M^2-CYCLE: an Optical Layer Algorithm for Fast Link Failure Detection in All-Optical Mesh Networks

Bin Wu and Kwan L. Yeung
Dept. of Electrical and Electronic Engineering
The University of Hong Kong
Pokfulam, Hong Kong
E-mail: {binwu, kyeung}@eee.hku.hk

***Abstract*—To achieve fast link failure detection in all-optical networks, the notion of monitoring-cycle (m-cycle) is introduced. The best known m-cycle construction algorithm (HST [7]) adopts a spanning tree-based approach. In this paper, we propose a new algorithm M^2-CYCLE to construct a set of *minimum-length* m-cycles (or m^2-cycles) for more efficient link failure detection. We prove that the performance of M^2-CYCLE is never worse than any spanning tree-based approach. Comparing M^2-CYCLE to the existing algorithms, we show that it uses the least amount of network resources (measured by the number of cycles, cover length and monitoring wavelength requirement) to achieve the most accurate link failure detection (measured by localization degree).**

Keywords-All-optical networks (AONs); cycle cover; fast link failure detection; monitoring-cycle.

I. INTRODUCTION

With the rapid progress of optical technologies, the communication infrastructure continuously evolves towards *all-optical networks* (AONs). In WDM (wavelength division multiplexing) optical networks, hundreds of wavelengths can be multiplexed onto a single fiber for efficient transmission. Therefore, fiber-cuts cause great data loss. In order to provide protection or restoration, fast link failure detection is a priori.

Link failure detection in AONs can be implemented at different protocol layers, e.g. physical/optical or network layer. In fact, most network layer routing protocols (such as OSPF and IS-IS) already have built-in fault detection mechanisms [1]. To accelerate the detection speed, cross-layer design is also proposed [2]. Nevertheless, such techniques can only render a detection time in seconds, which is much longer than the typical requirement of 50 ms [3] for optical recovery. Therefore, optical layer schemes are preferred. On the other hand, those schemes designed for traditional optical networks (e.g. SDH/SONET) cannot be transplanted to AONs because of the lack of electrical terminations [4-5].

At the optical layer, a fault can be detected by measuring optical power, analyzing optical spectrum, using pilot tones or optical time domain reflectometry (OTDR) [6-7]. This is carried out by a special optical device called *monitor* [8-10]. A channel-based monitoring scheme uses one monitor for each wavelength channel of a link, thereby requiring a very large number of monitors. A link-based monitoring scheme is more scalable, but still requires one monitor per link.

To further reduce the number of required monitors, the notion of *monitoring-cycle* (m-cycle) [6-7] is introduced. An m-cycle is implemented by assigning a dedicated loop-back supervisory wavelength to spy on the links along it. The basic idea is to find a *cycle cover*, defined as a set of m-cycles $\{c_1, c_2, \ldots, c_M\}$ that cover every link in the network, and assign a monitor to each m-cycle. Each link may be covered by more than one m-cycles. If a particular link fails, it triggers alarms in all the m-cycles covering this link. For a cycle cover with size M, an *alarm code* is of format $[a_1, a_2, \ldots, a_M]$, where a_i=1 if m-cycle c_i alarms and a_i=0 otherwise. The location of the fault can then be identified by decoding the alarm code. For example, the network in Fig. 1 is covered by M=13 m-cycles. If link 7－10 fails, alarm code [0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1] will be generated due to the fault detection by monitors on m-cycles c_9, c_{10}, c_{12}, and c_{13}. Similarly, if the fault is detected by monitors on c_2, c_8, and c_{10}, there must be a fault at link 7－6.

Based on the above idea, three algorithms (HDFS, SPEM [6] and HST [7]) are proposed to construct m-cycles. The objective is to localize the fault and minimize the network resource consumption (monitors, wavelengths, etc). Among them, HST constructs m-cycles based on a carefully designed *spanning tree*, and it delivers the best performance [7].

In this paper, we propose to construct a set of *minimum-length* m-cycles (or m^2-cycles) for fast link failure detection at optical layer. The length of a cycle is defined as the number of links it covers. Our algorithm is called M^2-CYCLE. For simplicity, we focus on networks connected by single fiber links although a multi-fiber extension is possible. Besides, we assume that the network has no *single-bridge link*, as it can separate the network into two unconnected parts if it is removed. Note that such single-bridge links are usually avoided in the network design [7]. We prove that the performance of M^2-CYCLE is never worse than *any* spanning tree-based approach, *no matter how the spanning tree is constructed*. Numerical results show that M^2-CYCLE requires much less network resources than HST [7].

The rest of the paper is organized as follows. In Section II, we review the spanning tree-based approach, and discuss the performance metrics. In Section III, M^2-CYCLE is presented, and its properties are proved in Section IV. Discussions are given in Section V and we conclude the paper in Section VI.

