Spanning-Tree Based Monitoring-Cycle Construction for Fault Detection and Localization in Mesh AONs

Hongqing Zeng and Changcheng Huang
Dept. of Systems and Comp. Eng., Carleton Univ.
1125 Colonel By Dr., Ottawa, ON, Canada K1S 5B6

Alex Vukovic
Communications Research Centre Canada
3701 Carling Ave., Ottawa, ON, Canada K2H 8S2

Abstract–**We previously showed the feasibility of a fault detection scheme for all-optical networks (AONs) based on decomposing networks into monitoring-cycles (m-cycles) [8]. In this paper, m-cycle construction for fault detection is formulated as a cycle cover problem with certain constraints. A heuristic spanning-tree based cycle construction algorithm is proposed and applied to four typical networks: NSFNET, ARPA2, SmallNet, and Bellcore. Three metrics: the grade of fault localization, wavelength overhead, and the number of cycles in a cover, are introduced to evaluate the performance of the algorithm. The results show that it achieves nearly optimal performance.**

Index Terms – **Fault detection and localization, all-optical network, monitoring cycle, cycle cover**

I. INTRODUCTION

Fault detection and localization are essential for providing continuous and reliable services in all-optical networks (AONs) with ever-increasing data rate as well as wavelength number and density in wavelength-division multiplexing (WDM). For AONs, the fault detection and localization can be performed in either physical or IP layer. Most routing protocols in the IP layer, e.g. OSPF or IS-IS, have inherent such functionality [1]. Unfortunately, the long detection time in IP layer (typical at seconds-level) makes it difficult to achieve time-critical recovery. Thus some effective and efficient fault detection mechanisms at optical layer are required. However, existing fault detection and localization mechanisms for conventional networks cannot be applied to AONs directly due to the lack of electrical terminations [2]. Even some detection methods deployed in optical networks with opto-electro-opto (OEO) conversion cannot be transplanted to AONs, e.g. examples in [3].In the physical layer, network faults can be detected by measuring the optical power, analyzing optical spectrum, using pilot tones, or by optical time domain reflectometry (OTDR), etc. [4]. A fault detection scheme was developed by assigning monitors to the sinks of each optical multiplex section and optical transmission section [5]. The scheme proposed in [6] modeled all possible states of a link as a finite state machine (FSM). The FSM for each link keeps tracks of the current state of the link by assigning a monitor to the link. Ideally, all potential faults could be completely detected and located by assigning a monitor to each link (channel). However, it is usually not feasible to implement the one-monitor-per-link scheme in large-scale networks because of the large number of required monitors and the real-time processing of huge amount of redundant alarms.

Other than assigning a monitor per link, some authors placed a monitor to each established lightpath [7]. Some heuristics were proposed to reduce the required number of monitors based on the information of redundant alarms. This scheme was effective when it was proposed, since the number of lightpaths in an AON was relatively small and they did not change frequently once established. However, the number of lightpaths soars so much nowadays with the use of DWDM technology that this scheme will introduce a huge cost due to the large number of monitors required. Furthermore, most AONs currently support dynamically lightpath provisioning so that the monitor placement has to be dynamically re-calculated and re-located once some lightpaths are changed.

In this paper, we propose a novel approach at physical layer for fault detection and localization in AONs through decomposing the given network into a set of cycles, which form a cycle cover for the network. A spanning-tree based cycle finding algorithm is developed and applied to four typical example networks: NSFNET, ARPA2, SmallNet, and Bellcore. The performance of the proposed approach is evaluated in terms of the grade of fault localization, costs, and impacts on wavelength utilizations.

This paper is organized into the following sections. Section II introduces the concept of monitoring cycles and formulates the problem of constructing monitoring cycles to the cycle cover problem. Section III proposes a heuristic spanning-tree based cycle finding algorithm. In section IV, the proposed cycle finding algorithm is applied to four typical example networks: NSFNET, ARPA2, SmallNet and Bellcore. The performances of the proposed algorithms are also evaluated. Finally, some conclusions are outlined in Section V.

