<table>
<thead>
<tr>
<th><strong>Title</strong></th>
<th>M 2-CYCLE: an optical layer algorithm for fast link failure detection in all-optical mesh networks</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Author(s)</strong></td>
<td>Wu, B; Yeung, LK</td>
</tr>
<tr>
<td><strong>Citation</strong></td>
<td>Proceedings of the Global Telecommunications Conference, 2006 (GLOBECOM 2006), San Francisco, CA, USA, 27 November - 1 December 2006</td>
</tr>
<tr>
<td><strong>Issued Date</strong></td>
<td>2006</td>
</tr>
<tr>
<td><strong>URL</strong></td>
<td><a href="http://hdl.handle.net/10722/98958">http://hdl.handle.net/10722/98958</a></td>
</tr>
<tr>
<td><strong>Rights</strong></td>
<td>Globecom. IEEE Conference and Exhibition. Copyright © IEEE.; ©2006 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.; This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License.</td>
</tr>
</tbody>
</table>
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, ..., 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 \([a_1, a_2, ..., a_M]\), 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 \(c_2, c_6,\) 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...
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, \( D_L = 1.0 \).

- **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, \( L_c = \sum L_i \) 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 \( c_i \). Then, \( W = \max \{t_i\} \) is the (maximum)
monitoring wavelength requirement.

III. \( M^2 \)-CYCLE ALGORITHM

To construct \( M^2 \)-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 \( M^2 \)-cycles can be obtained.
For example, the \( M^2 \)-cycles based on link 7 → 10 in Fig. 2 are
\( 7 \rightarrow 6 \rightarrow 10 \rightarrow 7, 7 \rightarrow 8 \rightarrow 10 \rightarrow 7 \) and \( 7 \rightarrow 9 \rightarrow 10 \rightarrow 7 \).

\( M^2 \)-CYCLE algorithm is summarized in Fig. 5. It is
designed based on the above \( M^2 \)-cycle construction and consists of
two main operations: expansion and refinement. Expansion
is to construct a base set of \( M^2 \)-cycles, and refinement is to remove/add any redundant/missing \( M^2 \)-cycles
from/to the base set.

A. Expansion: Constructing the Base Set

Initially, all the links are marked as uncoversed, and the base
set \( B \) is null \((\Phi)\). Based on each link, \( M^2 \)-cycles are constructed
and put into a list \( \Theta \) in ascending order of their lengths.

First, we scan through \( \Theta \) and find the first \( M^2 \)-cycle that
traverses some uncovered links. These uncovered links form a
set \( F \). We then enter the inner-loop expansion iteration (Steps
\( 2b \rightarrow 2d \) in Fig. 5) with set \( F \). A running set \( T \) is initialized to
\( \Phi \). For each link in \( F \), we find its associated \( M^2 \)-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-cycle covers at least one uncovered link, add it to \( B \). At the
same time, mark all the newly covered links as covered, and
add them to \( T \). Define this inner-loop iteration as a round. At
the end of each round, we update \( F \) by setting \( T \rightarrow F \), and
proceed to the next round with the updated \( F \). This process
continues until we cannot add any new \( M^2 \)-cycle to \( B \). That is,
the \( M^2 \)-cycles associated with each link in \( F \) do not cover any
new uncovered links. Then by repeating the above process,
we scan through \( \Theta \) again and find the next \( M^2 \)-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 \( M^2 \)-cycles

A careful study on Fig. 2 shows that, if we remove \( c_6: 7 \rightarrow 8 \)
\( \rightarrow 9 \) (or \( c_8: 7 \rightarrow 9 \rightarrow 10 \)) from \( B \), any link failure can still
be identified by a unique alarm code. To identify and remove
such redundant \( M^2 \)-cycles, we first construct an alarm code
table \( T_A \) from \( B \), as in Fig. 3. For each column/m-cycle in \( T_A \),
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 \( M^2 \)-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 \( M^2 \)-cycle is not redundant. Otherwise, this redundant
\( M^2 \)-cycle is removed from \( B \), and the corresponding column is
deleted from \( T_A \). We then repeat this process until all \( m^2 \)-cycles
in \( T_A \) are checked. In Fig. 3, only \( c_8 \) is removed from \( B \).

C. Refinement 2: Adding Missing \( M^2 \)-cycles

On the other hand, the \( m^2 \)-cycles in \( B \) may not be sufficient
to identify all the link failures that should be identified. In Fig.
Since a spanning tree-based algorithm generates an m-cycle from each chord, we may find the first chord that has some uncovered links. These uncovered links form a set F.

4) Assume m-cycles $c_1$ to $c_3$ are added to B. Then, no matter how $c_6$ is chosen (either $1-9-10-13-12-10$ or $1-11-10-13-12-10$), 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, 1, 1] or [0, 0, 0, 0, 1, 0], depending on which $c_6$ is used).

To address this issue, we construct an m-cycle based on link 9-10 by temporarily removing 1-9, and vice versa. If such two m-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-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, 1, 1, 1] and [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-cycle.

### IV. PROPERTIES OF M²-CYCLE

**Theorem 1:** M²-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 M²-CYCLE, and then covered by adding a new m-cycle to B. Note that Step 2b (removing redundant m-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-cycles generated by M²-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 $S_d$ 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-cycle to the base set B, we also draw it in the draft (In other words, the m-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-cycles generated by M²-CYCLE with the number of chords in the draft.