II. SPANNING TREE-BASED M-CYCLE CONSTRUCTION AND PERFORMANCE METRICS

A. Spanning Tree-Based M-cycle Construction

HST [7] constructs m-cycles based on a spanning tree. The spanning tree roots at the node with the maximum degree, and always extends at the nodes with the maximum number of neighbors that are not yet included in the tree. Let node 7 in Fig. 1 (taken from [7]) be the root. All links incident on node 7

This work is supported by Hong Kong Research Grant Council Earmarked Grant HKU 7150/04E.

are first added to the tree. The tree then extends at node 9, and the corresponding links 3—9, 4—9 and 5—9 are added. This process continues until a spanning tree is built. Links in the spanning tree are called *trunks* (denoted by bold lines), and other links are called *chords*. HST generates an m-cycle from each chord, along which all other links are trunks. For example, the m-cycle generated from chord 5—6 is c_8: 5—6—7—9—5.

B. Performance Metrics

To evaluate the performance of m-cycle construction algorithms, the following metrics are used.

- *Localization degree (D_L):* Ideally, there should be a one-to-one mapping between the set of alarm codes and the set of links to be monitored, such that a particular alarm code indicates a unique link failure. However, some alarm codes may not be able to localize the fault to a particular link.[1] To measure the accuracy of the fault detection, localization degree D_L is defined as $D_L = L/A$, where L is the total number of links to be monitored, and A is the size of the alarm code set. Note that we only consider a single link failure at a time. So, A cannot be larger than L. A smaller D_L means a better fault localization. Ideally, $D_L=1$.
- *Number of cycles (M):* Because one monitor is assigned to each m-cycle, obviously we want to minimize the number of required monitoring cycles M.
- *Cover length (L_C):* It is the total number of supervisory wavelength-links required, i.e. the total bandwidth for monitoring. $L_C = \sum_i L_i$ where L_i is the length of m-cycle c_i.
- *Monitoring wavelength requirement (W):* Each m-cycle requires a dedicated wavelength channel on each link it covers. Let t_i be the number of monitoring cycles that cover link i. Then, $W = \max_i\{t_i\}$ is the (maximum) monitoring wavelength requirement.

III. M²-CYCLE ALGORITHM

To construct m²-cycles *based on a given link*, we can *temporarily* remove this link and then calculate all the shortest paths between its two end nodes. Combining each shortest path found with the given link, a set of m²-cycles can be obtained. For example, the m²-cycles based on link 7—10 in Fig. 2 are 7—6—10—7, 7—8—10—7 and 7—9—10—7.

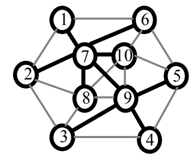
M²-CYCLE algorithm is summarized in Fig. 5. It is designed based on the above m²-cycle construction and consists of two main operations: *expansion* and *refinement*. Expansion is to construct a *base set* of m²-cycles, and refinement is to remove/add any redundant/missing m²-cycles from/to the base set.

A. Expansion: Constructing the Base Set

Initially, all the links are marked as *uncovered*, and the base set $\boldsymbol{B}$ is null ($\boldsymbol{\Phi}$). Based on each link, m²-cycles are constructed and put into a list $\boldsymbol{\Theta}$ in ascending order of their lengths.

First, we scan through $\boldsymbol{\Theta}$ and find the first m²-cycle that traverses some uncovered links. These uncovered links form a set $\boldsymbol{F}$. We then enter the inner-loop expansion iteration (Steps 2b—2d in Fig. 5) with set $\boldsymbol{F}$. A running set $\boldsymbol{T}$ is initialized to $\boldsymbol{\Phi}$. For each link in $\boldsymbol{F}$, we find its associated m²-cycles. If an

[1] E.g. the faults at 8—13 and 13—14 in ARPA2 in Fig. 8 have identical alarm codes, because any cycle covering 8—13 must also cover 13—14.

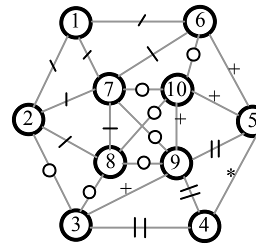


m-cycles

chord	chord	chord
c_1: **1−2**−7−1	c_5: **3−4**−9−3	c_9 : **5−10**−7−9−5
c_2: **1−6**−7−1	c_6: **3−8**−7−9−3	c_{10}: **6−10**−7−6
c_3: **2−3**−9−7−2	c_7: **4−5**−9−4	c_{11}: **8−9**−7−8
c_4: **2−8**−7−2	c_8 : **5−6**−7−9−5	c_{12}: **8−10**−7−8
		c_{13}: **9−10**−7−9

Performance metrics: D_L=1.000, M=13, L_C=43, W=6

Fig. 1. HST for SmallNet (10 nodes, 22 links).



