $l_6$ | $(h_6, h_1)$, $(h_6, h_2)$, $(h_6, h_3)$, $(h_6, h_4)$, $(h_6, h_5)$ | $(h_6, h_1)$, $(h_6, h_2)$, $(h_6, h_3)$ | 3/5 = 0.6 |

TABLE 6. Core link error ($l_0$): Flow export from only switch $S_0$.

| link | total observable flow-producing host-pairs | performance-degraded observable flow-producing host-pairs | NLOF score |
|---|---|---|---|
| $l_0$ | $(h_1, h_4)$, $(h_1, h_5)$, $(h_1, h_6)$,$(h_2, h_4)$, $(h_2, h_5)$, $(h_2, h_6)$ $(h_3, h_4)$, $(h_3, h_5)$, $(h_3, h_6)$ | $(h_1, h_4)$, $(h_1, h_5)$, $(h_1, h_6)$,$(h_2, h_4)$, $(h_2, h_5)$, $(h_2, h_6)$ $(h_3, h_4)$, $(h_3, h_5)$, $(h_3, h_6)$ | 9/9 = 1 |
| $l_1$ | $(h_1, h_2)$, $(h_1, h_3)$, $(h_1, h_4)$, $(h_1, h_5)$, $(h_1, h_6)$ | $(h_1, h_4)$, $(h_1, h_5)$, $(h_1, h_6)$ | 3/5 = 0.6 |
| $l_2$ | $(h_2, h_1)$, $(h_2, h_3)$, $(h_2, h_4)$, $(h_2, h_5)$, $(h_2, h_6)$ | $(h_2, h_4)$, $(h_2, h_5)$, $(h_2, h_6)$ | 3/5 = 0.6 |
| $l_3$ | $(h_3, h_1)$, $(h_3, h_2)$, $(h_3, h_4)$, $(h_3, h_5)$, $(h_3, h_6)$ | $(h_3, h_4)$, $(h_3, h_5)$, $(h_3, h_6)$ | 3/5 = 0.6 |
| $l_4$ | $(h_4, h_1)$, $(h_4, h_2)$, $(h_4, h_3)$ | $(h_4, h_1)$, $(h_4, h_2)$, $(h_4, h_3)$ | 3/3 = 1 |
| $l_5$ | $(h_5, h_1)$, $(h_5, h_2)$, $(h_5, h_3)$ | $(h_5, h_1)$, $(h_5, h_2)$, $(h_5, h_3)$ | 3/3 = 1 |
| $l_6$ | $(h_6, h_1)$, $(h_6, h_2)$, $(h_6, h_3)$ | $(h_6, h_1)$, $(h_6, h_2)$, $(h_6, h_3)$ | 3/3 = 1 |

TABLE 7. Core link error ($l_0$): Flow export from only switch $S_1$.

| link | total observable flow-producing host-pairs | performance-degraded observable flow-producing host-pairs | NLOF score |
|---|---|---|---|
| $l_0$ | $(h_1, h_4)$, $(h_1, h_5)$, $(h_1, h_6)$,$(h_2, h_4)$, $(h_2, h_5)$, $(h_2, h_6)$ $(h_3, h_4)$, $(h_3, h_5)$, $(h_3, h_6)$ | $(h_1, h_4)$, $(h_1, h_5)$, $(h_1, h_6)$,$(h_2, h_4)$, $(h_2, h_5)$, $(h_2, h_6)$ $(h_3, h_4)$, $(h_3, h_5)$, $(h_3, h_6)$ | 9/9 = 1 |
| $l_1$ | $(h_1, h_4)$, $(h_1, h_5)$, $(h_1, h_6)$ | $(h_1, h_4)$, $(h_1, h_5)$, $(h_1, h_6)$ | 3/3 = 1 |
| $l_2$ | $(h_2, h_4)$, $(h_2, h_5)$, $(h_2, h_6)$ | $(h_2, h_4)$, $(h_2, h_5)$, $(h_2, h_6)$ | 3/3 = 1 |
| $l_3$ | $(h_3, h_4)$, $(h_3, h_5)$, $(h_3, h_6)$ | $(h_3, h_4)$, $(h_3, h_5)$, $(h_3, h_6)$ | 3/3 = 1 |
| $l_4$ | $(h_4, h_1)$, $(h_4, h_2)$, $(h_4, h_3)$, $(h_4, h_5)$, $(h_4, h_6)$ | $(h_4, h_1)$, $(h_4, h_2)$, $(h_4, h_3)$ | 3/5 = 0.6 |
| $l_5$ | $(h_5, h_1)$, $(h_5, h_2)$, $(h_5, h_3)$, $(h_5, h_4)$, $(h_5, h_6)$ | $(h_5, h_1)$, $(h_5, h_2)$, $(h_5, h_3)$ | 3/5 = 0.6 |
| $l_6$ | $(h_6, h_1)$, $(h_6, h_2)$, $(h_6, h_3)$, $(h_6, h_4)$, $(h_6, h_5)$ | $(h_6, h_1)$, $(h_6, h_2)$, $(h_6, h_3)$ | 3/5 = 0.6 |

