Single-Link Failure Detection in All-Optical Networks Using Monitoring Cycles and Paths

Satyajeet S. Ahuja, Srinivasan Ramasubramanian, and Marwan Krunz
Department of ECE, University of Arizona, Tucson, AZ 85721
satyajeetahuja@gmail.com, {srini, krunz}@ece.arizona.edu

***Abstract*— In this paper, we consider the problem of fault localization in all-optical networks. We introduce the concept of monitoring cycles (MCs) and monitoring paths (MPs) for unique identification of single-link failures. MCs and MPs are required to pass through one or more monitoring locations. They are constructed such that any single-link failure results in the failure of a unique combination of MCs and MPs that pass through the monitoring location(s). For a network with only one monitoring location, we prove that three-edge connectivity is a necessary and sufficient condition for constructing MCs that uniquely identify any single-link failure in the network. For this case, we formulate the problem of constructing MCs as an integer linear program (ILP). We also develop heuristic approaches for constructing MCs in the presence of one or more monitoring locations. For an arbitrary network (not necessarily three-edge connected), we describe a fault localization technique that uses both MPs and MCs and that employs multiple monitoring locations. We also provide a linear-time algorithm to compute the minimum number of required monitoring locations. Through extensive simulations, we demonstrate the effectiveness of the proposed monitoring technique.**

***Index Terms*— All-optical networks, fault localization.**

I. Introduction

Optical networks have gained tremendous importance due to their ability to support very high data rates using the dense wavelength division multiplexing technology. With such high data rates, a brief service disruption in the operation of the network can result in the loss of a large amount of data. Commonly observed service disruptions are caused by fiber cuts, equipment failure, excessive bit errors, and human error. It is desired that these faults be uniquely identified and corrected at the physical layer before they are even noticed at higher layers. Therefore, it is critical for optical networks to employ fast and effective methods for identifying and locating network failures. Some failures, such as optical cross-connect port blocking and intrusion, can affect a single or a specific subset of wavelengths within a link. Other failures, including fiber cuts and high bit error rates (BERs), may affect all the wavelengths that pass through a fiber duct. In this work, we focus on the detection of the latter type of failures, and present a fault detection technique that can uniquely localize any single-link failure. For ease of explanation, we use the notion of "failure," although the treatment applies as well to assessing other metrics that significantly impact the link performance, such as optical power, optical signal-to-noise ratio (SNR), and BER. In order to rapidly measure the performance of a link (or a collection of links), it is essential to analyze the signal in the optical domain via optical spectrum analyzers (monitors).

Failure detection in all-optical networks (AONs) may be performed by monitoring links individually, requiring a dedicated monitoring lightpath per link. The link quality in this case is assessed by transmitting specific test patterns. To assess the quality at various transmission rates, the test patterns must be transmitted several times at various rates. Instead of dedicating a monitoring wavelength per link, one may use "in-band" approaches, whereby data channels (wavelengths) carry the test patterns in the overhead fields of the frame (similar to L1 monitoring in SONET/SDH networks). This requires electronic dropping of the data channel at the two ends of the link. In such a scenario, the data connection is no longer all-optical, and moreover it is not transparent (i.e., the end-user's framing procedure must be known to the network provider).

Various optical-level mechanisms for failure detection were proposed in the literature. These include optical spectral analysis, optical power detection, pilot tones, and optical time domain reflectometry (OTDR). In [6], a failure detection scheme was proposed, in which monitors are assigned to each optical multiplexing and transmission section. Such a scheme requires an excessive number of inherently expensive monitors, and is not scalable and cost effective for current AONs. In [13], a failure detection mechanism that employs independent wavelengths as supervisory channels was proposed. A cycle cover was used to monitor the links in the network. For each cycle, a node is designated to monitor that cycle's performance. Since the network is decomposed into cycles, faults can be localized at the cycle level, but this type of localization may not identify the failed link within that cycle.

In [12], an adaptive technique for fault diagnosis using "probes" was presented in which probes are established *sequentially*, each time using information about already established probes. While the sequential probing helps achieve adaptiveness, it also increases the fault localization time. In [7], a *non-adaptive* fault diagnosis through a set of probes (lightpaths) was developed where all the probes are employed in advance. The techniques presented in [7] and [12] assume that a probe can originate from and terminate at any location. Such techniques can localize multiple-link failures, but they involve a high setup cost (a monitor is needed at each location) and incur high protocol overhead (for exchanging messages

This work was supported by NSF under grants ANI-0313234, ANI-0325979, and 0435490. Any opinions, findings, conclusions, and recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

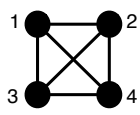
between locations). By reducing the number of monitoring locations, the monitoring overhead and the detection delay can be significantly reduced because fewer messages are exchanged between monitoring locations.

In this paper, we develop a mechanism for locating single-link failures using monitoring cycles (MCs) only or a combination of MCs and monitoring paths (MPs). The MCs pass through one or more monitoring locations, and are selected such that the failure of any given link would result in the failure of a *unique* combination of MCs. We show that when one monitoring location is employed, finding a set of MCs that is sufficient to identify any single-link failure requires the network to be three-edge connected. We provide upper and lower bounds on the number of MCs needed to uniquely identify any single-link failure. The problem of constructing MCs in a three-edge-connected network is formulated as an ILP, and a heuristic approach is presented to solve it. We then study the single-link failure detection problem when multiple monitoring locations are employed. The effectiveness of employing multiple monitoring locations is compared with that of a single monitoring location. We then turn our attention to failure detection in an *arbitrary* topology (i.e., not necessarily three-edge connected). In this case, both MCs and MPs with multiple monitoring locations are needed to localize any single-link failure. We provide necessary and sufficient conditions for the number of monitoring locations needed to identify all such failures. An $\mathcal{O}(|\mathcal{L}|)$ algorithm is presented for calculating the minimum number of required monitoring locations for a network with $|\mathcal{L}|$ links. Simulations are conducted to study the effectiveness of the monitoring technique.

The rest of the paper is organized as follows. In Section II, we consider single-link fault-localization using MCs only and with one monitoring location. We show that three-edge connectivity of the network graph is a necessary and sufficient condition for the existence of a solution. We also provide theoretical bounds on the number of required MCs. In the same section, we present the ILP and heuristic solutions for the MC-based fault localization problem in a three-edge-connected network. In Section III, we study the problem of identifying appropriate MCs when multiple monitoring locations are used, and propose a heuristic approach for solving it. In Section IV we study the problem of identifying single-link failures in an arbitrary network and provide necessary and sufficient conditions on the number of monitoring locations. Section V presents the simulation results and finally the paper is concluded in Section VI.

II. Fault Localization Using Cycles And One Monitoring Location

We first consider the scenario when the network employs one monitoring location. Given the monitoring location, the goal is to construct a set of MCs (or simply cycles) that pass through the monitoring location. The choice of the monitoring location depends on various factors such as its geographic location, security issues, and network topology. A wavelength is reserved for each cycle, and the failure of a cycle may be detected at the monitoring location using a dedicated monitor per cycle or by time sharing the available set of monitors at the monitoring location.



Links	c_1	c_2	c_3	Links	c_1	c_2	c_3
$(1,2)$	×		×	$(4,1)$	×	×	
$(2,3)$		×		$(4,2)$	×	×	×
$(3,4)$			×	$(1,3)$		×	×

Fig. 1. Example network topology where node 1 is the monitoring location, monitoring cycles and its associated syndromes.

In addition to detecting link failures, MCs may also be used to assess various performance metrics, such as optical power, optical signal-to-noise ratio (SNR), and BER. For example, if high BERs is observed on a link, the cycles traversing that link will also exhibit a high BERs. Since the combination of cycles is unique to a given link, the observed phenomena can be uniquely attributed to that link's performance.

Problem MC-1 (Monitoring Cycle Problem With One Monitoring Location): Given a network graph $\mathcal{G}(\mathcal{N}, \mathcal{L})$, where $\mathcal{N}$ is the set of nodes and $\mathcal{L}$ is the set of undirected edges, and given a monitoring location $m \in \mathcal{N}$. Find a set of MCs $\mathcal{C}$ such that every link in the network is present in at least one cycle and every single-link failure results in the failure of a unique subset of cycles in $\mathcal{C}$. All MCs must pass through the given monitoring location m. Such a unique subset is called the *syndrome* of the link. The set $\mathcal{C}$ is called as the fault-detection (FD) set.

We refer to a set of MCs that can uniquely identify all single-link failures as a *feasible solution*. In a feasible solution, every link must be present in at least one cycle, and hence a link failure results in the failure of at least one cycle. In addition, a link failure results in a unique syndrome. Consider the example in Figure 1, where node 1 is the monitoring location. A feasible solution is given by the three cycles $c_1 = (1 \rightarrow 2 \rightarrow 4 \rightarrow 1)$, $c_2 = (1 \rightarrow 3 \rightarrow 2 \rightarrow 4 \rightarrow 1)$, and $c_3 = (1 \rightarrow 2 \rightarrow 4 \rightarrow 3 \rightarrow 1)$. The associated syndromes are also shown in Figure 1. For example, the syndrome for link $(1,2)$ is the set of cycles $\{c_1, c_3\}$.

As every link is required to be on at least one cycle, the network must be at least two-edge connected[1] [2],[3]. However, as we show next, three-edge connectivity is a necessary and sufficient condition for computing a feasible solution to the MC-1 problem.

Theorem 1: Three-edge connectivity is a necessary and sufficient condition for the existence of a solution to the MC-1 problem.

Proof: The necessity part is proved by contradiction. Assume to the contrary that there is a feasible solution to MC-1 in a two-edge but not three-edge-connected network. Since the network is not three-edge connected, there exists two links, say ℓ and ℓ', whose removal disconnects the network. Therefore, any cycle that traverses through one of these two links must also traverse the other. Thus, the failure of any of these two links cannot be uniquely identified through MCs, leading to a contradiction. Therefore, a feasible solution may be obtained only if any two link failures do not disconnect the

[1]A k-edge-connected network is one in which the removal of any $k-1$ edges does not disconnect the network.

network, i.e., three-edge connectivity is a necessary condition for the existence of a solution.

We now prove the sufficiency part by providing a method for constructing a feasible solution in a three-edge-connected network. For any two links $\ell = (x_\ell, y_\ell)$ and $\ell' = (x_{\ell'}, y_{\ell'})$ in the network, we define three types of cycles:

- $\mathcal{T}_1$: Set of cycles containing ℓ but not ℓ'.
- $\mathcal{T}_2$: Set of cycles containing ℓ' but not ℓ.
- $\mathcal{T}_3$: Set of cycles containing both ℓ and ℓ'.

