A Fault-Tolerant Multipoint Cycle Routing Algorithm (MCRA)

David Lastine, Suresh Sankaran, and Arun K. Somani

Department of Electrical and Computer Engineering
Iowa State University, Ames, Iowa 50011
{dlastine,sankaran,arun}@iastate.edu

Abstract. In this paper we propose a new efficient fault tolerant multipoint routing algorithm for optical networks. The routing for a multipoint request is accomplished by finding a bidirectional cycle simple or nonsimple including all nodes that are participating in the multipoint session. Each link can be used only once. Use of a cycle ensures that a single link (or node in case of simple cycle) failure does not interrupt the session except the failed node if it was part of the multipoint session. Determining the smallest cycle with a given set of Multi-point (MP) nodes is a NP-Complete problem. Therefore, we explore heuristic algorithms to determine an appropriate cycle to route multipoint connections. We allow non-simple cycles to route requests as they use fewer resources than simple cycles in some cases. We also provide an ILP formulation for routing multipoint request and compare its results with the output of our best heuristic algorithm. On Arpanet for over 80% of the time, our best heuristic is able to find a cycle that is within 1.2 times that of the optimal.

Keywords: Cycle routing algorithm, Multipoint Communication, Multicasting, Fault Tolerant Routing.

1 Multipoint Connection Problem

Multipoint (MP) to Multipoint communication is the transmission of information from all source nodes to all destination nodes. The sources and destinations form a subset of nodes in the network. We shall refer to them as multipoint (MP) nodes. Currently there exist many applications such as multimedia collaborations, video conferencing and shared workspaces which require MP to MP services. The increasing number of users of these applications necessitates a fiber-optic based optical network that can satisfy the bandwidth needs per request. However, failures in fiber-optic lines happen as frequently as every couple of days. This motivates us to explore fault aware MP routing.

The challenge in providing efficient protection to multipoint to multipoint communication is to recognize that it provides more opportunities for resource sharing than distinct entities of multicast communication. The relationship between multipoint and multicast is analogous to the situation between multicast and unicast. Consider implementing a fault tolerant multicast connection as a set of fault tolerant unicast connections. While this would be feasible, connections for different destinations would independently reserve capacity on every link they use, even when a link is common between them. Multicast specific routing strategies avoid this redundant reservation. All multipoint routing algorithms try to avoid redundancies that would be incurred by repeatedly using multiple unicast strategies.

1.1 Multicast Protection Scheme

MP to MP connection is a generalization of multicast. It is equivalent of a multiple multicast request. Significant work has been done for protecting and restoring multicast connections in light tree, little work has been done on efficient protection of multipoint request. In [1] multicast tree protection is classified into the following categories: path-based, segment-based, tree-based, p-cycle based, and ring-based protection.

Most of the existing approaches route a multicast light tree that is protected by another tree or a cycle of some type. The idea of using a cycle to both route and protect a multicast request has received little attention in the literature. The idea is considered in [2]. The paper considers routing dynamic multicast connection using a cycle in NSF network. If their algorithm fails to find a cycle, they classify it as a blocked connection. We note that the lowest blocking probability reported for the network is 0.1, which is rather high.

We in this paper, solve the general problem of MP to MP connection using a cycle-based approach. Since this problem has received little attention in literature, for comparison we review the multicast protection schemes below.

A straightforward approach to protect a multicast tree is to compute a link-disjoint or arc-disjoint backup tree providing 1+1 dedicated protection [3] but this protection scheme consumes excessive resources. Although it may not be always possible to find an arc-disjoint backup tree once a primary tree has been discovered, it may be possible to protect each segment in the primary tree by finding a segment-disjoint path. Two protection schemes called Optimal Path Pair-based Shared Disjoint Segment(OPP-SDS-H) and Optimal Path Pair-based Shared Disjoint Path(OPP-SDP) have been proposed in [4] and the results suggest OPP-SDP outperforming SDS protection scheme. In dynamic multicast session, Link Based-Shared protection algorithm (LB-SPDM) is shown to perform better in terms of blocking probability than OPP-SDP [5]. Since OPP-SDP is a failure-independent method which looks for only one protection path for each working path for any link failure on that path, it makes the protection path unable to share wavelengths with working path unlike in failure dependent LB-SPDM protection.