Base set of m²-cycles

c_1: 1−2−7−1	c_8 : 5−6−10−5
c_2: 2−7−8−2	c_9 : 7−9−10−7
c_3: 7−1−6−7	c_{10}: 8−9−3−8
c_4: 6−7−10−6	c_{11}: 9−10−5−9
c_5: 7−8−10−7	c_{12}: 3−4−9−3
$\boxed{c_6\text{: 7−8−9−7}}$	c_{13}: 4−5−9−4
c_7: 2−3−8−2	c_6 is removed by the refinement operation

Round	$\boldsymbol{B}$ and $\boldsymbol{F}$
	$\boldsymbol{F}$={1−2, 1−7, 2−7};
1st	$\boldsymbol{B}$={$\underline{c_1, c_2, c_3}$}, $\boldsymbol{F}$={1−2, 1−7, 2−7, 1−6, 6−7, 7−8, 8−2};
2nd	$\boldsymbol{B}$={$c_1, c_2, c_3, \underline{c_4, c_5, c_6, c_7}$},
	$\boldsymbol{F}$={6−10, 10−8, 8−3, 3−2, 7−10, 7−9, 8−9};
3rd	$\boldsymbol{B}$={$c_1, c_2, c_3, c_4, c_5, c_6, c_7, \underline{c_8, c_9, c_{10}}$},
	$\boldsymbol{F}$={6−5, 5−10, 10−9, 9−3};
4th	$\boldsymbol{B}$={$c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9, c_{10}, \underline{c_{11}, c_{12}}$},
	$\boldsymbol{F}$={5−9, 9−4, 4−3};
5th	$\boldsymbol{B}$={$c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9, c_{10}, c_{11}, c_{12}, \underline{c_{13}}$}, $\boldsymbol{F}$={4−5}.

Fig. 2. Expansion in M²-CYCLE (starts from c_1 and the links in $\boldsymbol{F}$ after each round are marked in the topology.).

m²-cycle covers at least one uncovered link, add it to $\boldsymbol{B}$. At the same time, mark all the *newly* covered links as *covered*, and add them to $\boldsymbol{T}$. Define this inner-loop iteration as a *round*. At the end of each round, we update $\boldsymbol{F}$ by setting $\boldsymbol{T} \rightarrow \boldsymbol{F}$, and proceed to the next round with the updated $\boldsymbol{F}$. This process continues until we cannot add any new m²-cycle to $\boldsymbol{B}$. That is, the m²-cycles associated with each link in $\boldsymbol{F}$ do not cover any new uncovered links. Then by repeating the above process, we scan through $\boldsymbol{\Theta}$ again and find the next m²-cycle that has some uncovered links. If none can be found, the expansion ends.

Fig. 2 shows an example for the SmallNet. Compared with the result in Fig. 1, a different set of cycles are found.

B. Refinement 1: Removing Redundant m²-cycles

A careful study on Fig. 2 shows that, if we remove c_6: 7—8—9—7 (or c_9: 7—9—10—7) from $\boldsymbol{B}$, any link failure can still be identified by a unique alarm code. To identify and remove such redundant m²-cycles, we first construct an alarm code table $\boldsymbol{T_A}$ from $\boldsymbol{B}$, as in Fig. 3. For each column/m²-cycle in $\boldsymbol{T_A}$, we shadow it and check if there are any all-zero rows or identical alarm codes in the not-shadowed part. If there are all-zero rows, it means that the corresponding links are covered only by this m²-cycle, and thus it is not redundant. If there are identical alarm codes, we check if some of them become different from others after removing the shadow. If yes, then this m²-cycle is not redundant. Otherwise, this *redundant* m²-cycle is removed from $\boldsymbol{B}$, and the corresponding column is deleted from $\boldsymbol{T_A}$. We then repeat this process until all m²-cycles in $\boldsymbol{T_A}$ are checked. In Fig. 3, only c_6 is removed from $\boldsymbol{B}$.

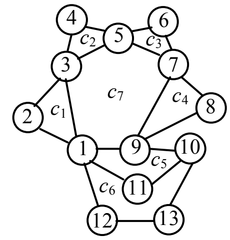
C. Refinement 2: Adding Missing m²-cycles

On the other hand, the m²-cycles in $\boldsymbol{B}$ may not be sufficient to identify all the link failures that *should* be identified. In Fig.

1-4244-0357-X/06/$20.00 ©2006 IEEE
This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE GLOBECOM 2006 proceedings.

	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}	c_{12}	c_{13}
1−2	1												
1−6			1										
1−7	1		1										
2−7	1	1											
2−8		1					1						
2−3							1						
3−8							1			1			
3−9										1		1	
3−4												1	
4−9												1	1
4−5													1
5−9											1		1
5−10								1			1		
5−6								1					
6−10				1				1					
6−7			1	1									
7−8		1			1	1							
7−9						1			1				
7−10				1	1				1				
8−9						1				1			
8−10					1								
9−10									1		1		

Fig. 3. T_A for the solution in Fig. 2. A "1" indicates that an m^2-cycle covers the corresponding link, and 0*s* are omitted.