II. MONITORING CYCLES AND CYCLE CONSTRUCTION FORMULATION

We previously proposed a fault detection and performance-monitoring scheme based on decomposing an AON into a set of cycles [8]. All nodes and links in the network appear in at least one of these cycles, which form a cycle cover of the network. A network monitor is assigned to one node in each cycle and a loopback supervisory channel is set up in this cycle. A cycle with monitor and supervisory channel is defined as "**monitoring cycle (*m*-cycle)**". Depending on the type of monitors in *m*-cycles (e.g. optical power meters, optical spectrum analyzers, transceivers, etc.), various performance parameters of AONs can be measured, such as optical power, channel wavelength, optical signal-to-noise ratio, and bit error ratio. Flexible index thresholds can be set to determine whether a network fault occurs.

A meshed AON can be modeled as a finite undirected graph *G(V,E)*, where *V* is the set of vertices (nodes) and *E* is the set of edges (links). We assume that such a graph is connected and it contains neither loops nor multiple edges. Furthermore, a bridge link[†] is a single-failure point for the network, thus is usually avoided during the network topology design. Thus *G(V,E)* is assumed to be bridgeless.

A cycle (denoted as *c*) of the graph *G* is a sub-graph of *G* that is connected and regular of degree two. It is often identified with its edge-set. A cycle cover (denoted as *C*) of a graph is a set of cycles in which each vertex and edge of the graph appears at least in one of these cycles. According to the *m*-cycle definition, the set of *m*-cycles is a cycle cover for a given graph. Let $C = \{c_1, c_2, \cdots, c_M\}$ be such a set of *m*-cycles. For an edge $e \in E$, let $C(e)$ denote the number of cycles in *C* that contain *e*, i.e. $C(e) = |\{i : e \in c_i\}|$ [‡]. When $C(e) = t$, we say that the cover time of edge *e* is *t* in *C*. The length of a cycle is the number of edges it contains, denoted by $len(c_i) = |c_i|$. The length of *C*, denoted as $len(C)$, is as summary of all cycles' lengths in *C*. Obviously we have,

$$len(C) = \sum_{i=1}^{M} |c_i| = \sum_{j=1}^{L} C(e_j) \qquad (1)$$

In building an *m*-cycle set (a cycle cover) for a graph, we have to take the following considerations into account,

1) The grade of fault localization: A network fault triggers alarms in the *m*-cycles in which it appears, but not others. Reversely, if alarms are received in some *m*-cycles but no others, it implies that the potential faulty links are the common links of these *m*-cycles. Generally, a binary indication bit m_j can be defined for *m*-cycle c_j, to indicate whether or not a fault occurs and thus an alarm appears in it,

$$m_j = \begin{cases} 1 & \text{an alarm appears in } c_j \\ 0 & \text{no alarm appears in } c_j \end{cases} ; \; j = 1,2,\cdots M \qquad (2)$$

The sequence of such bits for a link forms an alarm code (M bits in total). Alarms are sent to a centralized network management unit (NMU) and alarm codes are generated in real time. Furthermore, for any link $e_i \in E \; (i = 1,2,\cdots,L)$ and m-cycle c_j, a binary associative bit a_{ij} is defined as,

$$a_{ij} = \begin{cases} 1 & e_i \text{ is covered by } c_j \\ 0 & e_i \text{ is not covered by } c_j \end{cases} \qquad (3)$$

where $i = 1,2,\cdots,L$ and $j = 1,2,\cdots M$. The sequence of associative bits of a link corresponding to all the *m*-cycles forms the associative code (M bits in total). By matching the real-time alarm codes with associative codes of all links, any fault can be located in the network. To quantitatively measure the grade of localization, we introduce the concept of ***Localization Degree*** (denote as *I*) which is defined as the average size of candidate sets over all possible alarm codes,

$$I = \frac{\sum_{candidate\ sets} size\ of\ the\ candidate\ set}{number\ of\ non\text{-}empty\ candidate\ sets} \qquad (4)$$

In the ideal case, every candidate set has only a single element and $I_{ideal} = 1$ (defined as complete localization). In building *m*-cycles for fault detection, we want to minimize the localization degree, i.e. *MIN I*.