Let $\alpha$ be the ratio of those "common flows" to all the flows traversing link $i$,

$$\alpha = \frac{|\boldsymbol{l}_i \cap \boldsymbol{l}_f|}{|\boldsymbol{l}_i|}. \tag{12}$$

Since the faulty link is the source of the outlier flows traversing link $i$,

$$|\boldsymbol{a}_i| \leq N_f |\boldsymbol{l}_i \cap \boldsymbol{l}_f| \tag{13}$$

and it therefore follows that the NLOF score ($N_i$) for a link $i$ is,

$$N_i = \frac{|\boldsymbol{a}_i|}{|\boldsymbol{l}_i|} \leq \frac{N_f |\boldsymbol{l}_i \cap \boldsymbol{l}_f|}{|\boldsymbol{l}_i|}. \tag{14}$$

Using the ratio of those "common flows", $\alpha$,

$$N_i \leq \alpha N_f \tag{15}$$

and therefore has a range from 0 to $N_f$.

As a result, any link's NLOF score can approach the NLOF score of the faulty link. In the extreme case, when all of the flows traversing the faulty link are the same as those traversing the link $i$ where $i$ is not the faulty link, unfortunately $N_i = N_f$.

It is important that the ratio of "common flows" is as low as possible. There are two factors that affect that ratio $\alpha$:

1) the naturally occuring *traffic matrix*
2) flow observability impacted by the *flow export location*

Our new NLOF:MLL algorithm mitigates these NLOF score sensitivies by making Equation (14) not relevant because the NLOF score is no longer the ratio of outlier flows to all flows traversing the link.

## REFERENCES

[1] A. Gupta, "Network management: Current trends and future perspectives," *J. Netw. Syst. Manag.*, vol. 14, no. 4, pp. 483–491, Dec. 2006.

[2] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie, "A survey on software-defined networking," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 1, pp. 27–51, 1st Quart., 2015.

[3] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015.

[4] S. Ayoubi et al., "Machine learning for cognitive network management," *IEEE Commun. Mag.*, vol. 56, no. 1, pp. 158–165, Jan. 2018.

[5] C. Benzaid and T. Taleb, "AI-driven zero touch network and service management in 5G and beyond: Challenges and research directions," *IEEE Netw.*, vol. 34, no. 2, pp. 186–194, Mar./Apr. 2020.

[6] J. Gallego-Madrid, R. Sanchez-Iborra, P. M. Ruiz, and A. F. Skarmeta, "Machine learning-based zero-touch network and service management: A survey," *Digit. Commun. Netw.*, vol. 8, no. 2, pp. 105–123, Apr. 2022.

[7] E. Coronado et al., "Zero touch management: A survey of network automation solutions for 5G and 6G networks," *IEEE Commun. Surveys Tuts.*, vol. 24, no. 4, pp. 2535–2578, 4th Quart., 2022.

[8] C. Mendoza, V. Dasari, and M. P. McGarry, "Detecting network soft-failures with the network link outlier factor (NLOF)," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Dec. 2019, pp. 3978–3983.

[9] C. Mendoza, V. Dasari, and M. P. McGarry, "The network link outlier factor (NLOF)," in *Proc. Disruptive Technol. Inf. Sci. IV*, Apr. 2020, Art. no. 114190O.

[10] C. Mendoza and M. P. McGarry, "The network link outlier factor (NLOF) for fault localization," *IEEE Open J. Commun. Soc.*, vol. 1, pp. 1539–1550, 2020.

[11] R. A. Maxion and F. E. Feather, "A case study of Ethernet anomalies in a distributed computing environment," *IEEE Trans. Rel.*, vol. 39, no. 4, pp. 433–443, Oct. 1990.

[12] F. Feather, D. Siewiorek, and R. Maxion, "Fault detection in an Ethernet network using anomaly signature matching," *SIGCOMM Comput. Commun. Rev.*, vol. 23, no. 4, pp. 279–288, Oct. 1993.

[13] E. Kiciman and A. Fox, "Detecting application-level failures in component-based Internet services," *IEEE Trans. Neural Netw.*, vol. 16, no. 5, pp. 1027–1041, Sep. 2005.

[14] H. Hajji, "Statistical analysis of network traffic for adaptive faults detection," *IEEE Trans. Neural Netw.*, vol. 16, no. 5, pp. 1053–1063, Sep. 2005.

[15] G. W. Hart and A. T. Bouloutas, "Correcting dependent errors in sequences generated by finite-state processes," *IEEE Trans. Inf. Theory*, vol. 39, no. 4, pp. 1249–1260, Jul. 1993.