Base set of m^2-cycles

c_1: 1−2−3−1
c_2: 3−4−5−3
c_3: 5−6−7−5
c_4: 7−8−9−7
c_5: 1−9−10−11−1
c_6: 1−9−10−13−12−1 or 1−11−10−13−12−1
c_7: 1−3−5−7−9−1

Fig. 4. Some m^2-cycles may be missing from the base set ***B***.

4, assume m^2-cycles c_1 to c_5 are added to ***B***. Then, no matter how c_6 is chosen (either 1—9—10—13—12—1 or 1—11—10—13—12—1), all the links are covered but the faults at 1—9 and 9—10 are still indistinguishable due to the same alarm code ([0, 0, 0, 0, 1, 1] or [0, 0, 0, 0, 1, 0], depending on which c_6 is used).

To address this issue, we construct an m^2-cycle based on link 9—10 by *temporarily* removing 1—9, and vice versa. *If such two m^2-cycles exist*, we add the one with shorter length to ***B***. (Otherwise the two faults are indistinguishable by any cycle-based monitoring scheme.) T_A is then updated by the new m^2-cycle. In Fig. 4, c_7: 1—3—5—7—9—1 is added and the first c_6 is used. Then the two faults can be identified by alarm codes [0, 0, 0, 0, 1, 1, 1] and [0, 0, 0, 0, 1, 1, 0], respectively.

In practice, if some links have identical alarm codes in T_A, we use this process for possible adding of an extra m^2-cycle.

IV. Properties of M^2-CYCLE

Theorem 1: M^2-CYCLE gives a *cycle cover* C_M to cover every link in the network.

Proof: Because the network is connected and has no single-bridge link, any uncovered link can be identified in Step 2e of M^2-CYCLE, and then covered by adding a new m^2-cycle to ***B***. Note that Step 3b (removing redundant m^2-cycles) cannot turn any *covered* link to an *uncovered* one. Consequently, every link in the network is covered by C_M. #

Theorem 2: The number of m^2-cycles generated by M^2-CYCLE will never be larger than the number of m-cycles generated by any spanning tree-based algorithm.

Proof: Let $G(V, E)$ denote the network and S_O denote an arbitrary spanning tree in $G(V, E)$. Assume that we have a blank sheet on hand called *draft*. Each time when we add an m^2-cycle to the base set ***B***, we also draw it in the draft (In other words, the m^2-cycle is *introduced* to the draft). Our approach is to count the *minimum possible number of chords* that are introduced to the draft. So, *each link in the draft is assumed as a trunk unless we can identify it as a chord.* Since a spanning tree-based algorithm generates an m-cycle from each chord, we can thus compare the number of m^2-cycles generated by M^2-CYCLE with the number of chords in the draft.

M^2-CYCLE ALGORITHM

Input:

A network $G(V, E)$ without single-bridge link.

Output:

A cycle cover C_M={c_1, c_2, …,c_M}, and an alarm code table T_A.

Step 1: Initialization:

Mark all the links in $G(V, E)$ as *uncovered*. Initialize a base set ***B*** to null (Φ). Construct m^2-cycles based on each link in $G(V, E)$ and add them to a list Θ in ascending order of their lengths.

Step 2: Expansion to construct the base set B:

2a) Scan through Θ and find the first m^2-cycle that has some uncovered links. These uncovered links form a set ***F***.

2b) Set $\Phi \rightarrow T$.

2c) Pick up a link from ***F*** and find all m^2-cycles based on it. If an m^2-cycle covers at least one uncovered link, add this m^2-cycle to ***B***. Mark all the links newly covered by this m^2-cycle as *covered* and add them to ***T***. Repeat *Step 2c)* for all the links in ***F***. Then set $T \rightarrow F$.

2d) Repeat *2b)-2c)* until no new m^2-cycle can be added to ***B***.

2e) Check if there are any uncovered links left in $G(V, E)$. If yes, go to *2a)*. Otherwise go to *Step 3*.

Step 3: Refinement to remove/add m^2-cycles:

3a) Construct an alarm code table T_A, where each row denotes an alarm code for a link in $G(V, E)$ and each column represents an m^2-cycle in ***B***. Initialize all the entries in T_A to 0. For each m^2-cycle, find the links it covers and set the corresponding entries to 1 in T_A.

3b) For each column in T_A, shadow it and check if there are all-zero rows or identical alarm codes in the remaining not-shadowed part. If some all-zero rows exist or some identical alarm codes become different from others after removing the shadow, then un-shadow it and check the next column. Otherwise delete it from T_A and the corresponding m^2-cycle from ***B***. Repeat *3b)* until all the columns are checked.