2) Wavelength overhead: In each link, some wavelength channels are reserved for *m*-cycles. These channels cannot be used for carrying user traffic and therefore become an overhead. The number of reserved wavelengths (denoted as Λ) within a link is equal to the cover times of that link in the cycle cover. Given a graph with *N* vertices, *L* edges and *M m*-cycles, the average number of reserved wavelengths (Λ_{ave}) for all edges is equal to the average cover time and,

$$\Lambda_{ave} = \sum_{i=1}^{L} C(e_i) \Big/ L = len(C)/L \qquad (5)$$

To quantitatively analyze the relative overhead due to *m*-cycles, we define the average wavelength overhead per link brought to the network by *m*-cycles as $WOH_{ave} = \Lambda_{ave} / F$, where F is the number of total available wavelengths per link. To minimize the wavelength overhead, we have to minimize Λ_{ave}, which is equivalently to minimize the cycle cover length. Consequently, finding *m*-cycles can also be formulated to the least cost cycle cover problem for un-weighted graphs.

3) The number of cycles in the cover: Since a monitor and a dedicated supervisory channel are assigned for each *m*-cycle, this is the number of required monitors and reserved wavelengths. Thus it is a measurement of the cost for *m*-cycle based fault detection and localization. To minimize this cost, we have to minimize the number of cycles, i.e. $MIN\,|C|$.

It has been proved that cycle covers exist for each bridgeless, connected, undirected graph and could be obtained in running time $o(n^2)$ [9]. A polynomial-time algorithm was also reported for constructing cycle covers [10]. Such works are focused on the least cost cycle cover problem. They do not consider the localization degree and cycle numbers. We previously developed two *m*-cycle construction algorithms: heuristic depth-first searching and shortest path Eulerian matching algorithm, with a balance of all three considerations

[†] An edge is a bridge of a graph if the graph becomes from a connected graph to be a disconnected one after deleting it.
[‡] $|\cdot|$ represents the set cardinality, i.e. the number of elements in a finite set.

[8]. In this paper, we propose a heuristic spanning-tree based *m*-cycle construction algorithm for the same purpose, while improving the performance in terms of localization degree.

III. Heuristic spanning-tree based cycle construction

A. Preliminary

For a connected, bridgeless, simple graph *G(V,E)*, there must exist a spanning-tree *T*. $\forall link\ e \notin T$ (*e* is called a "chord"), if the two endpoints of *e* are n_1 and n_2, then $n_1, n_2 \in T$ and there must exist a path $p \in T$ connecting n_1, n_2. Thus link *e* and path *p* form a cycle. We say this cycle is generated by the chord *e*. Each chord generates such a unique cycle. We also have the following cycle cover existence lemma [9],

Lemma: There exists at least one cycle cover for a bridgeless graph *G(V,E)*.

Cycles $c_1, c_2, \cdots, c_M$ are called independent if their associative vectors are linearly independent[§]. H. Walther has proved the following theorem in [11],

Walther's Theorem: Let *G* be a connected graph with *L* edges and *N* vertices. Then there exist *L-N+1*, but no more independent elementary cycles[**].

Based on the above lemma and theorem, we claim the following theorem,

Theorem: For a connected, bridgeless, simple graph *G* with a given spanning-tree, cycles generated by all chords construct a cycle cover for *G*.

Proof: Let *G* has *L* edges, *N* vertices, and the spanning-tree is *T*. Then all edges can be partitioned into two sets: *N-1* edges in *T* and *L-N+1* edges not in *T*.

(1) Each edge not in T is a chord. It generates a cycle and is covered by this cycle. There are L-N+1 such cycles.