To distinguish a failure between links ℓ and ℓ', it is obvious that a feasible solution must have cycles that belong to at least two of the above three types. For example, if feasible solution contains cycle $c_1 \in \mathcal{T}_1$ and cycle $c_3 \in \mathcal{T}_3$, then the failure of ℓ will result in the failure of c_1 and c_3, whereas the failure of ℓ' will result in failure of c_3 only. Similarly, for other cycle combinations, i.e., $(c_1 \in \mathcal{T}_1, c_2 \in \mathcal{T}_2)$ and $(c_2 \in \mathcal{T}_2, c_3 \in \mathcal{T}_3)$, we can easily show that the failures of ℓ and ℓ' can be distinguished. In a three-edge-connected network, one could easily construct cycles of type $\mathcal{T}_1$ and $\mathcal{T}_2$. A cycle of type $\mathcal{T}_1$ may be constructed using the following two steps: (1) remove link ℓ' from the network (the resultant network is at least two-edge connected); and (2) in the resultant network, compute a cycle traversing the monitoring node m and link ℓ, as shown in Figure 2.

1) Add a virtual node v and two virtual links (v, x_ℓ) and (v, y_ℓ). Remove link ℓ. Note that the resultant graph is still two-edge connected.
2) Obtain two link-disjoint paths $\mathcal{P}_1$ and $\mathcal{P}_2$ from v to m.
3) Form a cycle c by joining $\mathcal{P}_1$ and $\mathcal{P}_2$ after removing the virtual links and adding the link ℓ.

Fig. 2. Procedure to compute a cycle traversing a given node m and link ℓ in a two-edge-connected network.

A cycle of type $\mathcal{T}_2$ may be constructed using the same method above by interchanging ℓ and ℓ'. Finally, to obtain a feasible solution, for each link pair (ℓ, ℓ'), we find cycles of types $\mathcal{T}_1$ and $\mathcal{T}_2$ and take the union of all these cycles to form the FD set. ■

In Theorem 1, we do not show an algorithm for constructing a cycle of type $\mathcal{T}_3$. The FD set is constructed using cycles of type $\mathcal{T}_1$ and $\mathcal{T}_2$ only.

A. Bounds on the Required Number of Cycles

A network with $|\mathcal{L}|$ links requires at least $\lceil \log_2(|\mathcal{L}|+1) \rceil$ fault-detection cycles for any arbitrary choice of the monitoring node. This lower bound is achieved when all the edges are expressed as a binary representation of $\lceil \log_2(|\mathcal{L}|+1) \rceil$ fault-detection cycles. We assume that at any time instance the network may have no failures or at most one link failure. In order to encode $|\mathcal{L}|+1$ combinations in binary representations, we require $\lceil \log_2(|\mathcal{L}|+1) \rceil$ bits, hence the number of cycles. For the example network of 6 links and the set of MCs presented in Figure 1, achieves the lower bound ($\lceil \log_2(6+1) \rceil$ = 3).

The sufficiency proof stated above provides a method for obtaining a feasible solution by constructing cycles of type $\mathcal{T}_1$ and $\mathcal{T}_2$ for all possible link pairs. This results in an upper bound of $|\mathcal{L}|(|\mathcal{L}|-1)$ on the required number of cycles. A tighter bound can be obtained by modifying the construction of the feasible solution, as shown in Figure 3. Step 2 ensures

Step 1: Initialize: $\mathcal{C} \leftarrow \phi$.
Step 2: $\forall \ell, \ell' \in \mathcal{L} \;\; (\ell \neq \ell')$ do
1) If $\mathcal{C}$ does not contain a cycle of types $\mathcal{T}_1$ or $\mathcal{T}_2$, then construct a cycle c of either type.
2) $\mathcal{C} \leftarrow \mathcal{C} \cup \{c\}$.

Note: The above step adds at most $|\mathcal{L}|(|\mathcal{L}|-1)/2$ cycles.
Step 3: $\forall \ell \in \mathcal{L}$ do
1) If $\mathcal{C}$ does not contain any cycle involving link ℓ, then construct a cycle c' traversing m and link ℓ.
2) $\mathcal{C} \leftarrow \mathcal{C} \cup \{c'\}$.

Fig. 3. Construction of a feasible solution with $\binom{|\mathcal{L}|}{2} + 1$ cycles.

that for every two links ℓ and ℓ', at least one cycle of type $\mathcal{T}_1$ or $\mathcal{T}_2$ is present in $\mathcal{C}$. Without loss of generality, assume that there exists a cycle $c \in \mathcal{C}$ such that $c \in \mathcal{T}_1$. The maximum number of cycles added in this step is $\binom{\mathcal{L}}{2}$. There are two cases to consider: (1) There exists another cycle c' that contains ℓ', or (2) none of the cycles in $\mathcal{C}$ contains link ℓ'. In the former case, $c' \in \mathcal{T}_2$ or $c' \in \mathcal{T}_3$. Therefore, $\mathcal{C}$ has cycles belonging to at least two of the three types. In the latter case, we add a cycle c' that traverses the monitoring location m and link ℓ' (Step 3). Note that c' must belong to $\mathcal{T}_2$ or $\mathcal{T}_3$. Clearly, there can be at most one link in the network that belongs to this case[2]. Hence, the upper bound on the number of cycles is $\binom{\mathcal{L}}{2} + 1$.

In Theorem 1, we do not present an algorithm for constructing cycles of type $\mathcal{T}_3$. However, we argue that the exclusion of such cycles from the FD set does not change the upper bound on the size of the FD set.

Now suppose that an algorithm exists for obtaining cycles of type $\mathcal{T}_3$. We now argue that even when such cycles are used to construct a sufficient FD set, in the worst-case, the size of the FD set will still be $\binom{|\mathcal{L}|}{2} + 1$, i.e., the $\mathcal{T}_3$ cycles have no impact on the upper bound. We start with an empty FD set. For each link pair (ℓ, ℓ'), suppose that we obtain a cycle of type $\mathcal{T}_3$ and add it to the FD set. However, such a cycle cannot alone distinguish between the failures of ℓ and ℓ'. Thus, to uniquely identify the failures of the two links, we still need another cycle of type $\mathcal{T}_1$ or $\mathcal{T}_2$. This is true for each link pair (ℓ, ℓ'). Hence, one cycle for each link pair is still required for failure identification.

While the availability of $\mathcal{T}_3$ cycles does not impact the upper bound $\binom{|\mathcal{L}|}{2} + 1$, it provides more flexibility in selecting the FD set, which can lead to reducing the average cycle length in the solution set.

In [7], a stricter bound on the number of monitoring "probes" was derived under the assumption that the probes can originate and terminate at any node in the network, i.e.,

[2]If two links fall in this case, then Step 2 would have ensured that at least one of the links is present in the solution.

the number of monitoring locations is not restricted. However, in our case, monitoring cycles originate from and terminate at the same monitoring location.

B. Complexity

For a three-edge connected network, a solution to MC-1 problem can be obtained by using the procedure described in Section II-A which in the worst case involves $\mathcal{O}(|\mathcal{L}^2|)$ cycles. In [10], it has been shown that the adaptive or sequential diagnosis problem is co-NP-complete. We strongly believe that finding an optimal solution to the MC-1 problem that minimizes the total network resource consumed belongs to the class of NP-Complete problems. However, we do not give a formal proof for the same.

C. Construction of an Optimal FD set

From a hardware standpoint, building a monitoring system involves both operational and setup (one-time) cost. We assume that the operational cost per link is directly proportional to the number of wavelengths used for monitoring purposes over that link. Such dedicated wavelengths represent a loss of revenue for the network provider, as they would have otherwise been used to transport actual traffic. The setup cost is essentially the cost of monitors needed at the monitoring location.

When monitors are dedicated to individual cycles, a wavelength is reserved over all the links in a cycle. The total operational cost is then proportional to the sum of the hop length of cycles employed in the monitoring system. The setup cost in this case is directly proportional to the number of monitoring cycles employed. If a network is heavily loaded, i.e., if the cost of reserving wavelengths over a period of time will dominate over the fixed cost associated with monitors, it is desired to minimize the sum of length of cycles employed for monitoring purpose.

When monitors are time-shared, a single wavelength is reserved for monitoring purposes over each link, and only one monitor is employed at the monitoring location. In such a case, the operational cost is constant, i.e., one wavelength per link, and the setup cost is the cost of a monitor. To identify failures, cycles are sequentially scanned by the monitor. The larger the set of cycles, the higher the delay incurred in identifying a failure. Hence, when a single monitor is time-shared among a set of cycles, the objective should be to minimize the number of cycles in order to minimize the processing delay.

In this work, we assume that a monitor is dedicated to each cycle and that the cost of reserving wavelengths over links dominate over the cost of monitors. We first formulate the problem of constructing a feasible solution that minimizes the sum of the cycle lengths as an ILP. We then develop heuristic solutions to it. To guarantee the existence of a solution, we assume that the network is three-edge connected. This restriction is relaxed later in Section IV.

1) ILP Formulation: For a network graph $\mathcal{G}(\mathcal{N}, \mathcal{L})$ and a given monitoring location $m \in \mathcal{N}$, let $\mathcal{C}_m$ be the set of all cycles that pass through m. The objective of the ILP is to find an FD set such that the average number of cycles per link is minimized. The ILP formulation takes C_m as input. Let α_c be a binary variable that specifies if cycle $c \in \mathcal{C}_m$ is part of the solution or not. Let L_c be the hop-length of cycle c. For a link $\ell \in \mathcal{L}$, let $A_{\ell c}$ be a binary indicator that denotes whether link ℓ is present in cycle c or not; $A_{\ell c} = 1$ if $\ell \in \mathcal{L}$, and 0 otherwise. The ILP formulation is shown in Figure 4. The objective is to minimize the total number of hops consumed by the set of chosen cycles in $\mathcal{G}$. This is equivalent to minimizing the sum of the lengths of the cycles. Constraint C1 ensures that for any two links ℓ and ℓ', there exists at least one cycle among the chosen cycles on which exactly one of the two links is present. Hence, the set of cycles that pass through a link is different from those that pass through all other links by at least one cycle. Since this is true for all link pairs, any link failure can be uniquely identified provided there is at least one cycle passing through the failed link. Constraint C2 ensures that every link is part of at least one cycle.

Minimize $\sum_{c \in \mathcal{C}_m} L_c \alpha_c$
Subject to:
C1: $\sum_{c \in \mathcal{C}_m} \alpha_c \times (A_{\ell c}(1 - A_{\ell' c}) + (1 - A_{\ell c})A_{\ell' c}) > 0,$
$\forall \ell, \ell' \in \mathcal{L},\ \ell \neq \ell'$

C2: $\sum_{c \in C_m} \alpha_c A_{\ell c} > 0,\ \forall \ell \in \mathcal{L}$

Bounds: $\alpha_c = \{0, 1\},\ \forall c \in \mathcal{C}_m$

Fig. 4. ILP formulation for finding the optimal FD set in a three-edge-connected network.