3c) Check if any two links l_a and l_b have identical alarm codes in T_A. If yes, temporarily remove l_a from $G(V, E)$ and find an m^2-cycle based on l_b, and vice versa. If such two m^2-cycles exist, add the one with shorter length to ***B***. Then add it to T_A by creating a new column and set the entries to 1 or 0 according to the links it covers.

3d) Set $B \rightarrow C_M$. The final solution is in C_M, with alarm codes in T_A.

Fig. 5. M^2-CYCLE algorithm.

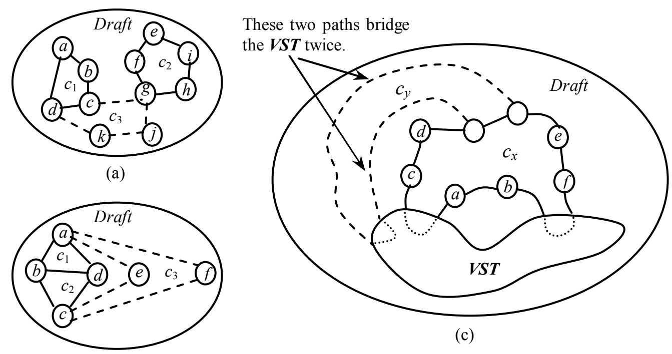


Fig. 6. Three possible draft scenarios.

Our proof is based on the three possible draft scenarios shown in Fig. 6. In particular, Figs. 6a & 6b are for the expansion operation and Fig. 6c targets at the refinement.

In Fig. 6a, when we add the first two m^2-cycles c_1: $a—b—c—d—a$ and c_2: $e—f—g—h—i—e$ to the draft, there must be a chord on each of them. Otherwise we can find a loop in S_O, which is impossible. In the draft, other links (except the two identified chords) form two separate "spanning trees" in c_1 and c_2 respectively. To distinguish such "spanning trees" in the draft from S_O in $G(V, E)$, we call them *virtual spanning trees* (***VST***s). ***VST***s expand (and merge) as more m^2-cycles are added to the draft. (Also note that ***VST***s are not necessarily sub-graphs of $\boldsymbol{S_O}$, as we only use them as an aid to count the minimum number of chords in the draft.) Then, assume a third m^2-cycle c_3: $c—d—k—j—g—c$ is added to the draft and it connects c_1 and c_2. This must introduce another new chord, otherwise we can find a loop in $\boldsymbol{S_O}$ (no matter where the first two chords locate in c_1 and c_2). To minimize the number of chords in the draft, the expanded/merged ***VST*** remains as a spanning tree. *In general, if an m^2-cycle connects two separate **VST**s, then this m^2-cycle also introduces a new chord, and the expanded/merged **VST** remains as a spanning tree in the draft.*

We now consider the scenario in Fig. 6b, where m^2-cycles c_1: $a—b—d—a$, c_2: $c—b—d—c$ and c_3: $f—a—e—c—f$ are added to the draft one by one. Essentially, adding c_2 is equivalent to bridging the two nodes b and d in the ***VST*** (formed by c_1) with a new path $b—c—d$. To avoid any loop in $\boldsymbol{S_O}$ and to minimize the number of chords in the draft, bridging b and d must introduce a new chord, and the expanded ***VST*** (formed by c_1 and c_2) remains as a spanning tree. Then, when c_3 is added, the ***VST*** formed by c_1 and c_2 is actually bridged twice by $a—e—c$ and $a—f—c$. So, c_3 introduces *two* new chords instead of one. *In general, each time when an m^2-cycle is added to an existing **VST**, the number of chords it introduces is equal to the number of times that the **VST** has been bridged.*

Combining the scenarios in Figs. 6a & 6b, the number of m^2-cycles constructed from the expansion operation of M^2-CYCLE does not exceed the number of chords in $\boldsymbol{G(V, E)}$.

We then consider the refinement operation of M^2-CYCLE. Assume that three links $a—b$, $c—d$, and $e—f$ in Fig. 6c are covered by the same set of m^2-cycles and thus the faults are indistinguishable. Let c_x be the *first* m^2-cycle that introduces the three links to the draft, and c_y be an m^2-cycle constructed later. Both c_x and c_y are obtained from the expansion operation. When c_x is constructed, the draft only contains c_x and all the m^2-cycles constructed before c_x (i.e. c_y is not included yet). Then, we temporarily remove any one of the three links and construct an m^2-cycle *in the current draft* based on one of the two remaining links. If such an m^2-cycle cannot be found, then the two links *remain* indistinguishable *in the current draft* (such as $c—d$ and $e—f$). On the other hand, if such a new m^2-cycle is obtained, e.g., $e—f$ is temporarily removed and an m^2-cycle based on $a—b$ can be obtained passing through some nodes in the *original* ***VST***, then c_x must have bridged the original ***VST*** twice and thus should have introduced two new chords. However, we have added only one m^2-cycle c_x to $\boldsymbol{B}$. So, by following Step 3c in Fig. 5 and add an extra m^2-cycle to distinguish the two faults, the number of m^2-cycles still does not exceed the number of chords.