The use of p-cycles and multicast trees to provide protection for dynamic multicast connections is considered in [6] and [7]. Several ways to place p-Cycles are examined in [8] and their resource usage and computation speed are compared. In [9] the blocking of multicast trees is examined for three different p-cycle placement scenarios. They consider placing the p-cycles without knowledge of the traffic, placing p-cycles dynamically as traffic arrives, and a hybrid scenario. Researchers in [10] analyzed a p-cycle based light-tree protection (ESHT) for combined node and link failure recovery. The capacity efficiency of this heuristic is close to OPP algorithm, while the blocking performance is in between OPP-SDP and OPP-SDS. The advantage of p-cycle based approach is faster restoration speed because p-cycle are pre-cross-connected.

[1] evaluated the performance of node-and-link protecting p-cycle based approach, tree-protecting p-cycle based approach and the OPP based approach. It is found that efficiency-score based heuristic algorithm of node-and-link protecting p-cycle outperforms the other heuristic algorithms. These protection strategies are shown to be efficient when applied for single multicast request but how to extend them efficiently to multipoint request is unclear.

1.2 Multipoint Protection Scheme

In this paper, we propose to route the multipoint request in a bidirectional cycle with UPSR like protection. We explore fault tolerant MP routing by establishing cycles to support MP requests. In order to provide a single link fault tolerance to a MP connection, there needs to exist a path from each node to all other nodes participating in the MP communication that does not use that link. A bidirectional simple or non-simple cycle satisfies this property if no link is reused. Since we only allow reusage of nodes and not links for a request, a single link failure does not disconnect the MP connection in a bidirectional cycle. This is done by having the transmitting node send its' traffic in both directions as shown in Figure 1(a) where node s_1 is transmitting. The transmitting node is responsible for ensuring the signal is dropped, once it has completed the loop. Receiving nodes listen to the incoming signal from both directions. In fault free operation a receiver will receive the same information twice and can either ignore the duplicate information or can use it for verification. When a fault occurs, as shown in Figure 1(b), then all nodes still receive the transmitted information, but only once. In this case, verification is not possible, but information is available at the destination.

By this kind of protection scheme, we eliminate the cost of a dedicated disjoint backup path and reduce latency since the traffic is transmitted in both the directions. When we compare with other protection mechanisms like dual multipoint tree and p-cycle, the multipoint cycle based protection scheme has the advantages that no loss of traffic occurs and the fault location need not be identified. This is true irrespective of which node is transmitting. Hence MP to MP protected connection is established.

1.3 Multipoint Connection Cycle

To route a multipoint connection, the network receives request to create a cycle involving a set of nodes participating in multipoint session. The cycle may

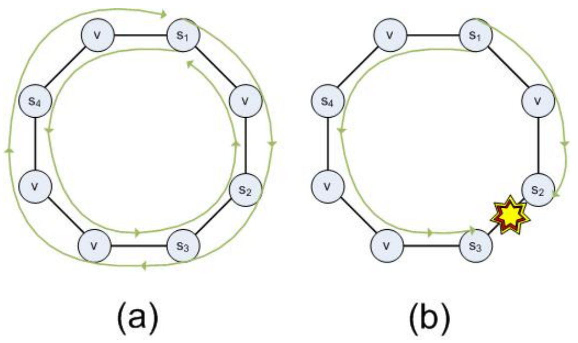

Fig. 1. Media Access Control Protocol

contain additional nodes as required due to the network topology, while containing all the nodes that are part of the request. If such a cycle cannot be established, then the request is rejected and the connection is blocked.

Finding the smallest cycle that includes a specified set of nodes is a hard problem. We formally define a decision version of the Cycle Based Routing (CBR) problem as follows.