2) Efficient Heuristic for Constructing Monitoring Cycles: The above ILP solution guarantees finding the optimal FD set. However, its worst-case complexity grows exponentially with the number of integer variables in the formulation. Also, the algorithm requires as input all the cycles that pass through the monitoring node. This involves enumerating all cycles, so this step requires an exponentially large number of computations. The exponential complexity of the ILP approach makes it impractical for application in large networks. We now present a heuristic called `MC-1` for finding a good solution. The basic idea behind this algorithm is summarized in Figure 5. Step

1) Initialize $\mathcal{C} = \phi$ and $\mathcal{S} = \phi$
2) While $\mathcal{L} \setminus \mathcal{S} \neq \phi$
   a) Select a link ℓ randomly from $\mathcal{L} \setminus \mathcal{S}$
   b) Compute cycle c passing through m and ℓ using procedure in Figure 2.
   c) $\mathcal{S} = \mathcal{S} \cup \{\ell' : \ell' \in c\}$, $\mathcal{C} = \mathcal{C} \cup \{c\}$
3) While there exists a set of links $\mathcal{L}'$ ($|\mathcal{L}'| \geq 2$), where every link is present in the same set of cycles,
   a) Assign a large weight to all links in $\mathcal{L}'$ and unit weight to all other links.
   b) Choose a random link $\ell^* \in \mathcal{L}'$
   c) Find a minimum-weight cycle c through m and ℓ^* using procedure in Figure 2.
   d) $\mathcal{C} = \mathcal{C} \cup \{c\}$

Fig. 5. Basic idea of the `MC-1` algorithm with node m as monitoring location.

2 in Figure 5 ensures that each link is part of at least one cycle. Such a set of cycles is called the *initial cover*. To find the first cycle of the initial cover, a link ℓ is chosen randomly

from the set $\mathcal{L}$ and a monitoring cycle passing through ℓ is added to the FD set using the procedure presented in Figure 2. Further cycles are sequentially added to the solution set by randomly choosing a link $\ell \in \mathcal{L}$ that is not part of any cycle in the solution set, and then a cycle that passes through ℓ is added to the solution set.

Step 3 ensures that each link is associated with a unique combination of cycles. Let $\mathcal{L}'$ denote the set of links that are present in the same set of cycles in $\mathcal{C}$. A large weight is assigned to all the links in $\mathcal{L}'$. A link ℓ^* is then chosen randomly from $\mathcal{L}'$ and a cycle c passing through ℓ^* is added to $\mathcal{C}$ using the procedure presented in Figure 2. It is guaranteed that cycle c does not contain at least one link from $\mathcal{L}' \setminus \{\ell^*\}$ because the graph $\mathcal{G}$ is three-edge connected.

Let us define $\mathcal{L}'_1 = \{\ell : \ell \in c \cap \mathcal{L}_1\}$ and $\mathcal{L}'_2 = \mathcal{L}_1 \setminus \mathcal{L}'_1$. The selection of cycle c guarantees that $\mathcal{L}'_1 \neq \phi$ and $\mathcal{L}'_2 \neq \phi$. It also guarantees that $|\mathcal{L}'_1| < |\mathcal{L}'|$ and $|\mathcal{L}'_2| < |\mathcal{L}'|$. Hence, by adding cycle c to the solution set, the `MC-1` algorithm splits $\mathcal{L}'$ into two sets of smaller size. The above procedure is repeated until all links have unique syndrome.

We maintain tag values for cycles and cumulative tag values for links to improve the running time of Step 3. The details of the tag values and other implementation details of the `MC-1` algorithm are presented in Appendix I.

Correctness and Complexity: Assume to the contrary that the `MC-1` algorithm never terminates, i.e., the algorithm never comes out of the while loop in Step 3 in Figure 5. If it never terminates, then $\exists\ \ell$ and a set of n links $\{\ell^{(1)}, \ell^{(2)}, ..., \ell^{(n)}\}$ that are present in the same set of cycles. Since the `MC-1` algorithm never terminates, a shortest cycle for link ℓ constructed using the procedure presented in Figure 2 results in a cycle c that also passes through the set of links $\{\ell^{(1)}, \ell^{(2)}, ..., \ell^{(n)}\}$. We show that this will never happen if the network is three-edge connected and that a shorter cycle c' can be constructed. Recall that before adding a cycle, we set a large weight (say V_∞) to the n links $\{\ell^{(1)}, \ell^{(2)}, ..., \ell^{(n)}\}$. The weight of cycle c (which is the sum of the weights of links in c) is then greater than nV_∞. Now, remove from the network graph one of the n links say $\ell^{(1)}$. The graph is still two-edge connected. Finding a shortest cycle using the procedure in Figure 2 will result in a cycle c' of weight $W(c') < (n-1)V_\infty + |\mathcal{L}| < W(c)$, because only $n-1$ links were assigned a large weight. The above is true provided that $V_\infty \geq |\mathcal{L}|$.

Note that the addition of a cycle using the procedure in Figure 2 results in distinguishing between the failures of at least two links. Hence, Step 3 requires a maximum of $\mathcal{O}(|\mathcal{L}|^2)$ iterations. A cycle traversing a node m and link ℓ may be obtained in $\mathcal{O}(|\mathcal{N}| \log |\mathcal{N}| + |\mathcal{L}|)$ time by running two instances of Dijkstra's algorithm [1]. The construction of an initial cover requires computing $O(|\mathcal{L}|)$ cycles (in the worst case, one cycle per link) and Step 3 requires $O(|\mathcal{L}|^2)$ cycles (one cycle for every link pair). Hence, the worst-case complexity of our algorithm is $O(|\mathcal{L}|^2(|\mathcal{N}| \log |\mathcal{N}| + |\mathcal{L}|))$.

III. Fault Detection Using Multiple Locations

We now consider a scenario where multiple locations are used to monitor single-link failures in a three-edge-connected network. This may be needed, for example, to provide more flexibility to the network operator, who may use multiple monitoring points for security and scalability reasons. Using more monitoring locations is also expected to result in a decrease in the average cycle length, and hence in faster localization of the failed link.

Problem MC-M (Monitoring Cycle Problem with Multiple Monitoring Locations): Given a graph $\mathcal{G}(\mathcal{N}, \mathcal{L})$ and a set of monitoring locations $\mathcal{M}$ (with $|\mathcal{M}| = M$), the goal is to find a set of MCs $\mathcal{C}$ such that every single-link failure results in the failure of a unique combination of MCs. Each MC must pass through at least one of the monitoring locations.

Depending on whether the monitoring locations share information about cycle failures or not, we define two modes of operation:

1. *With information exchange*: Monitoring locations exchange information about observed cycle failures to collaboratively localize a link failure. The information exchange process will add some delay to the failure localization time. However, as shown later, less network resources are needed for failure detection.
2. *Without information exchange*: The monitoring locations work independently and do not share any information. Fault localization is faster but may require additional network resources. In this case, every link failure results in the failure of a combination of cycles traversing a specific monitoring location.

Our `MC-M` algorithm for MC construction with multiple monitoring locations incorporates the two modes. For a given monitoring location, we define its *cloud* as the set of links that this location is responsible for monitoring. The main intuition behind the `MC-M` algorithm is to choose cycles with relatively short lengths. The basic idea of the `MC-M` algorithm can be described in the following two steps:

1. We associate each link with the cloud of the monitoring location that is closest to that link. By being closer to the monitoring location, it is more likely that the monitoring cycles passing through the link will be shorter. We refer to this step as *cloud formation*.
2. We find a set of cycles that pass through each monitoring location such that each link failure results in the failure of a unique combination of cycles: (1) in the network, if information exchange is employed; or (2) at a monitoring location, if information exchange is not employed. We refer to this step as *cycle formation*. When constructing cycles for the cloud of a given monitoring location, the `MC-M` algorithm assigns very high weights to links that are not part of that cloud. This discourages the use of such links in forming the MCs and facilitates the construction of short cycles.

Finally, the solution is further optimized by using cloud transfer. It is likely that when monitoring cycles for a cloud are added to the FD set, a set of links that are not part of this cloud will also result in unique syndromes. Such links can be removed from their respective clouds via a cloud transfer in order to conserve resources. Implementation details of the `MC-M` algorithm are presented in Appendix II.

A. Monitoring with Paths and Cycles

So far, we have used MCs to monitor single-link failures. This constrains the solution space because every test signal that originates from a monitoring location must eventually return to it in order to close the loop. The use of paths along with cycles provides greater flexibility. A *monitoring path* (MP) originates from one monitoring location and terminates at another. The failure of a link along the path will be detected by the terminating monitoring location associated with that path. If the starting and ending monitoring locations are the same, the monitoring path reduces to a MC. For the example network and the feasible solution presented in Figure 1, if we replace cycle c_3 with the monitoring path $p_1 = (2 \rightarrow 4 \rightarrow 3 \rightarrow 1)$ and if we use node 2 as a second monitoring location, we can still detect all single-link failures. The associated syndromes generated with various link failures are given in Table I.

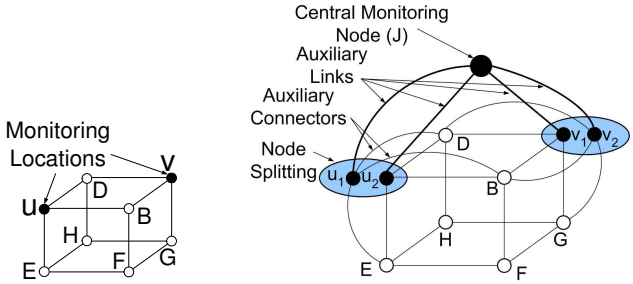
Link	c_1	c_2	p_1	Link	c_1	c_2	p_1	Link	c_1	c_2	p_1
(1,2)	×			(4,1)	×	×		(3,4)			×
(2,3)		×		(4,2)	×	×	×	(1,3)		×	×

TABLE I
SYNDROMES ASSOCIATED WITH CYCLES c_1, c_2, AND PATH p_1.

Before proceeding further, we formally define the monitoring problem with paths and cycles.

Problem MPC (Monitoring with Paths and Cycles): Given a graph $\mathcal{G}(\mathcal{N}, \mathcal{L})$ and a set of monitoring locations $\mathcal{M}$, the goal is to find a set of monitoring paths $\mathcal{P}$ and a set of monitoring cycles $\mathcal{C}$ such that every single-link failure results in the failure of a unique combination of paths and cycles in $\mathcal{P} \cup \mathcal{C}$.