For any links that *remain* indistinguishable, such as $c—d$ and $e—f$, assume the refinement operation finally adds a new m^2-cycle to distinguish them by temporarily removing $e—f$ and constructing an m^2-cycle based on $c—d$. This is possible *only if* there is an m^2-cycle c_y constructed later that covers some nodes or links between d and e (to provide a branch route). Since $c—d$ and $e—f$ are still indistinguishable after c_y is added, c_y must cover *either both or none* of them. In both cases, c_y must have bridged the ***VST*** twice as shown in Fig. 6c (note that c_x is now included in the ***VST***). So again two new chords are introduced by c_y but only one m^2-cycle c_y has been constructed. If we add an extra m^2-cycle to distinguish the two faults, the number of m^2-cycles does not exceed the number of chords in $\boldsymbol{G(V, E)}$.

Combining our proofs above for the three possible draft scenarios in Fig. 6, the number of m^2-cycles generated by M^2-CYCLE will never be larger than the number of m-cycles generated by any spanning tree-based algorithm. In addition, the refinement operation of M^2-CYCLE may further remove some redundant m^2-cycles. #

Theorem 3: Compared to any spanning tree-based m-cycle construction algorithm, the length of each m^2-cycle and the cover length L_C in M^2-CYCLE are both smaller.

Proof: From the expansion of M^2-CYCLE, the length of m^2-cycles constructed based on each link is minimized. In Step 3c of the refinement, if we need to add an extra m^2-cycle to $\boldsymbol{B}$, it is also constructed in a minimum length manner. Note that L_C is the sum of all the cycle lengths. Since the number of m^2-cycles in M^2-CYCLE is not more than that of m-cycles in a spanning tree-based approach, and the length of each m^2-cycle is minimized, obviously L_C in M^2-CYCLE is also smaller. #

Theorem 4: The monitoring wavelength requirement W is not more than that of any spanning tree-based algorithm.

Proof: Let $\boldsymbol{C_M}$ and $\boldsymbol{C_S}$ be the solutions of M^2-CYCLE and a spanning tree-based algorithm, respectively. Because the length of each m^2-cycle in $\boldsymbol{C_M}$ is minimized, we can enlarge some m^2-cycles in $\boldsymbol{C_M}$ and turn $\boldsymbol{C_M}$ into $\boldsymbol{C_S}$.[2] In Fig. 7, c_1: $b—a—e—b$, c_2: $c—b—e—f—c$ and c_3: $d—c—f—g—d$ are three m^2-cycles in $\boldsymbol{C_M}$. c_1 is also an m-cycle in $\boldsymbol{C_S}$ because it covers only one chord. If we replace the chord $b—e$ in c_2 by path $b—a—e$, and the chord $c—f$ in c_3 by path $c—b—a—e—f$, then both c_2 and c_3 are turned into m-cycles.

Without loss of generality, we consider $b—a—e—b$ in Fig. 7. Assume that $b—a$, $a—e$, and $b—e$ are covered by $\boldsymbol{C_M}$ for Δ_1, Δ_2, and x times, respectively. In order to transform $\boldsymbol{C_M}$ to $\boldsymbol{C_S}$, we need to reroute some cycles from $b—e$ to $b—a—e$ for $x-1$ times (note that each chord is covered only once in $\boldsymbol{C_S}$, and $b—a—e$ is the only path in the spanning tree that connects nodes b and e). Then, the number of cover times for each link is:

$$\begin{cases} b—a: & \Delta_1+(x-1) \\ a—e: & \Delta_2+(x-1) \\ b—e: & 1 \end{cases} \qquad (1)$$

Besides $b—e$, other chords (such as $c—f$ in c_3) may also need to be rerouted across $b—a—e$. Let W_M and W_S denote the monitoring wavelength requirement by $\boldsymbol{C_M}$ and $\boldsymbol{C_S}$. We have

$$W_S \geq \max\{\Delta_1, \quad \Delta_2\} + (x-1). \qquad (2)$$

Since the values of Δ_1, Δ_2 and x must be at least one (i.e. each link is covered at least once by $\boldsymbol{C_M}$), we get

$$W_S \geq \max\{\Delta_1, \quad \Delta_2\} \text{ and } W_S \geq x. \qquad (3)$$

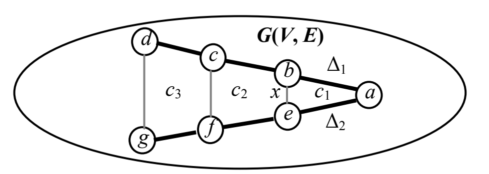


Fig. 7. An m^2-cycle can be enlarged to get an m-cycle.