CBR Problem: Given a graph $G(V, E)$, where V denotes the set of nodes and E denotes the set of edges in the graph, an integer k, $S \subseteq V$ as a subset of nodes, is there a simple or non-simple cycle graph $P(V(P), E(P)) \subseteq G$, where $V(P)$ is the set of nodes in P and $E(P)$ is the set of edges in P such that $S \subseteq V(P)$ and $|E(P)| \leq k$?

This problem is hard. Therefore, we develop heuristic algorithm to determine an appropriate cycle to route a multipoint connection. The evaluation of our heuristic is performed by routing many randomly generated requests on randomly generated networks. The number of nodes considered varies between twenty and sixty five nodes. We assume that networks consisting of more than sixty five nodes will have their routing performed in a hierarchical fashion. It is generally accepted that blocking is kept low by using a minimum amount of networking resources per request. To explore how efficiently our heuristics utilize resources we analyze different variations of the algorithm and compare the best performing heuristic them with the optimal results of an ILP.

2 Network Graph Model

The network is modeled using a connected graph $G(V, E)$. V is the set of vertices representing nodes in the network. E is the set of edges $(u, v) \in E$ where $u, v \in V$. An edge (u, v) represents the physical link that allows communication between nodes u and v. Links are assumed to have capacity which can take the values of 1 or 0 depending on availability or non-availability.

3 Multipoint Cycle Routing Algorithm (MCRA)

One application of CBR problem is in multipoint communication. $S \subseteq V$ is a set of nodes, which participate in a multipoint session. In this paper, we develop a multipoint cycle routing algorithm (MCRA). For this purpose, we associate costs with nodes. This cost is modeled as a binary variable. For non-MP nodes the cost is always one, for MP nodes we have considered two scenarios; in one scenario the cost is set to one and in the other the cost is set to zero. Since links are used only once we assume that links have no costs.

Since finding an optimal cycle is hard, we use simple heuristic algorithm to find a cycle. Our algorithm to find a fault tolerant cycle to route the MP request consists of multiple phases.

3.1 First Phase Algorithm

In the first phase, we consider all pairs of MP-nodes and compute a shortest distance path between each MP-node pair in the graph. We examine all these shortest paths, the one that includes the largest number of MP-nodes is recorded as the Initial $P(T)$. If there are more than one, we select the first one discovered.

One way to compute the shortest paths is to create the shortest path tree for each node $s_i \in S$. The shortest path tree is created by the function $D(s_i)$ (pseudo code for this function is not included as we simply use a variation of Bellman-Ford where we consider nodes not edges to be weighted). The function $M(s_j)$ retrieves a shortest path from destination s_j to source s_i (pseudo code for this function is also not provided as it is simply a tree traversal algorithm). Out of all the shortest paths in $|S|$ trees, the one with many MP-nodes is recorded as Initial $P(T)$ as shown in Figure 2 as Initial $P(T)$.

3.2 Second Phase Algorithm

In the second phase, a new segment passing through a yet to be added MP-node is inserted in between the two multipoint end nodes of Initial $P(T)$. It is done by considering each yet-to-be-covered MP-node and finding the segment to connect between multipoint end nodes in Initial $P(T)$. The segment is computed using the function $Find-Segment$ as described later on. While finding this segment, by assigning the link capacity to zero we do not use any of the links already present in Initial $P(T)$. We choose the best segment, that has many uncovered MP-nodes, to insert into Initial $P(T)$. This closes Initial $P(T)$ to form a cycle.

Figure 2 shows the second phase computation. For each uncovered MP node s_i, we calculate a segment $s_j- > s_i$ and $s_i- > s_k$. All edges except those on $P(T)$ are considered available. The set of segments $s_j - s_i - s_k$ (shown as dotted line) is selected for insertion for which $s_j - s_i - s_k$ has maximum number of uncovered MP nodes. Notice that a MP node on $P(T)$ can also belong to segment in $s_j - s_i - s_k$ but it does not count towards uncovered MP nodes.