We now describe a network transformation that converts the MPC problem into a modified version of the MC-1 problem. First, we split each monitoring node $u \in \mathcal{M}$ into two *virtual monitoring nodes* u_1 and u_2, as shown in Figure 6. We introduce a *central monitoring node* J to which we connect the virtual monitoring nodes using auxiliary links. Finally, for each edge $(u, v) \in \mathcal{L}$ s.t. $u \in \mathcal{M}$, we introduce auxiliary connector edges (u_1, v) and (u_2, v). We define the mapping $T(.)$ such that $T((u_1, v)) = \{(u_2, v)\}$ and $T((u_2, v)) = \{(u_1, v)\}$. Note that if both v and w are in $\mathcal{M}$ with $(v, w) \in \mathcal{L}$, four auxiliary connector edges, (v_1, w_1), (v_2, w_2), (v_1, w_2), and (v_2, w_1), need to be added. In this case, $T((v_i, w_j)) \stackrel{def}{=} \{(\bar{v}_i, w_j), (v_i, \bar{w}_j), (\bar{v}_i, \bar{w}_j)\}$, where $i,\ j = \{1,\ 2\}$, and $\bar{v}_i$ and $\bar{w}_i$ are, respectively, the toggles of v_i and w_i in the set {1,2}. For example, $T((v_1, w_1)) = \{(v_2, w_2), (v_1, w_2), (v_2, w_1)\}$. Node J is now the single monitoring location on the transformed graph. We now show that a solution to a slightly modified version of the MC-1 problem with monitoring location at J in the transformed network will result in a solution to the MPC problem. Consider the cycle $c = (J \rightarrow u_1 \rightarrow E \rightarrow F \rightarrow G \rightarrow v_2 \rightarrow J)$ in Figure 6. If we eliminate the auxiliary edges, c becomes the path $(u \rightarrow E \rightarrow F \rightarrow G \rightarrow v)$ in the original graph $\mathcal{G}$. Similarly, cycle $c' = (J \rightarrow u_1 \rightarrow E \rightarrow F \rightarrow B \rightarrow u_2 \rightarrow J)$ in the transformed network becomes cycle $(u \rightarrow E \rightarrow F \rightarrow B \rightarrow u)$ in $\mathcal{G}$. Notice that, an arbitrary cycle in the transformed network may not be a valid path or a cycle in $\mathcal{G}$. For example, the cycle $c'' = (J \rightarrow u_1 \rightarrow E \rightarrow u_2 \rightarrow J)$ in the transformed network is a self-loop $(u \rightarrow E \rightarrow u)$ in $\mathcal{G}$. Self-loops can be eliminated by removing some of the auxiliary connector links during cycle construction.



(a) Network before graph transformation.

(b) Network after transformation.

Fig. 6. Example of Network transformation presented in Section III-A for monitoring a network using paths and cycles.

Cycle Construction in the Transformed Network: For a two-edge-connected graph, we can construct cycles containing link $\ell = (x_\ell, y_\ell)$ by adding a virtual node r, removing ℓ, adding two virtual links $(r,\ x_\ell)$ and $(r,\ y_\ell)$, and finding two link-disjoint paths from the monitoring location J to node r. Such paths can be obtained by assigning a unit capacity to each link in the network and then sequentially augmenting two units of flow from J to r along the two "augmenting paths" in the *residual graph*[3] [1] of the network. To avoid generating a self-loop in the original graph, after augmenting a unit flow along the first augmenting path $\mathcal{P}_{aug1}$, we remove the set of links $T(\ell')$ that corresponds to the auxiliary connector edge $\ell' \in \mathcal{P}_{aug1}$. A self-loop can only be formed if $T(\ell')$ of an auxiliary connector edge $\ell' \in \mathcal{P}_{aug1}$ is present in the residual graph. Notice that the removal of auxiliary connector edges from the residual graph will not disconnect the graph because the original graph is three-edge connected. The MC-

1) Add a virtual node r and two virtual links $(r,\ x_\ell)$ and $(r,\ y_\ell)$. Remove link ℓ. Note that the resultant graph is still two-edge connected.
2) Augment a unit flow from J to r along the first augmenting path $\mathcal{P}_{aug1}$.
3) For every auxiliary connector edge $\ell' \in \mathcal{P}_{aug1}$, remove $T(\ell')$ from the residual graph.
4) Obtain the second augmenting path $\mathcal{P}_{aug2}$.
5) Construct two link-disjoint paths $\mathcal{P}_1$ and $\mathcal{P}_2$ by aggregating the flows of $\mathcal{P}_{aug1}$ and $\mathcal{P}_{aug2}$.
6) Form a cycle c by joining $\mathcal{P}_1$ and $\mathcal{P}_2$ after removing the virtual links and adding the link ℓ.

Fig. 7. Procedure to determine a cycle that traverses the monitoring location J and link ℓ in the transformed network presented in Figure 6.

[3]For a flow network $\mathcal{G}(\mathcal{N}, \mathcal{L})$ in which each link (u, v) is associated with capacity $c(u, v)$, and for a source s, sink t, and a flow f that is defined by a flow demand and a set of paths that support this demand, the residual network of $\mathcal{G}$ induced by f is $\mathcal{G}_f(\mathcal{N}, \mathcal{L}_f)$, where $\mathcal{L}_f = \{(u, v) \in \mathcal{L} : c_f(u, v) > 0\}$. $c_f(u, v)$ is the amount of additional flow that we can push from u to v before exceeding $c(u, v)$, and is given by $c_f(u, v) = c(u, v) - f(u, v)$ [3]. For flow f, an augmenting path from s to t is any path on f with positive flow.

1 procedure presented in Section II-C.2 can now be applied to the transformed network with monitoring location at J. However, instead of calling the `FindCycle` procedure to determine a MC, the `MC-1` will now use the procedure in Figure 7. The $\mathcal{L}_m$ input to `MC-1` in this case consists of all the links in the transformed graph except for the auxiliary links. The tag value (t_ℓ) for an auxiliary connector edge ℓ is set to the sum of tag values of cycles passing through ℓ and $T(\ell)$.

IV. Monitoring in Arbitrary Network Topologies

We have shown that three-edge connectivity is a necessary and sufficient condition to uniquely identify single-link failures using one monitoring location. However, many optical networks are less than three-edge connected, and so they may require additional monitoring locations. We now show that with multiple monitoring locations and by employing MPs along with MCs, all the links of an arbitrarily connected network can be monitored. For any two links ℓ and ℓ', we classify paths into three categories:

- $\mathcal{T}_1$: Set of monitoring paths containing ℓ but not ℓ'.
- $\mathcal{T}_2$: Set of monitoring paths containing ℓ' but not ℓ.
- $\mathcal{T}_3$: Set of monitoring paths containing both ℓ and ℓ'.

To distinguish the failures of links ℓ and ℓ', the FD set should contain paths of at least two of the above three types. We first derive the necessary conditions for monitoring a connected network using cycles and paths.

Definitions:

1) The *k-super-graph* (or simply *k-graph*) of a network graph $\mathcal{G}(\mathcal{N}, \mathcal{L})$ is a graph that represents the interconnectivity of the k-edge-connected components of $\mathcal{G}$ [8]. In other words, each node i of the k-graph is a k-edge-connected component of $\mathcal{G}$.
2) A *component link* $\ell \in \mathcal{L}$ is a link that connects two nodes in the same k-edge-connected component of $\mathcal{G}$.
3) A *connector link* $\ell \in \mathcal{L}$ is a link that connects nodes belonging to two different k-edge-connected components of $\mathcal{G}$. Such a link is also a link on the k-graph.

For a given network $\mathcal{G}$ and a given integer k, the k-graph of $\mathcal{G}$ is unique [8]. The k-graph can be at most $(k-1)$-edge-connected [8], [5], [9]. In this paper, we employ 2- and 3-graphs to determine the minimum number of monitors required to uniquely localize all single-link failures. Note that the 2-graph of a network is a tree.

Lemma 1: To uniquely identify all single-link failures in an arbitrary connected graph $\mathcal{G}(\mathcal{N}, \mathcal{L})$, it is necessary that all the components of the graph that result from the removal of any one link ℓ or two links ℓ and ℓ' must each contain a monitoring location.

Proof: Consider the removal of a link ℓ from $\mathcal{G}$. The removal of ℓ will reduce the graph in to one or two connected components. If it results in one connected component, then this component must contain at least one monitoring location, because otherwise the network will be without any monitoring location. If the removal of ℓ results in two components and only one component has a monitoring location, then no monitoring cycle or path can traverse ℓ.

Consider the removal of two links ℓ and ℓ' from $\mathcal{G}$. The removal of these links will reduce the graph into one, two, or three connected components. If it results in one connected component, then this component must contain at least one monitoring location, because otherwise the network will be without any monitoring location. If the removal of ℓ and ℓ' results in two components (see Figure 8(a)) and only one of the components has a monitoring location, then any monitoring path or cycle that traverses ℓ must traverse ℓ'. Thus, links ℓ and ℓ' are not uniquely identifiable. Therefore, each component must contain at least one monitoring location.

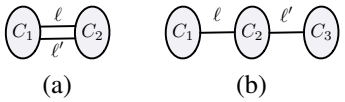


Fig. 8. Decomposition of the network into connected components when two links ℓ and ℓ' are removed. The removal results in: (a) two components and (b) three components.

Suppose that the removal of ℓ and ℓ' results in three components, as shown in Figure 8(b). The removal of link ℓ results in two components and the removal of link ℓ' further disconnects one of the two components. As there cannot be any cycles traversing links ℓ and ℓ', the failure of these two links can be identified using monitoring paths only, thus requiring at least two monitoring locations. If component $C_1(C_3)$ does not contain a monitoring location, then an MP traversing $\ell(\ell')$ cannot be formed. Therefore, C_1 and C_3 must each have at least one monitoring location. Further, if C_2 does not have a monitoring location, then every path from a monitoring location in C_1 to a monitoring location in C_3 will traverse links ℓ and ℓ'. Hence, the failures of ℓ and ℓ' cannot be distinguished. Therefore, C_2 must also have a monitoring location. ■

Corollary 1: To uniquely identify all single-link failures in line and ring networks, all the nodes must be monitoring locations.

Corollary 2: To uniquely identify all single-link failures in a tree network, all the nodes of degree one or two must be monitoring locations.

Corollary 3: In a two-edge-connected network, all three-edge-connected components with two or no connector edges must have a monitoring location.

The proofs for the above three corollaries are obvious from Lemma 1.