[2] In fact, a trivial difference may exist due to the removal of redundant m^2-cycles. But it does not affect our proof. Note that the direction of the cycle is not important. e.g. $b—a—e—b$ and $e—a—b—e$ are the same in Fig. 7.

1-4244-0357-X/06/$20.00 ©2006 IEEE
This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE GLOBECOM 2006 proceedings.

Formula (3) indicates that W_S is not smaller than the number of times that $b-a$, $a-e$ and $b-e$ are covered by $\boldsymbol{C_M}$. Note that $b-a-e-b$ is an m-cycle in $\boldsymbol{C_S}$. In fact, we can extend this analysis to any $c_s \in \boldsymbol{C_S}$ (such as $c-b-a-e-f-c$). Since $\boldsymbol{C_S}$ covers every link in the topology, from (3) we have $W_M \le W_S$. #

Theorem 5: The localization degree D_L of M^2-CYCLE is not larger than that in any spanning tree-based algorithm.

Proof: In M^2-CYCLE, if any two links have identical alarm codes, one link is temporarily removed, and an additional m^2-cycle is constructed based on the other link to distinguish the two faults. If this additional m^2-cycle cannot be obtained, any cycle-based algorithm will fail to distinguish the two faults, including any spanning tree-based algorithm. Since a smaller D_L means a better fault localization, D_L in M^2-CYCLE is not larger than that in any spanning tree-based algorithm. #

V. Discussion

Fig. 8 compares M^2-CYCLE with HST [7]. Note that a comparison has been given in [7] to show that HST outperforms HDFS and SPEM [6]. For the three topologies in Fig. 8, M^2-CYCLE achieves the same localization degree D_L as HST, but saves 2.5%, 12.5%, 16.36% on cover length L_C, and 40%, 33.33%, 62.5% on monitoring wavelength requirement W. For the SmallNet in Figs. 1 & 2, one monitor is also saved.

In Fig. 9a, M^2-CYCLE returns a solution with D_L=1, and all dashed links are covered only once using the 9 m^2-cycles c_1~c_9. However, HST needs 10 m-cycles because a redundant m-cycle c_{10} at the center is also included. We call such a center as an *inside track*. Fig. 9b shows that many inside tracks may exist in a large network, each of which introduces a redundant monitor. M^2-CYCLE does not have this problem.

If several nodes form a *segment* (i.e. they are consecutively connected without any bifurcation, e.g. 8－9－10－11 in ARPA2 in Fig. 8), then the link failures cannot be localized to individual links by any cycle-based monitoring scheme. To solve this problem, link-based monitors are needed [7].

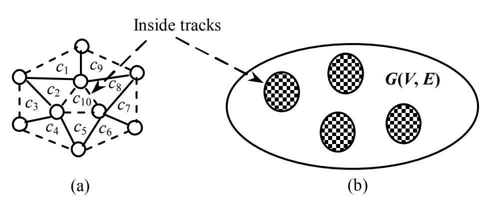


Fig. 9. Many inside tracks may exist in a large network.

VI. Conclusion

Monitoring-cycle (m-cycle) provides an efficient fast link failure detection mechanism in all-optical networks. In this paper, a new algorithm M^2-CYCLE was proposed to construct a set of minimum-length m-cycles (m^2-cycles). We proved that M^2-CYCLE outperforms the existing spanning tree-based approach, no matter how the spanning tree is constructed. Numerical results showed that M^2-CYCLE minimizes the required network resources, and gives the most accurate fault detection. The performance gap to the existing spanning tree-based algorithm increases with the network size.

References

[1] M. Goyal, K. K. Ramakrishnan, and W.-C. Feng, “Achieving faster failure detection in OSPF networks”, *IEEE ICC’03*, 2003.

[2] C. Assi, Y. Ye, A. Shami, S. Dixit and M. Ali, “A hybrid distributed fault-management protocol for combating single-fiber failures in mesh-based DWDM optical networks”, *IEEE Globecom’02*, 2002.

[3] T. Y. Chow, F. Chudak and A. M. Ffrench, “Fast optical layer mesh protection using pre-cross-connected trails”, *IEEE/ACM Trans. on Networking*, vol. 12, no. 3, pp. 539-548, JUN 2004.

[4] Y. Kobayashi, Y. Tada, S. Matsuoka, N. Hirayama, and K. Hagimoto, “Supervisory systems for all-optical network transmission systems”, *IEEE Globecom’96*, pp. 933-937, 1996.

[5] C. Mas and P. Thiran, “A review on fault location methods and their applications in optical networks”, *Optical Network Magazine*, vol. 2, no. 4, 2001.