MCRA algorithm

```
Input: Graph $G$, Subset $S$
Output: $P(T)$ - Self Protecting Multipoint Communication Cycle
$P(T) = NULL$; $P = NULL$;
//Phase 1 starts
foreach $s_i \in S$ do
    $D(s_i)$; creates shortest path tree rooted at $s_i$
    foreach $s_j \in S; j > i$ do
        $P = M(s_j)$; Finds the path from $s_j$ in the tree
        if # of MP nodes in P > # of MP nodes in P(T) then
            $P(T) = P$;
        end
    end
end
$\forall e \in P(T)$ capacity(e) = unavailable;
//Phase 1 ends - Phase 2 starts
$subpath = NULL$; $Bestpath = NULL$;
$s_j = FirstnodeP(T)$;
$s_k = LastnodeP(T)$;
foreach $s_i \in S$ and $s_i \notin P(T)$ do
    $subpath = Find_Segment(P(T), G, s_i, s_j, s_k)$;
    if # of uncovered MP nodes in subpath > # of MP nodes in Bestpath
    then
        $Bestpath = subpath$; pick new segment
    end
end
if $Bestpath == NULL$ then
    return $NULL$; request blocked
end
Insert $BestpathinP(T)$; P(T) becomes a cycle
//Phase 2 ends - Phase 3 starts
$subpath = NULL$; $BestSubpath = NULL$;
while $S \not\subseteq P(T)$ do
    foreach $s_i \in S - P(T)$ do
        $subpath = Select - Segment(s_i, GraphG)$;
        if # of uncovered MP nodes in subpath > # of MP nodes in
        BestSubpath then
            $BestSubpath = subpath$; pick new segment
        end
    end
    if $BestSubpath == NULL$ then
        return $NULL$; request blocked
    end
    Insert $BestSubpath$ into P(T);
end
Return P(T); //Phase 3 ends
```

Select-Segment

```
Input: $s_i \in S$ and $s_i \notin P(T)$, Graph $G$
Output: $BestSubpath$
$BestSubpath = NULL$;
foreach consecutive $s_j, s_k \in P(T)$, $s_j, s_k \in S$ do
    $subpath = Find - Segment$(P(T), $s_i$, $s_j$, $s_k$, Graph $G$);
    if $s(subpath) > s(BestSubpath)$ then
        $BestSubpath = subpath$;
    end
end
```

Find-Segment

```
Input: $P(T)$, $s_i$, $s_j$, $s_k$, Graph $G$
Output: $Subpath$
$Subpath = Find - Segment - Helper(P(T)$, $s_i$, $s_j$, $s_k$, Graph $G$);
if $Subpath = NULL$ then
    $Subpath = Find - Segment - Helper(P(T)$, $s_i$, $s_k$, $s_j$, Graph $G$);
    switches order of nodes
end
return $subpath$;
```

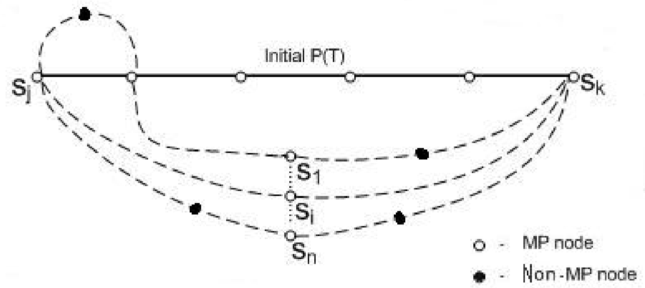

Fig. 2. Example of Phase 2

3.3 Third Phase Algorithm

In the third phase, each of the remaining MP-nodes are inserted into the cycle $P(T)$. New segments are selected after considering all remaining MP-nodes and all possible segments (a path between two consecutive MP-nodes s_j and s_k on P(T)) such that an uncovered node $s_i \in S - V(P(T))$ can be included between nodes s_j and s_k. A node s_i is inserted by replacing the path between nodes s_j and s_k on $P(T)$ by a segment that includes s_i. Each rerouting enables inclusion of one or more uncovered MP-nodes in $P(T)$. This is depicted in Figure 3.