(2) Assume $\exists e^* \in T$ and e^* is not covered by any cycles generated by those chords. Because of the lemma, there must exist another cycle c_0 in which e^* appears. The associative vector of c_0 is $\mathbf{a_0} = (a_0^1, a_0^2, \cdots, a_0^*, \cdots, a_0^L)$, where $a_0^* = 1$ and corresponding to the position of edge e^*. For all other cycles, the components at this position of the associative vectors are zero, because they do not cover edge e^*. Thus cycle c_0 is independent from all other *L-N+1* cycles. Consequently, by adding c_0, graph *G* has *L-N*+2 independent cycles. This is a contradiction to Walther's theorem. #

[§] A group of vectors $\mathbf{a_1}, \mathbf{a_2}, \cdots, \mathbf{a_n}$ are linearly independent if and only if $\sum_{i=1}^{i=n} k_i \mathbf{a_i} = 0$ holds when $k_1 = k_2 = \cdots = k_n = 0$.

[**] A cycle is called elementary cycle if no vertex is encountered more than once when traversing it. Each cycle can be partitioned into elementary cycles. In this paper, a cycle is an elementary cycle.

In the proof, please note that a cycle cover of graph *G* is uniquely determined by the given spanning-tree.

B. Heuristic spanning-tree (HST) based cycle finding

Breadth-first and depth-first spanning-trees (BFST and DFST) are well known and have been in common use for a long time. Numerous algorithms to generate such spanning-trees have been intensively studied [12]. Fig. 1 gives three spanning-trees for an example graph and *m*-cycles generated by corresponding chords. Numbers of cycles in the covers generated by various spanning-trees are the same for the graph. By enumerating the faulty candidates for all possible alarm codes, we find that localization degrees for them are also the same (*I=1*). However, the figure shows that the average cover time (i.e. the average wavelength overhead) per link is smaller for the cover generated by BFST than by DFST. Furthermore, the average cover time might be decreased by including nodes with large degrees in the tree (comparing b and c). This observation leads us to choose BFST and apply a heuristic rule of putting the large-degree nodes into the spanning-tree as early as possible while generating the spanning-tree for constructing a cycle cover. A heuristic spanning-tree (HST) based cycle construction algorithm is then given below,

1. Initial: for a graph G, label the degrees of all nodes; set the spanning-tree *T=null*; select the node with the maximum degree as the root. Add all links to *T* that are incident to the root.
2. For each node $n_i \in T$, update its degree label with the number of links that are incident to n_i and connect n_i with nodes not in T.
3. Select the node with the maximum degree label in T, add all links to T that are incident to the selected node and connect it with nodes not in T.
4. Repeat step 2-3 until all node-degree labels are zero. Now T is a spanning-tree of G.
5. Given T, construct the cycles for all chords, they form a cycle cover and are the required m-cycles.

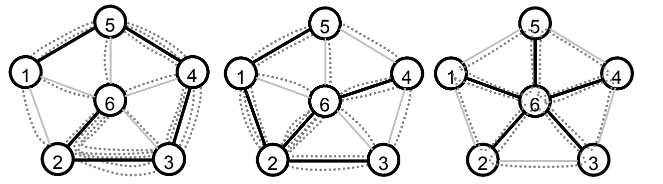


Fig. 1. Spanning-tree and cycle cover

(a) DFST: max cover time = 5/link, average cover time = 2.3/link

(b) BFST (rooted from random nodes): max cover time = 5/link, average cover time = 1.9/link

(c) BFST (rooted from the node with maximum degree): max cover time = 2/link, average cover time = 1.5/link

IV. Examples and evaluations

The HST based cycle construction algorithm is applied to four typical example networks (NSFNET, ARPA2, SmallNet, and Bellcore). The network topologies, spanning-trees, and *m*-cycles are shown in Fig. 2. The performances of the algorithm are evaluated in terms of those metrics described in Section II, including localization degree, cost, and wavelength overhead.