Minimum Number of Monitoring Locations and Their Placement: Using the lemma and corollaries described above, the minimum number of required monitoring locations in an arbitrary network can be obtained using the following procedure:

In Steps 3 and 4 of Figure 9, the degree of a k-edge-connected component is that of the corresponding node on the k-graph, $k = 2, 3$. An example of a 2-graph and its 3-graph is shown in Figure 10. Notice that the 2-graph of the example network is a tree. Each three-edge-connected component of degree two or less in the example network has at least one monitoring location inside it. Similarly, each two-edge-connected component has a monitoring location inside it. For example, component A is a two-edge-connected component

1) Obtain the 2-graph of the original network graph $\mathcal{G}(\mathcal{N}, \mathcal{L})$.
2) Decompose every node in the 2-graph (such a node is a 2-edge-connected component of the original graph) further into three-edge-connected components, hence forming the 3-graph of $\mathcal{G}$.
3) Assign a monitoring location to every three-edge-connected component of degree two or less.
4) Assign a monitoring location to every two-edge-connected component of degree two or less, if such a component does not contain a monitoring location from step 3.

Fig. 9. Procedure for finding the minimum number of monitoring locations and their placement.

of the graph, and it is of degree two. Hence, it needs at least one monitoring location inside it. However, all three-edge-connected components within A itself are of degree three and according to step 3 of Figure 9, they do not need a monitoring location. Hence, any arbitrary node is picked in step 4 as a monitoring location inside A. Similarly, component B is of degree two but a monitoring location is not assigned to it in step 4 because it already gets a monitoring location in one of its three-edge-connected component based on step 3.

Complexity: Unique decomposition of a graph into its 2- and

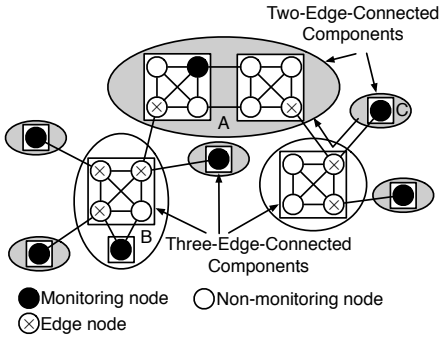


Fig. 10. An example showing the 2-graph and 3-graph of an arbitrary network.

3-graph can be done using $\mathcal{O}(|\mathcal{L}|)$ computations [8]. Identifying the degree of a component by identifying its outgoing edges can be done using $\mathcal{O}(|\mathcal{L}|)$ computations. Hence, the total complexity of the algorithm is $\mathcal{O}(|\mathcal{L}|)$.

Correctness: Given the 2-graph of a graph $\mathcal{G}$ and using Corollary 2, any feasible solution must have at least one monitoring location in each two-edge-connected component of degree two or less. This is ensured by step 4 in Figure 9. Moreover, by using Corollary 3, a feasible solution that has the minimum possible number of monitoring locations must have a monitoring location inside each three-edge-connected component of degree two, which is ensured by step 3. Finally, step 4 does not assign a monitoring location inside a two-edge-connected component if there is at least one monitoring location already assigned inside it from step 3. Hence, the procedure in Figure 9 provides the minimum number of monitoring locations.

Let N_1 denote the number of two-edge-connected components of degree two or less. Let N_2 denote the number of three-edge-connected components of degree two or less. Let N_3 denote the number of two-edge-connected components of degree two or less that have three-edge-connected components of degree two or less inside them. The minimum number of monitoring locations required in the network is then given by $N_1+N_2-N_3$. Note that the above procedure will place exactly one monitor if the given network is three-edge connected.

We define an *edge node* in a two-edge-connected component of a network as a node that is either a monitoring location or is part of a connector edge in the 2-graph. Observe that every edge node is either a monitoring location or is connected to at least one monitor outside its two-edge-connected component. By connected, we mean that there is a path that starts at the edge node and that terminates at a monitoring location outside the two-edge-connected component. This path does not use any edges inside the two-edge-connected component. Therefore, an edge node may be considered as a *virtual* monitor. For example, in Figure 10, there are two edge nodes inside component A. The edge node on the right side of component A is connected to the monitoring location in component C.

Corollary 4: If a two-edge-connected component of a connected network has exactly one edge node, then this component must be three-edge connected. In addition, the edge node must be a monitoring location.

We now prove that the placement of monitors as described in Figure 9 is sufficient to uniquely identify all single-link failures. As a first step, we show that there exists a monitoring path (or cycle) that traverses every link.

Theorem 2: The placement procedure in Figure 9 will result in monitoring locations that are sufficient to construct MPs/MCs that can cover all the links.

Proof: Consider a link $\ell \in \mathcal{L}$. The removal of this link either disconnects the network or not. If it disconnects the network, then the two components connected by ℓ must have a monitoring location (according to the necessary condition), and hence a path from a monitoring location in one component to that in the other must traverse link ℓ.

If the removal of link ℓ does not disconnect the network, then ℓ belongs to a two-edge-connected component of the graph $\mathcal{G}$. This component has one or more edge nodes. If it has one edge node, then by Corollary 4, the component is three-edge connected. Therefore, we can compute a MC traversing link ℓ by using the procedure presented in Figure 2. If the two-edge-connected component has more than one edge node, then two link-disjoint paths can be computed from the virtual node to two distinct edge nodes. As every edge node is by itself a monitoring location or is connected to a monitoring location outside of the two-edge-connected component, we can obtain an MP by extending the path between the edge nodes, if needed. ■

Theorem 3: The monitoring locations placed according to the procedure in Figure 9 are sufficient to uniquely identify all single-link failures.

Proof: We prove the sufficiency condition by constructing a set of monitoring paths and cycles based on the monitoring

locations obtained using Figure 9. Theorem 2 shows that for any link ℓ there exists an MP that traverses this link. We now show that ℓ is uniquely identifiable. After identifying the set of MPs that cover all the links, if two links ℓ and ℓ' have a common syndrome, they are present in the same set of paths, i.e., a set of paths of type $\mathcal{T}_3$. For these two links, we now show that there exists a path of type $\mathcal{T}_1$ or $\mathcal{T}_2$.

We divide the links in the network into three sets. The first set ($\mathcal{L}_1$) contains all links $\ell \in \mathcal{L}$ whose removal disconnects the graph. The second set ($\mathcal{L}_2$) contains all links $\ell \in \mathcal{L}$ whose removal does not disconnect the graph, but $\exists \ell' \in \mathcal{L}$ such that the removal of both ℓ and ℓ' disconnects the graph. Finally, the third set ($\mathcal{L}_3$) contains all other links. Each link in $\mathcal{L}_1$ connects two nodes in the 2-graph of $\mathcal{G}$. Similarly, each link in $\mathcal{L}_2$ connects two nodes in the 3-graph. Links in $\mathcal{L}_3$ are contained within the nodes of the 3-graph.

Case 1: $\ell' \in \mathcal{L}_1$. We consider two sub-cases: (a) $\ell \in \mathcal{L}_1$, and (b) $\ell \notin \mathcal{L}_1$.

Case 1(a): The removal of links ℓ and ℓ' results in three components. Therefore, ℓ' connects two of these components. By the necessary condition, each of these two components must have a monitoring location. A path from one monitoring location in one component to a monitoring location in the other must traverse link ℓ.

Case 1(b): ℓ is part of a two-edge-connected component whose degree after the removal of link ℓ' is, say d. If $d = 0$ or 1, there exists a monitoring location in the component itself. As the component is two-edge connected[4], there exists two link-disjoint paths from the monitoring location to the two nodes (x_ℓ and y_ℓ) connected by link ℓ. Thus, there exists a MC that traverses link ℓ and not link ℓ'. If $d > 1$, then there exists a monitor along every edge node of the component. We can find a path between two edge nodes that traverses link ℓ by replacing link ℓ with a virtual node v and with two virtual links, and computing link-disjoint paths from v to the two edge nodes. Therefore, there exists a path of type $\mathcal{T}_1$ if $\ell' \in \mathcal{L}_1$.

Case 2: $\ell' \in \mathcal{L}_2$. We consider three sub-cases: (a) $\ell \in \mathcal{L}_1$, (b) $\ell \in \mathcal{L}_2$, and (c) $\ell \in \mathcal{L}_3$.

Case 2(a): We can swap links ℓ and ℓ' and use Case 1(b). Hence, the two links are uniquely identifiable.

Case 2(b): The failure of the two links does not disconnect the graph. If the two links are present in different two-edge-connected components, then the removal of ℓ' does not affect the two-edge connectivity of the component in which ℓ is present. Therefore, similar to the argument in Case 1(b), we can find a monitoring path that traverses ℓ and not ℓ'.

If both ℓ and ℓ' belong to $\mathcal{L}_2$ and their removal does not disconnect the graph, then the network can be decomposed into three components C_1, C_2, and C_3, as shown in Figure 11. There exists two links between C_1 and C_2 of which one of them must be ℓ. Similarly, there exists two links between C_2 and C_3 of which one of them is ℓ'. According to the necessary condition, components C_1 and C_3 must have a monitoring location. Therefore, a path from a monitoring location in C_1 to a monitoring location in C_3 that traverses link ℓ may be obtained by removing links ℓ' and ℓ''.

[4]Note that the two-edge connectivity of the component is not disturbed by the removal of ℓ' because $\ell' \in \mathcal{L}_1$

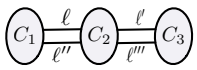


Fig. 11. Decomposition of a network into three components when ℓ and ℓ' belong to $\mathcal{L}_2$ and their failure does not disconnect the network.

Case 2(c): We swap the links and apply Case 3(b). Therefore, a monitoring path of type $\mathcal{T}_2$ exists.

Case 3: $\ell' \in \mathcal{L}_3$. We consider two sub-cases: (a) $\ell \in \mathcal{L}_1$, and (b) $\ell \notin \mathcal{L}_1$.

Case 3(a): We swap the links and apply Case 1(b). Thus, a monitoring path of type $\mathcal{T}_2$ exists.

Case 3(b): The removal of ℓ' does not affect the two-edge connectivity of the component in which link ℓ is present. Therefore, there exists a path between two distinct edge nodes in that component that traverses link ℓ. If the component has only one edge node, then it is three-edge connected. Hence, there exists a MC that traverses link ℓ after the removal of ℓ'. ■

We now present an example of monitoring a network using paths and cycles. The network in Figure 12 has three three-edge-connected components: $C_1 = \{1, 2, 3, 4\}$, $C_2 = \{5, 6, 7, 8\}$, and $C_3 = \{9, 10, 11, 12\}$. Using the procedure in Figure 9, we present a solution to the monitoring problem with monitoring locations at nodes 7 and 11. It can easily be verified that $N_1 = 2, N_2 = 1$, and $N_3 = 1$. The links in component C_1 are monitored by the two monitoring locations provisioned outside C_1, i.e., inside components C_2 (node 7) and C_3 (node 11). The set of five paths and three cycles that result in unique syndromes for all links are p_1= 11-9-4-1-3-2-5-8-6-7, p_2= 11-12-9-10-3-4-2-5-6-7, p_3 = 11-10-12-9-4-1-2-5-7, p_4 = 11-10-3-2-5-8-7, p_5 = 11-12-10-3-4-1-2-5-7, c_1 = 7-8-6-5-7, c_2 = 11-9-10-12-11, c_3 = 7-8-5-6-7. The set of single-link failures and their corresponding syndromes are presented in Table II.