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-cycles $c_1$: $a-b-c$ 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 $S_d$, 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 $S_d$ in $G(V, E)$, we call them virtual spanning trees (VSTs). VSTs expand (and merge) as more m-cycles are
added to the draft. (Also note that VSTs are not necessarily sub-graphs of $S_0$, as we only use them as an aid to count the minimum number of chords in the draft.) Then, assume a third m-cycle $c_3: c_d = c_j = c_g = c$ 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 $S_0$ (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-cycle connects two separate VSTs, then this m-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-cycles $c_1: a_b = d = a, c_2: c_b = d = c$ and $c_3: f = a = e = c$ 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 $S_0$ 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 $c_1$ is added, the VST formed by $c_1$ and $c_2$ is actually bridged twice by $a = e = c$ and $a = f = c$. So, $c_1$ introduces two new chords instead of one. In general, when any time an m-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-cycles constructed from the expansion operation of M-CYCLE does not exceed the number of chords in $(V, E)$.

We then consider the refinement operation of M-CYCLE. Assume that three links $a = b, c = d$, and $e = f$ in Fig. 6c are covered by the same set of m-cycles and thus the faults are indistinguishable. Let $c_2$ be the first m-cycle that introduces the three links to the draft, and $c_2$ be an m-cycle constructed later. Both $c_2$ and $c_3$ are obtained from the expansion operation. When $c_3$ is constructed, the draft only contains $c_2$ and all the m-cycles constructed before $c_3$ (i.e. $c_3$ is not included yet). Then, we temporarily remove any one of the three links and construct an m-cycle in the current draft based on one of the two remaining links. If such an m-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-cycle is obtained, e.g., $e = f$ is temporarily removed and an m-cycle based on $a = b$ can be obtained passing through some nodes in the original VST, then $c_2$ must have bridged the original VST twice and thus should have introduced two new chords. However, we have added only one m-cycle $c_3$ to $b$. So, by following Step 3c in Fig. 5 and add an extra m-cycle to distinguish the two faults, the number of m-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-cycle to distinguish them by temporarily removing $e = f$ and constructing an m-cycle based on $c = d$. This is possible only if there is an m-cycle $c_2$ 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 $c_3$ is added, $c_2$ must cover either both or none of them. In both cases, $c_2$ must have bridged the VST twice as shown in Fig. 6c (note that $c_2$ is now included in the VST). So again two new chords are introduced by $c_2$, but only one m-cycle $c_3$ has been constructed. If we add an extra m-cycle to distinguish the two faults, the number of m-cycles does not exceed the number of chords in $(V, E)$.

Combining our proofs above for the three possible draft scenarios in Fig. 6, the number of m-cycles generated by M-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-CYCLE may further remove some redundant m-cycles.

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

**Proof**: From the expansion of M-CYCLE, the length of m-cycles constructed based on each link is minimized. In Step 3c of the refinement, if we need to add an extra m-cycle to $b$, 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-cycles in M-CYCLE is not more than that of m-cycles in a spanning tree-based approach, and the length of each m-cycle is minimized, obviously $L_c$ in M-CYCLE is also smaller.

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

**Proof**: Let $C_M$ and $C_S$ be the solutions of M-CYCLE and a spanning tree-based algorithm, respectively. Because the length of each m-cycle in $C_M$ is minimized, we can enlarge some m-cycles in $C_M$ and turn $C_M$ into $C_S$. In Fig. 7, $c_1: b = a = e = b, c_2: c = b = e = f = c$ and $c_3: d = e = f = g = d$ are three m-cycles in $C_M$, $c_1$ is also an m-cycle in $C_S$ because it covers only one chord. If we replace the chord $b = c$ in $c_2$ by path $b = a = e$, and the chord $c = f$ in $c_1$ 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 $b = a = e = f$ in Fig. 7. Assume that $b = a, e = b$ and $e = c$ are covered by $C_M$ for $d = 3, d = 2$, and $x$ times, respectively. In order to transform $C_M$ to $C_S$, 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 $C_S$, 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:

$$
\begin{align*}
\{b = a: & \Delta_1 + (x - 1) \\
\{a = e: & \Delta_2 + (x - 1) \\
\{b = e: & 1
\end{align*}
$$

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 $C_M$ and $C_S$. We have

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

Since the values of $\Delta_1, \Delta_2$ and $x$ must be at least one (i.e. each link is covered at least once by $C_M$), we get

$$
W_S \geq \max\{\Delta_1, \Delta_2\} + x.
$$

![Fig. 7. An m-cycle can be enlarged to get an m-cycle.](image)

2 In fact, a trivial difference may exist due to the removal of redundant m-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.
Formula (3) indicates that $W_5$ is not smaller than the number of times that $b-a$, $a-e$ and $b-e$ are covered by $C_M$. Note that $b-a-e-b$ is an $m$-cycle in $C_5$. In fact, we can extend this analysis to any $c_i \in C_6$ (such as $c-b-a-e-f-c$). Since $C_6$ covers every link in the topology, from (3) we have $W_5 \leq W_5$. #

**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 $m$-cycle is constructed based on the other link to distinguish the two faults. If this additional $m$-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 $m$-cycles $C_1-\ldots-C_9$. 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].

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 ($m$-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