Find-Segment-Helper

```
**Input**: $P(T)$, $s_i$, $s_j$, $s_k$, Graph $G$
**Output**: $Subpath$
Set $s_k.Cost = |V|$;
$D(s_j, G)$; Builds a shortest path tree rooted at $s_j$
$P_1 = M(s_i)$; Find shortest path between $s_i$ and $s_j$
Set $s_k.Cost = 1$;
$\forall e \in P_1 \;\; capacity(e) = 0$;
Set $s_j.Cost = |V|$;
$D(s_k, G)$;
$P_2 = M(s_i)$;
Set $s_j.Cost$ =1;
$\forall e \in P_1 \;\; capacity(e) = 1$;
if $P_1 = NULL \; or \; P_2 = NULL$ then
    return $NULL$;
end
else
    return $P_1 \bigcup P_2$;
end
```

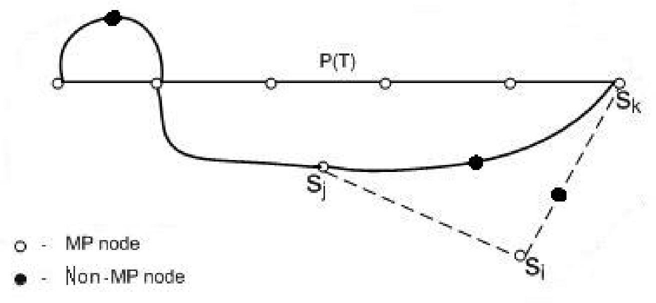

Fig. 3. Example of Phase 3

No link that is used in $P(T)$ is available to compute segment $s_j - s_i - s_k$ (shown as dotted line) except the links on path $s_j - s_k$ in $P(T)$.

The candidate segments for a specific uncovered node are found by calling the function $Select - Segment$. One uncovered MP-node s_i and the current $P(T)$ are input to $Select - Segment$. The best segment to insert MP-node s_i and the location in $P(T)$ between nodes s_j and s_k is returned by $Select - Segment$ function.

$Select - Segment$ Function $Select - Segment$ is responsible for finding the best new segment that includes s_i in $P(T)$. It performs the necessary work to loop over all the consecutive MP-nodes pairs on $P(T)$ starting with one pair of MP-nodes s_j and s_k to include node s_i. It does so by considering every segment of $P(T)$, i.e., a path between two consecutive MP-nodes on $P(T)$, and checks if the path can be replaced by a new segment containing the node s_i. $Select-Segment$ uses a function $Find-Segment$ to find the new segment.

$Find-Segment$ Function $Find-Segment$ is used to find a short, low cost segment passing through node s_i between nodes s_j and s_k. Recall that MP nodes are assigned cost 0 and other nodes are assigned cost 1. we later experiment with other cost models too. The arguments for this function are the current $P(T)$, Graph G, node s_i - a MP-node to be included in $P(T)$, and the two consecutive MP-nodes s_j and s_k on $P(T)$ such that the path between them is to be replaced to insert node s_i.

For each s_i, function $Find-Segment$ is called p times by the function $Select-Segment$, where p is the number of MP-nodes currently in $P(T)$. Then the segment with the least cost and the most MP-nodes is selected to be inserted into $P(T)$. The maximum value of p can be at most $|S|-1$. When finding a segment a currently used link is not allowed to be used again.

3.4 Detailed Example

Figure 4 depicts the three phases of execution of our heuristic algorithm to find a cycle in Arpanet for a multipoint request of consisting of nodes in set $S = \{2,\ 5,\ 13,\ 14,\ 17,\ 19\}$. The top of the figure shows the segment found in the first phase is and it is {2, 4, 11, 13, 17}. This segment includes three of the nodes from S.

This example was produced by our implementation. Recall that there may be more than one shortest path between a pair of nodes in a given graph. Different numbering of nodes and processing steps could lead to many valid executions of the Bellman-Ford algorithm. The outcome will be different based on which shortest path is found by a particular implementation of the Bellman-Ford algorithm. In the example graph there are several shortest paths between nodes 2 and 19. The shortest path algorithm may find the path $\{2,1,3,7,10,19\}$. This path has only two nodes from S while the Initial $P(T)$ selected by our algorithm has three nodes from S. There exists a shortest path $\{2,4,11,13,17,19\}$ between nodes 2 and 19. This path has four nodes from S. If this had been the path found by our implementation then it would have become the Initial $P(T)$. However, this did not happen to be the path found by our shortest path algorithm.