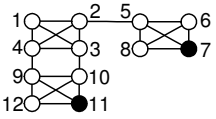


Fig. 12. Example of a network with monitoring locations at nodes 7 and 11.

V. Simulation Results

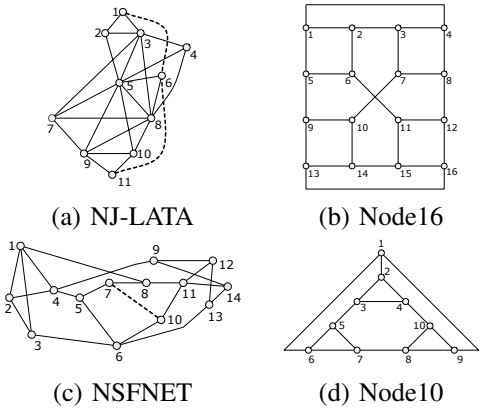
A. Results for the MC-1 Problem

We compare the performance of the MC-1 algorithm with the optimal results obtained using the ILP solution. We first consider the four topologies in Figure 13. These small topologies are selected because they enable us to obtain the ILP solution using the CPLEX solver [4] in reasonable time. For the NJ-LATA and NSFNET topologies, we add a few additional links to make these topologies three-edge connected (a necessary condition for the existence of a solution using one monitoring location). For a given network topology, we vary the location of the monitoring node and report the average number of monitoring-related wavelengths per link, given by $\sum_{c \in \mathcal{C}} L_c / |\mathcal{L}|$. This measure reflects the amount of resources per link needed for link-failure detection. We also report the

ℓ	p_1	p_2	p_3	c_1	c_2	p_4	c_3	p_5	t_ℓ
tag	1	2	4	8	16	32	64	128	
$(1,3)$	×								1
$(2,4)$		×							2
$(1,2)$			×					×	132
$(2,3)$	×					×			33
$(3,4)$		×						×	130
$(4,1)$	×		×					×	133
$(5,6)$		×		×			×		74
$(6,7)$	×	×					×		67
$(7,8)$				×		×	×		104
$(8,5)$	×					×	×		97
$(5,7)$			×	×				×	140
$(6,8)$	×			×					9
$(9,10)$		×			×				18
$(10,11)$			×			×			36
$(11,12)$		×			×			×	146
$(12,9)$		×	×						6
$(9,11)$	×				×				17
$(10,12)$			×		×			×	148
$(4,9)$	×		×						5
$(3,10)$		×				×		×	162
$(2,5)$	×	×	×			×		×	167

TABLE II

SINGLE-LINK FAILURES AND ASSOCIATED SYNDROMES FOR THE NETWORK IN FIGURE 12.



(a) NJ-LATA (b) Node16

(c) NSFNET (d) Node10

Fig. 13. Network topologies used in the simulations.

number of cycles associated with the monitoring location, which reflects the setup cost of the monitoring system.

For symmetric topologies (Node10 and Node16), we only consider as monitoring locations nodes that are symmetrically different from other nodes. For example, in the Node16 topology, nodes 1, 4, 13 and 16 are isomorphic, i.e., one can rotate and flip the topology to replace one node with the other. Hence, placing the monitoring node at any of these locations will result in the same performance.

Figure 14 depicts the observed performance. About two wavelengths per link are found sufficient to localize single-link failures. This confirms the effectiveness of our FD approach. The ILP solution is obtained under two objective functions: minimizing the total number of cycles (ILP/MNC) and minimizing the sum of cycle lengths (ILP/MNCL). The second objective is equivalent to minimizing the number of cycles per link. It is noted that for `MC-1`, the number of cycles in the FD set is approximately linear in the number of links. Like ILP/MNCL, `MC-1` aims at minimizing the average number of cycles per link. It does that using 20-50% extra resources compared with ILP/MNCL.

We also report the number of cycles in the FD set. If the network is designed with one monitor per MC, then the number of cycles reflects the initial setup cost of the monitoring system. On the other hand, if one monitor is time-shared by all cycles, then the number of cycles is an indicator of the processing delay[5]. When the network is lightly loaded, the cost of monitors overshadows the cost of reserving wavelengths, and so the network designer should aim at minimizing the number of MCs in the FD set.

Next, we consider randomly generated Waxman topologies[6] [11] of 10, 30, and 50 nodes. To ensure that topology is three-edge connected, if two links ℓ and ℓ' are found to disconnect the graph into two components, we add an additional link between two randomly chosen nodes, one from each component. For a given topology, we vary the location of the monitoring location and use the `MC-1` algorithm to find a feasible set of cycles. Figure 15 presents the histogram of the number of wavelengths (cycles) that traverse a link (results are averaged over all nodes serving as monitoring locations). We make the following observations from this figure:

1) Most links require no more than 3 wavelengths.
2) In large networks (50 nodes), a significant number of wavelengths (> 20) are required for links that are near the monitoring location. This is undesirable and leads to unfair distribution of wavelengths. To reduce the maximum number of wavelengths consumed over a link, one should use multiple monitoring locations.

Monitor Placement: The choice of the monitoring location within a three-edge-connected component of an arbitrary network topology is an important design factor. A poor choice can significantly increase the average number of monitoring-related wavelengths per link. We now present a simple placement heuristic to efficiently locate the monitoring location inside a three-edge-connected component $\mathcal{N}_c$.

For any two nodes n_1 and n_2 in $\mathcal{N}_c$, we identify two link-disjoint paths between them such that the sum of the hop lengths of the two paths (denoted by $W(n_1, n_2)$) is minimized. We then choose a node $n^* \in \mathcal{N}_c$ such that $n^* = \min_{n_i \in \mathcal{N}_c} \sum_{n_j \in \mathcal{N}_c} W(n_i, n_j)$. In case of a tie, we choose the node with the minimum average shortest path distance (w.r.t. to hop length). Further ties are resolved randomly.

Table III compares the placement heuristic with the "optimal" results[7], in terms of the average number of cycles per link. The results indicate that the monitoring location returned by the placement heuristic incurs less than 20% extra resources than the "optimal" monitoring location.

B. Results for the MC-M Problem

We now study the performance of the `MC-M` algorithm. We use a randomly generated Waxman's topology [11] of 50 nodes. To ensure that topology is three-edge connected, if

[5]To identify failures, the monitor sequentially scans the cycles in the FD set. Thus, a large FD set implies a higher processing delay.

[6]In a Waxman graph, an edge is added between two nodes with a probability that decreases exponentially with the distance between them.

[7]The optimal results are obtained by running `MC-1` for all possible monitoring locations and choosing the node with the minimum average number of cycles per link.

Network	NJ-LATA						NSFNET						Node 16				Node 10					
No. of monitoring nodes	1	3	5	7	9	11	1	3	5	7	9	11	1	2	5	6	1	2	3	5	6	7
No. of possible FD cycles	717	1003	1096	760	928	749	296	246	240	265	278	322	223	223	223	223	36	33	35	35	33	35
No. of cycles in FD set (ILP with MNC)	6	6	6	6	6	6	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
No. of cycles in FD set (ILP with MSCL)	8	7	8	8	7	7	6	5	6	6	6	6	5	5	5	5	5	5	6	6	5	6
No. of cycles in FD set (MC-1)	13	13	16	13	12	10	11	11	12	10	10	10	9	10	8	11	7	7	7	8	8	7
Avg. no. of cycles/link (ILP with MSCL)	2.12	1.8	1.72	1.96	1.88	2.04	2.04	2.13	2.17	2.13	2.04	2.04	2.3	2.1	2.1	2.1	1.86	1.86	1.93	1.93	1.86	1.93
Avg. no. of cycles/link (MC-1)	3.16	2.4	2.4	2.56	2.7	2.9	3.04	2.65	2.73	2.91	3	3.04	3.3	3.1	3.2	3.3	2.46	2.4	2.66	2.7	2.7	2.4

Fig. 14. Simulation results for the MC-1 problem (NJ-LATA, NSFNET, Node16, and Node10 topologies).

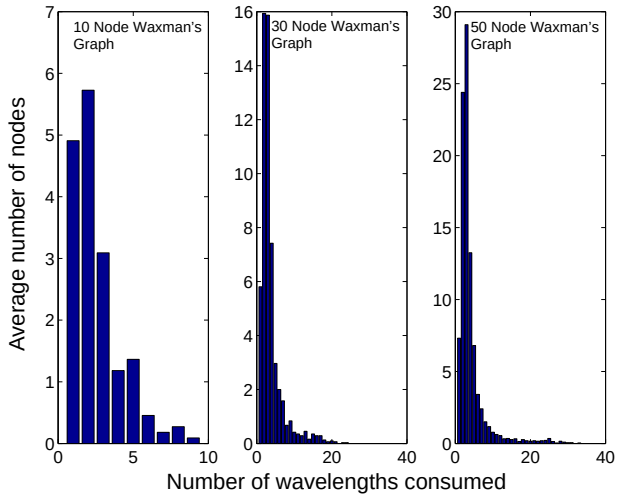


Fig. 15. Distribution of monitoring-related wavelengths over various links in Waxman's topologies using the MC-1 algorithm.

Network	Placement Heuristic		Simulated Optimal	
	Location	Cycles per Link	Location	Cycles per Link
NJLATA	5	2.4	8	2.24
NSFNET	11	3.04	6	2.57
Node16	16	2.91	7	2.79
Node10	9	2.53	2	2.4

TABLE III

COMPARISON BETWEEN THE PLACEMENT HEURISTIC AND OPTIMAL PLACEMENT USING THE MC-1 ALGORITHM.

two links ℓ and ℓ' are found to disconnect the graph into two components, we add an additional link between two randomly chosen nodes, one from each component. We call the resulting topology as Random-3. Each link is assumed to have a propagation delay that is sampled from a uniform distribution in the range $[0, 50]$ msec.

The $\mathcal{M}$ monitoring locations are randomly selected from the 50 nodes. We vary $\mathcal{M}$ to study the performance of the MC-M algorithm. For each simulation run, we consider both cases described in Section III (i.e., with and without information exchange). The performance is evaluated in terms of the average number of cycles per link (i.e., average number of wavelengths per link that are reserved for monitoring link failures), the total number of cycles in the FD set, and the average detection time for a link failure.