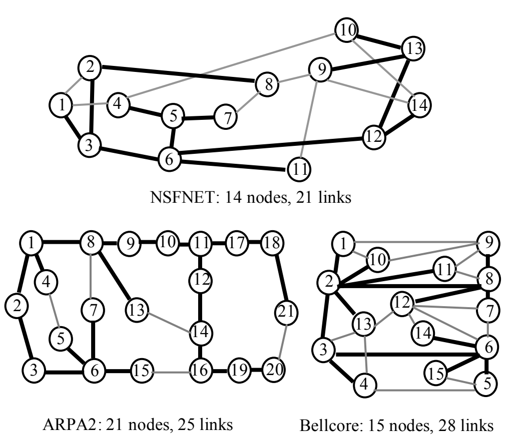
[6] H. Zeng, C. Huang, A. Vukovic and M. Savoie, “Fault detection and path performance monitoring in meshed all-optical networks”, *IEEE Globecom 2004*.

[7] H. Zeng, C. Huang and A. Vukovic, “Spanning tree based monitoring-cycle construction for fault detection and localization in mesh AONs”, *IEEE ICC’05*, 2005.

[8] Y. Hamazumi, M. Koga, K. Kawai, H. Ichino and K. Sato, “Optical path fault management in layered networks”, *IEEE Globecom’98*, vol. 4, pp. 2309-2314, Nov. 1998.

[9] C.-S. Li, and R. Ramaswami, “Automatic fault detection, isolation, and recovery in transparent all-optical networks”, *IEEE J. of Lightwave Tech.*, vol. 15, no. 10, pp. 1784-1793, Oct. 1997.

[10] S. Stanic, S. Subramaniam, H. Choi, G. Sahin and H.-A. Choi, “On monitoring transparent optical networks”, *Int’l Conf. on Parallel Processing Workshops*, pp. 217-223, Aug. 2002.



	NSFNET	ARPA2	Bellcore	
HST	c_1: 1−2−3−1 c_2: 1−4−5−6−3−1 c_3: 4−10−13−12−6−5−4 c_4: 7−8−2−3−6−5−7 c_5: 8−9−13−12−6−3−2−8 c_6: 9−11−6−12−13−9 c_7: 9−14−12−13−9 c_8: 10−14−12−13−10	c_1: 4−5−6−3−2−1−4 c_2: 7−8−1−2−3−6−7 c_3: 13−14−12−11−10−9−8−13 c_4: 15−16−14−12−11−10−9−8−1−2−3−6−15 c_5: 20−21−18−17−11−12−14−16−19−20	c_1: 1−9−8−2−1 c_2: 1−10−2−1 c_3: 3−13−2−3 c_4: 4−5−6−3−4 c_5: 4−13−2−3−4 c_6: 6−7−8−2−3−6 c_7: 6−12−8−2−3−6	c_8: 5−15−6−5 c_9: 7−12−8−7 c_{10}: 8−11−2−8 c_{11}: 9−10−2−8−9 c_{12}: 9−11−2−8−9 c_{13}: 12−13−2−8−12 c_{14}: 12−14−6−3−2−8−12
	D_L=1.105, M=8, L_C=40, W=5	D_L=2.500, M=5, L_C=40, W=3	D_L=1.077, M=14, L_C=55, W=8	
M^2-CYCLE	c_1: 1−2−3−1 c_2: 10−13−12−14−10 c_3: 10−13−9−14−10 c_4: 1−3−6−5−4−1 c_5: 6−11−9−13−12−6 c_6: 8−9−11−6−5−7−8 c_7: 8−9−11−6−3−2−8 c_8: 4−10−14−12−6−5−4	c_1: 1−2−3−6−5−4−1 c_2: 1−2−3−6−7−8−1 c_3: 6−15−16−14−13−8−7−6 c_4: 13−14−12−11−10−9−8−13 c_5: 16−19−20−21−18−17−11−12−14−16	c_1: 1−2−10−1 c_2: 1−10−9−1 c_3: 2−11−8−2 c_4: 11−8−9−11 c_5: 2−3−13−2 c_6: 3−13−4−3 c_7: 5−6−15−5	c_8: 6−7−12−6 c_9: 7−12−8−7 c_{10}: 12−6−14−12 c_{11}: 13−12−8−2−13 c_{12}: 12−13−3−6−12 c_{13}: 3−6−5−4−3 c_{14}: 2−11−9−10−2
	D_L=1.105, M=8, L_C=39, W=3	D_L=2.500, M=5, L_C=35, W=2	D_L=1.077, M=14, L_C=46, W=3	

Fig. 8. Compare M^2-CYCLE with HST [7] (the three topologies are taken from [7]).

1-4244-0357-X/06/$20.00 ©2006 IEEE
This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE GLOBECOM 2006 proceedings.