Table 1 enumerates all possible alarm codes and faulty candidate sets for NSFNET as an instance. Similar results can be obtained for other example networks (ARPA2, SmallNet, and Bellcore) in the way. Table 2 summarizes these results and compares with the HDFS and SPEM algorithms reported in [8]. The comparison shows that the HST algorithm has much better performance than HDFS and SPEM algorithms in terms of the fault localization degree. Further analyses indicate that the HST algorithm performs even better in terms of localization degree for graphs with larger average node degrees. More specifically, for graphs with average node degree larger than 3.0, the localization degrees of HST algorithm are very close to the ideal case. This observation implies that such fault detection and localization approaches are suitable for complex networks (with large average node degree), and thus are scalable.

TABLE 1. Fault localization results: NSFNET - HST

alarm codes								**faulty**
c1	c2	c3	c4	c5	c6	c7	c8	**candidates**
0	0	0	0	0	0	0	0	null
							1	10-14
						1		9-14
						1	1	12-14
					1			6-11, 9-11
				1				8-9
				1	1	1		9-13
			1					5-7, 7-8
			1	1				2-8
		1						4-10
		1					1	10-13
		1		1	1			6-12
		1		1	1	1	1	12-13
	1							1-4
	1		1	1				3-6
	1	1						4-5
	1	1	1					5-6
1								1-2
1			1	1				2-3
1	1							1-3
others								N/A

The cost of the proposed scheme is measured by the number of required monitors and reserved wavelengths for *m*-cycles. The cost of wavelength is evaluated by Λ_{ave}, $\Lambda_{\max}$, and WOH_{ave}, as described in Section II. Results in Table 2 show that numbers of both maximum and average reserved wavelengths for *m*-cycles obtained by HST algorithm are larger than HDFS and SPEM. This is the payment for the benefit in localization degree. Nevertheless, with the DWDM technology, the number of wavelengths in a single link tends to become larger. For example, it was reported even in 2001 that 432 wavelengths could be multiplexed into a single fibre [13]. In current commercial DWDM systems, it is easy to boost the number of available wavelengths in a fibre to 192 or above [14]. Even for a small number of available wavelengths per link, e.g. F=64, the wavelength overheads for HST algorithm are small (around 3%, see Table 2). Such overheads have trivial impact on network utilization, if it is not negligible.

The cost of monitors is weighted by the number of monitors for *m*-cycles, i.e. the number of *m*-cycles (denoted as *M*). For comparing with the monitor-per-link case, a cost gain is calculated as $G = (L - M) / L$ where *L* is the number of links. The cost gains of HST are compared in Table 2 with HDFS and SPEM for example networks. Because the monitor-per-link approach always achieves complete localization, for a fair comparison, we add some extra monitors for those links that cannot be fully localized under the HST algorithm to achieve complete localization. For example, a straightforward method would be the following one. If there are $K \geq 2$ links in a faulty candidate set, we assign $K-1$ extra monitors to $K-1$ of those K links. More efficient methods might be applied for achieving complete localization, e.g. using extra *m*-cycles. Therefore, HST algorithm still has good cost gains, although the *M* values of HST are larger than HDFS and SPEM. Denote the number of extra monitors as M', the complete localization can be achieved and the cost gain calculation is revised as,

$$G' = (L - (M + M'))/L \tag{6}$$

Revised cost gains obtained from the four example networks are also compared in Table 2. Again, the average node degree affects the cost gain. For graphs whose average node degree is 3.0 or above, the cost gain for HST is 40–52%. Even for the worst case, ARPA2, the cost gain is still no less than 20%. Such results show that under a fair comparison, the cost gains of HST are better than HDFS and SPEM.

The fault detection scheme in [7], as described in Section I, placed a monitor per path. In a *N*-node network, typically each node has to communicate with all the others, thus the number of potential paths is $N(N-1)$. Even with the 50% savings of monitors (in maximum) by applying the proposed heuristic optimization algorithm, the number of required monitors is still $O(N^2)$. Clearly, the *m*-cycle based approach achieves significant cost gains in all examples compared to either the monitor-per-link or monitor-per-path case.