Figure 16 depicts the average number of cycles per link versus $\mathcal{M}$. Note that this number decreases with $\mathcal{M}$ when information is exchanged among the monitoring locations. When no information exchange takes place, the average number of links per cycle is affected by two factors: the number of monitoring locations employed and the number of cycles that pass through a link that are not used to detect the failure of that link[8]. In this case, the number of cycles consumed per link first decreases and then increases with $\mathcal{M}$. The initial decrease is due to the use of multiple monitoring locations. The subsequent increase is due to the excessive number of cycles that pass through links but do not contribute to these links' failure detection. Observe that for a large number of monitoring locations, information exchange among locations can save up to 10% of network resources. The use of paths and cycles further optimizes the performance by significantly reducing the average number of consumed wavelengths.

Figure 17 shows the performance of the MC-M algorithm in terms of the number of cycles required to construct the FD set. Notice that this number is equivalent to the number of monitors required for failure detection. If the monitoring locations share information about cycle failures, then with a large number of monitoring locations, approximately 20% of the cost associated with employing monitors can be saved. Notice that by using paths and cycles for fault detection, the total number of wavelengths consumed (which is directly related to the number of monitors required) is less than the case when monitoring locations do not share information.

In Figure 18, we show the average detection time in the two cases. The detection time when information is not exchanged depends on the delay associated with the longest cycle that passes through the link and its monitoring location. When information is exchanged between monitoring locations, the detection time of a given link is given by the sum of the delay of the longest cycle that passes through the link and the time to disseminate information among the multiple monitoring locations that are responsible for detecting the failure of that link. We assume that the processing time at a monitoring location is negligible compared with the delay associated with traversing a cycle in the network. When failure information is exchanged between monitoring locations, the detection time first increases with $\mathcal{M}$ and then decreases. The initial increase is due to the additional delay associated with sharing information among monitoring nodes. Note that if a monitoring location can localize a link failure by just observing the cycles associated with it, then the information may not be shared. The subsequent decrease is due to the decrease in the average length of the longest cycle in the syndrome of a link. When

[8]These cycles are used by other monitoring locations.

information is not shared, the average detection time decreases with $\mathcal{M}$.

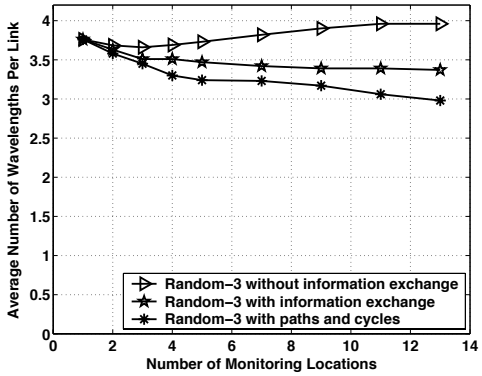


Fig. 16. Average number of cycles per link vs. the number of monitoring locations (Random-3 topology)

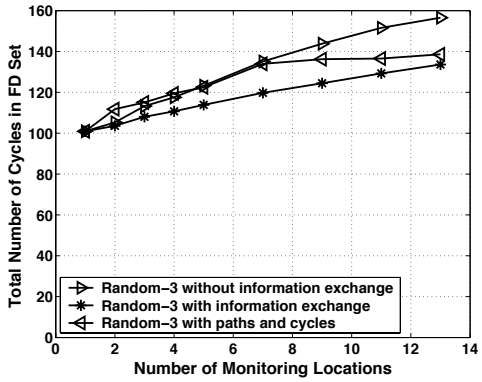


Fig. 17. Total number of cycles in the FD set vs. $\mathcal{M}$ (Random-3 topology).

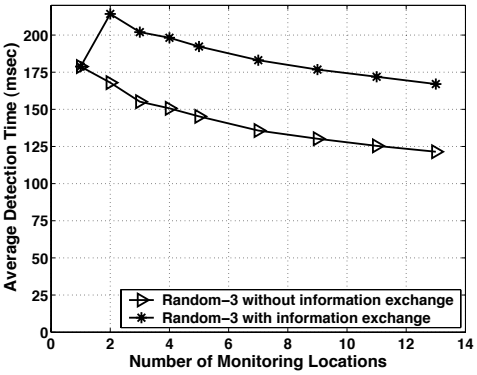


Fig. 18. Average detection time vs. $\mathcal{M}$ (Random-3 topology).

C. Monitoring an Arbitrary Network Topology

In this part, we use randomly generated Waxman's topologies of different average node degrees. For a given topology, we determine the required number of monitoring locations by using the procedure presented in Section IV. For a three-edge-connected component that requires a monitor, an internal node is chosen randomly to be the monitor. For the chosen set of monitors, we use the network transformation presented in Section III-A and then apply the `MC-1` algorithm to find the set of paths and cycles needed to monitor all single-link failures in the network. For each topology, we report the number of required monitoring locations and the average number of cycles per link. Figure 19 shows that the average number of required monitors decreases with the average node degree. Intuitively, this is true because the average size of the three-edge-connected component increases with the increase in average node degree. Figure 20 shows the required average cycles per link for detecting all single-link failures. We make the following observations from Figures 19 and 20:

1) For sparse networks (average node degree ≤ 2), the average number of required cycles per link is significantly small. This is because for such networks, the number of needed monitoring locations is large. In this case, many links get monitored by their adjacent nodes.
2) The average number of required cycles increases with the average node degree.
3) For dense topologies, less than 30% of nodes require a monitor, which is a clear improvement over the technique presented in [6], in which all nodes need to have a monitor to detect all single-link failures.
4) For small networks (≤ 80 nodes), less than three wavelengths are used, on average, for monitoring purposes. This shows the effectiveness and low overhead associated with the presented monitoring technique. For large networks (100 nodes), an average of four wavelengths per link are required.

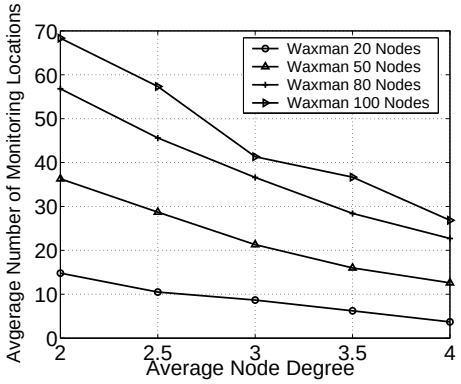


Fig. 19. Average number of monitors vs. average node degree.

VI. Conclusions

We presented a novel approach for unique identification of single-link failures in AONs using MCs and MPs. When only one monitoring location is employed, we proved that the network needs to be three-edge connected to be able to uniquely identify all single-link failures. For such a network, we showed that $\mathcal{O}(|\mathcal{L}|^2)$ is the upper bound and $\mathcal{O}(\log(|\mathcal{L}|))$ is the lower bound on the number of required cycles. We described an ILP formulation and a heuristic approach (`MC-1` heuristic) to determine the FD set. We then considered the problem of fault localization using multiple monitoring locations with and

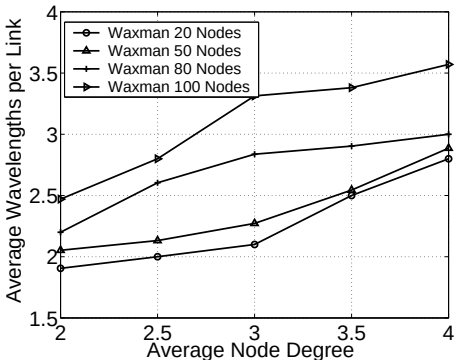


Fig. 20. Average number of monitors vs. average node degree.

without information exchange between monitoring locations. We proposed a heuristic (MC-M heuristic) approach to find the FD set for multiple monitoring locations. Finally, we described a fault-detection scheme that can uniquely identify all single-link failures in an arbitrarily connected network. We provided a necessary and sufficient condition on the number of monitors, and an $\mathcal{O}(|\mathcal{L}|)$ algorithm to calculate the minimum number of monitoring locations. Simulation results confirm the effectiveness of the proposed monitoring technique and the presented solutions.

Our treatment was limited to single-link failures, which affect a single or a specific subset of wavelengths within a link (e.g., optical cross-connect port blocking and intrusion). It is possible for multiple links to fail simultaneously due to the failure of a common resource (e.g., a fiber cut affects all the wavelengths that pass through the link). Such a failure is known as Shared Risk Link Group (SRLG) failure, and will be dealt with in our future work.

APPENDIX I
IMPLEMENTATION DETAILS FOR THE MC−1 ALGORITHM

When a cycle c is added, it is assigned a tag value. The tag for the ith cycle that is added to the solution is set to 2^{i-1}. For each link $\ell \in \mathcal{L}$, we maintain a tag t_ℓ that denotes the sum of the tags of cycles selected thus far and that traverse link ℓ. At any step in the algorithm, if two links have the same tag value, then they are covered by the same set of cycles and hence their failures cannot be uniquely identified. The algorithm ensures that no two links have the same tag value.

The input to the algorithm is a network graph, $\mathcal{G}(\mathcal{N}, \mathcal{L})$, a monitoring location m, the set of links $\mathcal{L}_m$ that the network provider wants to monitor, and the initial tag value U (set to 0). The algorithm starts by finding the initial cover. After a cycle is selected, the algorithm increments by a large value the weights $w(.,.)$ of all the links associated with that cycle. This is done to encourage the spreading of cycles to all links. It results in an efficient initial cover. The use of link weights and the fact that shortest cycles based on $w(.,.)$ are chosen help in finding cycles that pass through links not covered by any other cycle. In Step 3, the algorithm determines if there are two links that have the same tag value, i.e., are associated with the same set of cycles[9]. If so, then one of these links is chosen randomly, say ℓ. For all $\ell' \in \mathcal{L},\ \ell \neq \ell'$, the weights of the links in the network are adjusted as follows:

$$w_{\ell'} = \begin{cases} V_\infty, & \text{if } t_{\ell'} = t_\ell \\ 1, & \text{otherwise} \end{cases} \quad (1)$$

where V_∞ denotes a large value. The algorithm now finds the least-weight cycle that traverses the monitoring location and link ℓ. A pseudocode for the algorithm is shown in Figure 21.