[16] A. T. Bouloutas, S. Calo, and A. Finkel, "Alarm correlation and fault identification in communication networks," *IEEE Trans. Commun.*, vol. 42, no. 234, pp. 523–533, Feb.–Apr. 1994.

[17] I. Rouvellou and G. W. Hart, "Automatic alarm correlation for fault identification," in *Proc. IEEE INFOCOM*, vol. 2, Apr. 1995, pp. 553–561.

[18] K. Appleby, G. Goldszmidt, and M. Steinder, "Yemanja—A layered fault localization system for multi-domain computing utilities," *J. Netw. Syst. Manag.*, vol. 10, no. 2, pp. 171–194, Jun. 2002.

[19] E. Athanasopoulou and C. N. Hadjicostis, "Probabilistic approaches to fault detection in networked discrete event systems," *IEEE Trans. Neural Netw.*, vol. 16, no. 5, pp. 1042–1052, Sep. 2005.

[20] A. Johnsson and C. Meirosu, "Towards automatic network fault localization in real time using probabilistic inference," in *Proc. IFIP/IEEE Int. Symp. Integr. Netw. Manag. (IM)*, May 2013, pp. 1393–1398.

[21] A. Johnsson, C. Meirosu, and C. Flinta, "Online network performance degradation localization using probabilistic inference and change detection," in *Proc. IEEE Netw. Oper. Manag. Symp. (NOMS)*, May 2014, pp. 1–8.

[22] B. Arzani et al., "007: Democratically finding the cause of packet drops," in *Proc. 15th USENIX Symp. Netw. Syst. Des. Implement. (NSDI)*, Apr. 2018, pp. 419–435.

[23] G. E. P. Box, G. M. Jenkins, and G. C. Reinsel, *Time Series Analysis: Forecasting and Control*. Hoboken, NJ, USA: Wiley, 2008.

[24] M. Thottan and C. Ji, "Proactive anomaly detection using distributed intelligent agents," *IEEE Netw.*, vol. 12, no. 5, pp. 21–27, Sep./Oct. 1998.

[25] "Mininet: Rapid prototyping for software defined networks." 2022. [Online]. Available: https://github.com/mininet/mininet

[26] P. Lucente. "PMACCT: IP traffic accounting." 2023. [Online]. Available: https://github.com/pmacct/pmacct

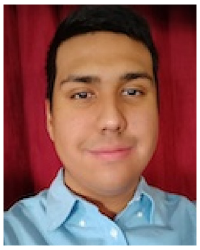
[27] "Medium enterprise design profile reference guide." Cisco. Dec. 2013. [Online]. Available: https://www.cisco.com/c/en/us/td/docs/solutions/Enterprise/Medium_Enterprise_Design_Profile/MEDP/chap2.html

[28] "Understanding the design of the midsize enterprise campus solution." Juniper. Nov. 2016. [Online]. Available: https://www.juniper.net/documentation/en_US/release-independent/nce/information-products/topic-collections/nce/nce-143-campus-solution-configuring/topic-93447.html

[29] G. Kramer, B. Mukherjee, and G. Pesavento, "IPACT a dynamic protocol for an Ethernet PON (EPON)," *IEEE Commun. Mag.*, vol. 40, no. 2, pp. 74–80, Feb. 2002.

[30] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson, "On the self-similar nature of Ethernet traffic (extended version)," *IEEE/ACM Trans. Netw.*, vol. 2, no. 1, pp. 1–15, Feb. 1994.

[31] Q. Huang, H. Sun, P. P. C. Lee, W. Bai, F. Zhu, and Y. Bao, "OmniMon: Re-architecting network telemetry with resource efficiency and full accuracy," in *Proc. Annu. Conf. ACM Spec. Interest Group Data Commun. Appl. Technol. Archit. Protocols Comput. Commun.*, Jul. 2020, pp. 404–421.



**CHRISTOPHER MENDOZA** received the Ph.D. degree in electrical and computer engineering from the University of Texas at El Paso in December 2021, under the supervision of Prof. M. P. McGarry. His research interests include automated network fault management using machine learning and natural language processing.

**MICHAEL P. MCGARRY** (Senior Member, IEEE) received the B.S. degree in computer engineering from the Polytechnic Institute of NYU and the M.S. and Ph.D. degrees in electrical engineering from Arizona State University in 2004 and 2007, respectively. He is an Associate Professor with the University of Texas at El Paso, where he has been on the faculty since 2010. From 2008 to 2010, he was an Assistant Professor with the University of Akron. His published works in optical access networks and software-defined networking have received over 2000 citations according to Google Scholar. One of his articles was a recipient of the IEEE Communications Society Best Tutorial Paper Award in 2009. He has served for several years and continues to serve as an Associate Editor for IEEE COMMUNICATIONS SURVEYS AND TUTORIALS as well as *Optical Switching and Networking* (Elsevier). His current research interest is in applying machine learning to automate communication network management.