The middle of the figure shows the second phase of the execution where a segment {2,5,6,7,10,19} is added to form a cycle. This segment includes two previously uncovered MP-nodes of S i.e. 5 and 19.

The bottom of the figure shows the third phase of execution where connection between nodes 17 and 19 has been replaced by the non-simple path {17, 19, 16, 15, 14, 20, 19}. The segment is non-simple since node 19 is both one

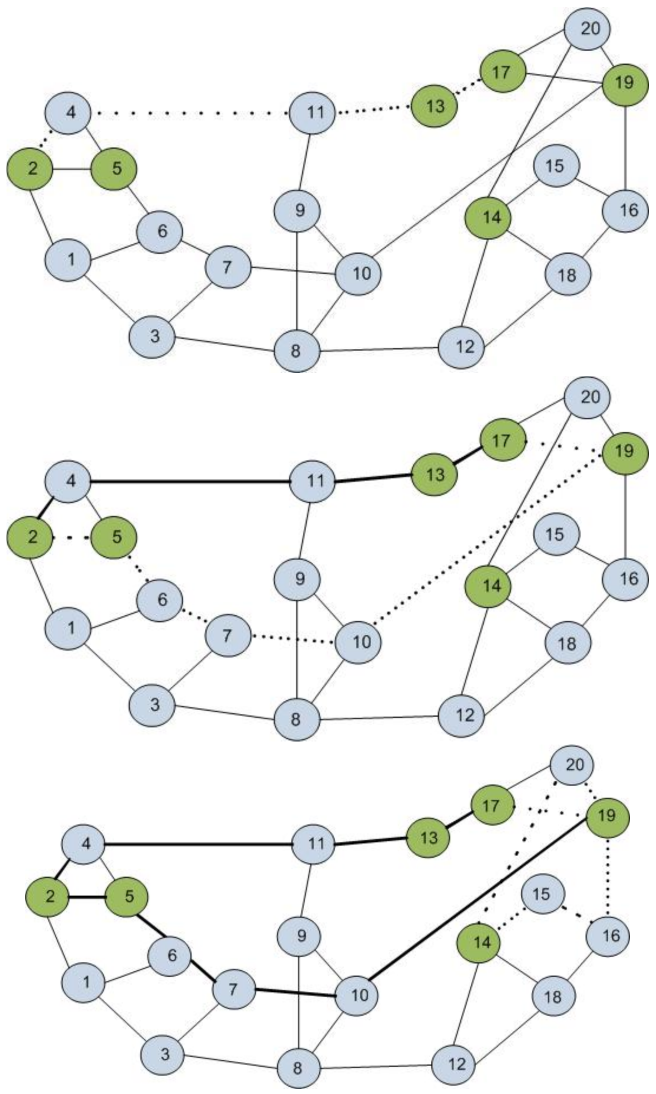

Fig. 4. Step 1, 2 and 3 for Cycle Routing Algorithm

of its' end point and appears in its middle. The segment also happens to include the link that connected 17 and 19 in the previous step. This happened because the algorithm found path $\{19, 16, 15, 14\}$ and then found $\{17, 19, 20, 14\}$. Notice this is not optimal cycle but all nodes have been included. The non-optimality can be observed by noting there exists paths $\{17, 20, 14\}$ and $\{19, 16, 15, 14\}$ which could have been used to create a simple cycle using one less link. Since all nodes in S have been included the algorithm terminates.