Procedure MC-1$(\mathcal{G}(\mathcal{N}, \mathcal{L}),\ m, \mathcal{L}_m)$
1) $\forall \ell,\ t_\ell = 0,\ U = 0.$
2) $\{\mathcal{C},\ \{t_\ell\},\ U\}$ = AddCycles $(\mathcal{G}(\mathcal{N}, \mathcal{L}), m, \{t_\ell\}, \mathcal{L}_m, U).$

Procedure AddCycles$(\mathcal{G}(\mathcal{N}, \mathcal{L}),\ m,\ \{t_\ell\},\ \mathcal{L}_m, U)$
1) Initialize $\mathcal{C} = \phi$, $w_\ell = 1\ \forall \ell \in \mathcal{L}_m$, and $w_\ell = |\mathcal{N}|\ \forall \ell \in \mathcal{L} \setminus \mathcal{L}_m$
2) While $\exists\ \ell \in \mathcal{L}_m$ such that $t_\ell = 0$
   a) $c =$ FindCycle$(\mathcal{G}, m, \ell, \{w_\ell\})$
   b) $\mathcal{C} = \mathcal{C} \cup \{c\};\ U = U + 1;$
   c) $\forall \ell' \in c,\ w_{\ell'} = w_{\ell'} + V_\infty;\ t_{\ell'} = t_{\ell'} + 2^U$

   Note: Initial cover is now ready
3) While $\exists \{\ell, \ell'\} \in \mathcal{L}^2$ s.t. $t_\ell = t_{\ell'},\ t_\ell > 0,\ t_{\ell'} > 0,$
   a) Define $J = \{\ell'' : t_{\ell''} \neq t_\ell, \ell'' \in \mathcal{L}\}$
   b) $\forall \ell \in J, w'_\ell = 1.$
   c) $\forall \ell \in \mathcal{N} - J, w'_\ell = V_\infty.$
   d) $c =$ FindCycle $(m, \ell, w'(.), \mathcal{L}).$
   e) $\mathcal{C} = \mathcal{C} \cup \{c\},\ U = U + 1$
   f) $\forall \ell' \in c, t_{\ell'} = t_{\ell'} + 2^U$

Procedure FindCycle$(m, \ell, w(.), \mathcal{L})$
Finds a cycle passing through m and ℓ using the procedure mentioned in Figure 2.

Fig. 21. Pseudocode for the fault-detection algorithm with one monitoring location and a 3-edge-connected network.

APPENDIX II
IMPLEMENTATION DETAILS FOR THE MC−M ALGORITHM

As shown in Figure 22, the input to MC-M is a set of monitoring locations $\mathcal{M}$ and a flag I_{ex} that indicates whether the monitoring locations exchange information ($I_{ex} = 1$) or not ($I_{ex} = 0$). For each $m \in \mathcal{M}$, the algorithm starts by finding its cloud $\mathcal{L}_m$. It executes Dijkstra's algorithm to find the shortest-path tree (w.r.t. hop length) that is rooted at m. Let $D_m[n]$ be the weight of the shortest path from m to any node n. For a link $\ell = (x_\ell, y_\ell)$, let m_{x_ℓ} and m_{y_ℓ} denote the nearest monitoring locations to nodes x_ℓ and y_ℓ, respectively. For each link $\ell \in \mathcal{L}$, let m_ℓ denote the monitoring location associated with this link, chosen randomly from m_{x_ℓ} and m_{y_ℓ}. Let $\mathcal{C}_m$ denote the set of cycles associated with node m.

For each monitoring location m, MC-M finds a set of cycles by running the AddCycles procedure on $\mathcal{L}_m$ as discussed in Section II-C.2. The weights of all the links that are not part of node m's cloud are set to $|\mathcal{N}|$ so as to discourage the use of

[9]We maintain a list of the network links which do not have unique tag values, sorted according to their tag values. The sorted list is updated in $\mathcal{O}(|\mathcal{L}| \log(|\mathcal{L}|))$ time whenever link tag values are to be modified following the addition of a cycle of length $\mathcal{O}(|\mathcal{L}|)$. Step 3 in Figure 21 can then be executed in $\mathcal{O}(|\mathcal{L}|)$ by comparing the key values of adjacent members in this list. Note that the link tag values are modified only when a cycle is added to the solution set $\mathcal{C}$.

these links in the cycles returned by the `AddCycles` procedure. If monitoring locations do not share information, then the tag values for all the links are reset to zero before calling `AddCycles`. This way, a monitoring location has no previous information about the cycles used by other monitoring locations. The tag values are retained between successive calls to `AddCycles` if information is shared between monitoring locations. In Figure 22, U is the cycle number associated with the added cycle during multiple calls to `AddCycles`.

In computing the cycles that traverse a monitoring location m, the use of links outside m's cloud is discouraged but not eliminated. In some cases, it is possible that the tag value associated with a link outside m's cloud is unique w.r.t. monitoring location m, i.e., the link failure can be uniquely identified by the monitoring location. In such a case, the link is transferred to m's cloud.

Procedure `MC-M`($\mathcal{G}(\mathcal{N},\mathcal{L})$, $\mathcal{M}$, I_{ex})

1) Initialize: $U = 0$; $\forall \ell \in \mathcal{L}, t_\ell = 0$; $\mathcal{C} = \phi$.
2) Cloud Formation:
   a) $\forall \ell \in \mathcal{L}, w_\ell = 1$.
   b) $\forall m \in \mathcal{M}$, D_m = `Dijkstra`$(\mathcal{G}(\mathcal{N},\mathcal{L}), m, w(.))$
   c) $\forall u \in \mathcal{N}$,
      Set $m_u = r$ s.t. $D_r[u] = \min_{x \in S}(D_x[u])$.
   d) $\forall \ell \in \mathcal{L}$, select m_ℓ randomly from $\{m_{x_\ell}, m_{y_\ell}\}$.
   e) $\forall x \in \mathcal{M}$, $C[x] = \{\ell : m_\ell = x\}$.
3) Cycle Formation:
   a) $\forall m \in \mathcal{M}$,
   - $\forall \ell \in \mathcal{L}$,
     If $(I_{ex} == 0)$, set $t_\ell = 0$.
     If $\ell \in C[m]$, set $w_\ell = 1$
     else $w_\ell = |\mathcal{N}|$.
   - $\{\mathcal{C}_x, t(.), U\}$ =
     `AddCycles`$(\mathcal{G}, m, w(.), t(.), C[m], U)$
   - $\mathcal{C} = \mathcal{C} \cup \mathcal{C}_x$
   - $Tr = \{\ell' : m_{\ell'} \neq x, t_{\ell'} > 0$, and $t_{\ell'} \neq t_{\ell''}, \forall \ell'' \in C[x]\}$
   - $\forall \ell'' \in Tr$ and $I_{ex} == 0, M_{\ell''} = x$.

Fig. 22. Pseudocode for the `MC-M` algorithm.

Acknowledgement: Authors would like to thank Ravi Balasubramanian for his help in simulations.

REFERENCES

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network flows: Theory, Algorithm, and Applications*. Prentice Hall Inc., 1993.

[2] J. A. Bondy and U. S. R. Murthy. *Graph Theory with Applications*. New York: American Elsevier Publishing, 1976.

[3] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. Prentice Hall Inc., 1998.

[4] CPLEX. http://www.cplex.com.

[5] Z. Galil and G. F. Italiano. Maintaining the 3-edge-connected components of a graph on-line. *SIAM Journal on Computing*, 22(1):11–28, 1993.

[6] Y. Hamazumi, M. Koga, K. Kawai, H. Ichino, and K. Sato. Optical path fault management in layered networks. *Proceedings of the IEEE Globecom Conference*, 4:2309–2314, Nov. 1998.

[7] N. Harvey, M. Patrascu, Y. Wen, S. Yekhanin, and V. W. S. Chan. Non-adaptive fault diagnosis for All-optical networks via combinatorial group testing on graphs. *Proceedings of the IEEE INFOCOM Conference*, May 2007.

[8] J. Hopcraft and R. E. Tarjan. Dividing a graph into triconnected components. *SIAM Journal of Computing*, 2(3):135–158, 1973.

[9] S. Maclane. A structural characterization of planar combinatorial graphs. *DUKE Math Journal*, 3(3):460–472, 1937.

[10] V. Raghavan and A. Tripathi. Sequential diagnosability is Co-NP complete. *IEEE Transactions on Computers*, 40(5):584–595, May 1991.

[11] B. M. Waxman. Routing of multipoint connections. *IEEE Journal on Selected Areas in Communications*, 69:1617–1622, Dec. 1988.

[12] Y. Wen, V. W. S. Chan, and L. Zheng. Efficient fault diagnosis algorithms for All-optical WDM networks with probabilistic link failures. *IEEE/OSA Journal of Lightwave Technology*, 23(10):3358–3371, Oct. 2005.

[13] H. Zeng, C. Huang, and A. Vukovic. Monitoring cycles for fault detection in meshed all-optical networks. *International Conference on Parallel Processing Workshop (ICPP)*, 1:434–439, Aug. 2004.

Satyajeet S. Ahuja received B.E. (Hons.) degree in Electronics and Communication Engineering from Maulana Azad College of Technology (MACT), Bhopal, India, in 1999. He received his M.E. (Hons.) degree in Electrical and Communication Engineering (Telecommunications) from Indian Institute of Science (IISc), Bangalore, India, in 2002 and Ph.D. degree in ECE from the University of Arizona, Tucson, USA in 2008. He held research positions at Google; Tejas Networks, Bangalore, India; and VSNL, Bangalore, India. He is currently working at Infinera Corporation, Sunnyvale, CA, USA. His research interests include design and analysis of algorithms for optical and wireless networks, network monitoring, fault localization, network design, routing, media streaming, and protocol design.

Srinivasan Ramasubramanian (M'02-SM'08) received the B.E. (Hons.) degree in Electrical and Electronics Engineering from Birla Institute of Technology and Science (BITS), Pilani, India, in 1997, and the Ph.D. degree in Computer Engineering from Iowa State University, Ames, in 2002. He is currently an Associate Professor in the Department of Electrical and Computer Engineering at the University of Arizona, where he held the position of Assistant Professor from August 2002 to July 2008. He is a co-developer of the Hierarchical Modeling and Analysis Package (HIMAP), a reliability modeling and analysis tool, which is currently being used at Boeing, Honeywell, and several other companies and universities. His research interests include architectures and algorithms for optical and wireless networks, multipath routing, fault tolerance, system modeling, and performance analysis. He has served as the TPC Co-Chair of BROADNETS 2005 and ICCCN 2008 (Optical Networking Symposium) conferences and is an editor of the Springer Wireless Networks Journal.

Marwan M. Krunz is a professor of electrical and computer engineering at the University of Arizona. He received the Ph.D. degree in EE from Michigan State University in 1995. From 1995 to 1997, he was a postdoctoral research associate with the department of computer science, University of Maryland, College Park. His recent research interests include routing and MAC design for wireless networks, cognitive radios, fault detection in optical networks, traffic modeling, and media streaming. He has published more than 120 journal articles and refereed conference papers in these areas. He received the NSF CAREER Award (1998-2002). He currently serves on the editorial board for the IEEE/ACM Transactions on Networking, the IEEE Transactions on Mobile Computing, and the Computer Communications Journal. He served as a Technical Program Chair for the IEEE INFOCOM 2004, the IEEE SECON 2005, and the IEEE WoWMoM 2006 Conferences.