V. Conclusion

In mesh AONs, network faults can be detected and located by decomposing them into monitoring-cycles (*m*-cycles). We formulated the *m*-cycle construction as the cycle cover problem with certain constraints. A heuristic spanning-tree (HST) based *m*-cycle construction algorithm is developed and evaluated in terms of the localization degree, wavelength overhead, and cost gain. The proposed HST algorithm is applied to four typical networks (NSFNET, ARPA2, SmallNet, and Bellcore) and compared to the previously reported algorithms, HDFS and SPEM. The comparison results show that the performance of localization degree for HST algorithm is better than HDFS and SPEM. Analyses indicate that the average node degree of a network plays an important role in the performance of *m*-cycle based fault detection and localization approaches. The fact that *m*-cycle based approaches can achieve better performance in networks with larger average node degree implies that such approaches are suitable for complex networks and thus scalable.

The HST algorithm introduces more monitors than HDFS and SPEM, however, in a fair comparison of achieving complete localization, it has better cost gains than HDFS and SPEM. Additionally, all the three *m*-cycle construction algorithms have good cost gains over either monitor-per-link or monitor-per-path case. Finally, the wavelength overheads due to *m*-cycles are negligible in all approaches. Therefore, the HST algorithm is effective and cost-efficient.

REFERENCES

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

[2] C. Mas and P. Thiran, "A review on fault location methods and their applications in optical networks," *Optical Network Magazine*, vol. 2, No. 4, July/Aug. 2001

[3] 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

[4] M. Mèdard, D. Marquis, and S. R. Chinn, "Attack detection methods for all-optical networks," *Network and Distributed System Security Symposium*, 1998

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

[6] 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

[7] 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

[8] H. Zeng, C. Huang, A. Vukovic, and M. Savoie, "Fault Detection and Path Performance Monitoring in Meshed All-Optical Networks," *IEEE Globecom 2004* (accepted)

[9] A. Itai and M. Rodeh, "Covering a graph by circuits," *Automata, languages and programming,* Lecture Notes in Computer Sci. 62, pp.289-299, Berlin: Springer-Verlag, 1978

[10] G. Fan, "Covering graphs by cycles," *SIAM journal on Discrete Mathematics*, vol. 5, No. 4, pp. 491-496, Nov. 1992

[11] H. Walther, "Chapter 1: Flows and tensions on networks," *Ten applications of graph theory*, Boston: D. Reidel Pub. Co. 1984

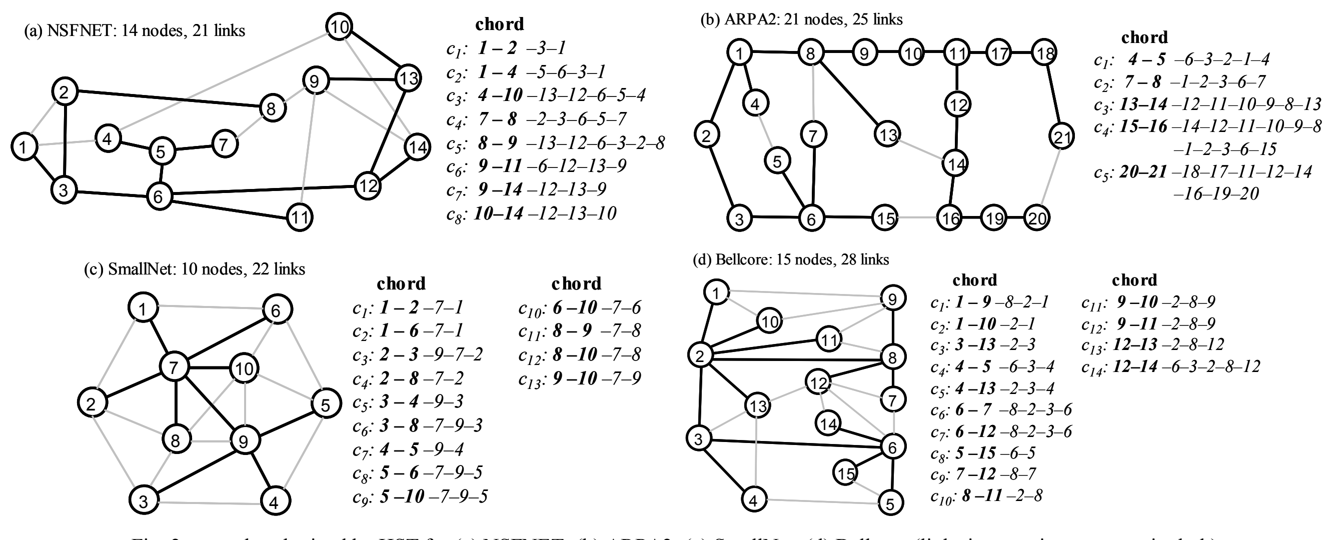
[12] R. J. Wilson and J. J. Watkins, "Chapter 10: Trees," *Graphs: an introductory approach*, New York: Wiley, 1990

[13] S. V. Kartalopoulos, *Fault Detectability in DWDM – Toward Higher Signal Quality & System Reliability*, Piscataway: IEEE Press, 2001

[14] R. Elliott, "Dark fibre pricing analysis Europe 1998-2002," [Online document], Nov. 2002, available at http://www.band-x.com/information/Dark_Fibre_Report-prices_98-02webversion.pdf

TABLE 2. Comparison of localization degree, wavelength overhead, and cost gains

Network example	Ave. node degree	Algorithm	localization degree	max candidate set size	Λ_{max}	Λ_{ave}	WOH ave (%)	M	G (%)	M'	M+M'	G' (%)
NSFNET	3.00	**HST**	**1.105**	**2**	**5**	**1.90**	**2.97**	**8**	**61.9**	**2**	**10**	**52.4**
		HDFS	1.500	3	3	1.57	2.45	6	71.4	7	13	38.1
		SPEM	3.000	7	2	1.24	1.94	4	80.9	15	19	9.5
ARPA2	2.38	**HST**	**2.500**	**6**	**3**	**1.60**	**2.50**	**5**	**80.0**	**15**	**20**	**20.0**
		HDFS	3.130	6	3	1.36	2.13	4	84.0	16	20	20.0
		SPEM	5.000	8	2	1.20	1.88	4	84.0	18	22	12.0
SmallNet	4.40	**HST**	**1.000**	**1**	**6**	**1.95**	**3.05**	**13**	**40.9**	**0**	**13**	**40.9**
		HDFS	1.470	3	3	1.55	2.42	8	63.6	7	15	31.8
		SPEM	3.670	6	2	1.18	1.84	4	81.8	16	20	9.1
Bellcore	3.73	**HST**	**1.077**	**2**	**8**	**1.96**	**3.06**	**14**	**50.0**	**2**	**16**	**42.9**
		HDFS	2.150	6	3	1.43	2.23	6	78.6	15	21	25.0
		SPEM	4.670	8	2	1.14	1.78	5	82.1	21	26	7.1



(a) NSFNET: 14 nodes, 21 links
(b) ARPA2: 21 nodes, 25 links
(c) SmallNet: 10 nodes, 22 links
(d) Bellcore: 15 nodes, 28 links

(a) **chord**

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*

(b) **chord**

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) **chord**

c_1: ***1 – 2*** *–7–1*
c_2: ***1 – 6*** *–7–1*
c_3: ***2 – 3*** *–9–7–2*
c_4: ***2 – 8*** *–7–2*
c_5: ***3 – 4*** *–9–3*
c_6: ***3 – 8*** *–7–9–3*
c_7: ***4 – 5*** *–9–4*
c_8: ***5 – 6*** *–7–9–5*
c_9: ***5 –10*** *–7–9–5*

chord

c_{10}: ***6 –10*** *–7–6*
c_{11}: ***8 – 9*** *–7–8*
c_{12}: ***8 –10*** *–7–8*
c_{13}: ***9 –10*** *–7–9*

(d) **chord**

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*

chord

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*

Fig. 2. *m*-cycles obtained by HST for (a) NSFNET; (b) ARPA2; (c) SmallNet; (d) Bellcore (links in spanning-trees are in dark)