3.5 Special Cases

In order to give a clear explanation of the execution of our algorithm, we omitted details relating to a couple of special cases in algorithm description. Case l: It

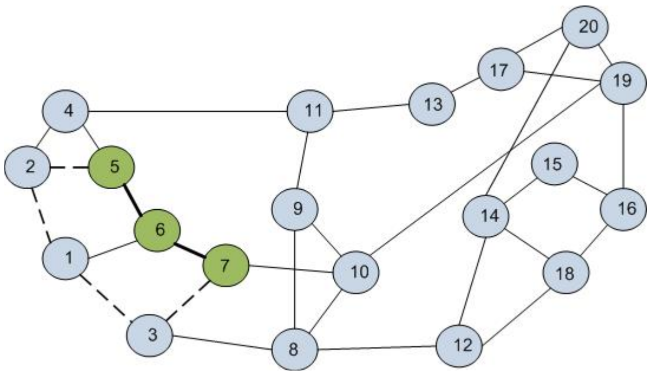

Fig. 5. Example of Special Case 1

is possible for all MP nodes to be included in the path found in Phase I. If this happens then there are no uncovered MP nodes to include when closing the cycle. In this case we simply find the shortest path between the two end nodes of Initial $P(T)$ or report failure if there is no such path.

Figure 5 shows the example of Special Case 1. The MP-nodes in the network are $\{5, 6, 7\}$. The Initial $P(T)$ (solid line) includes all the MP-nodes in its path $\{5, 6, 7\}$. Since all the MP-nodes are covered, our heuristic algorithm finds the shortest path $\{5, 2, 1, 3, 7\}$ between end MP-nodes of Initial $P(T)$ and the algorithm terminates.

Case 2: Sometimes the selection of the Initial $P(T)$ creates a situation where the cycle cannot be closed. Instead of reporting failure in this case we start inserting uncovered MP nodes. Inserting new nodes may free up critical resources allowing the formation of a cycle. If we could not close the cycle initially, we check if the cycle can be closed after each insertion of a new segment.

The example for Case 2 is shown in Figure 6. The Initial $P(T)$ is the path $\{1, 2, 3, 9, 17, 23, 24, 25\}$. Since the links in Initial $P(T)$ cannot be reused, this disconnects the graph preventing the completion of a cycle. When MP node 13 is inserted between MP nodes 2 and 24, the edge between nodes 9 and 17 becomes available. This reconnects the graph allowing for the cycle to be created.

4 Variations of MCR Heuristics Algorithm

To explore the impact on optimality by various design features of our heuristic algorithm, we considered four variations of algorithm MCRA. The variations are

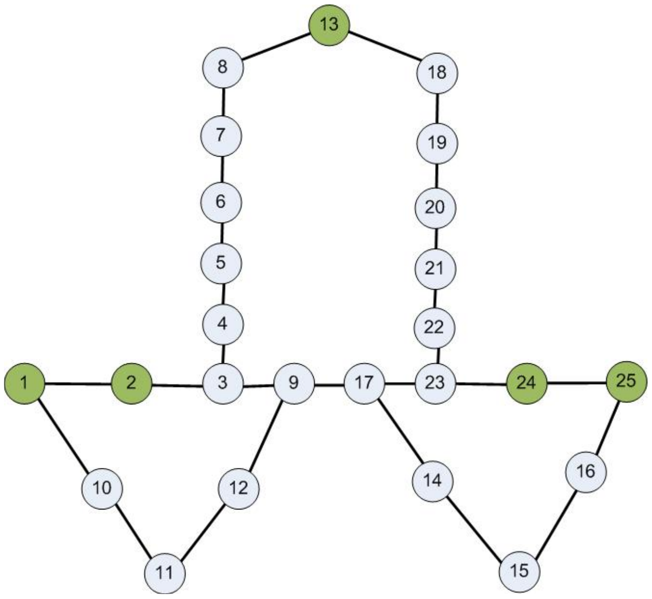

Fig. 6. Example of Special Case 2

obtained by using different node cost and adjusting when the cycle is likely to be closed. The cycle closing can be adjusted by skipping Phase 2 and considering the segments between end nodes of P(T) as part of the segments found between consecutive multipoint nodes in Phase 3.

We use notation ij to describe the four variations of the heuristic algorithm. Index i takes binary value of 0 or 1 whereas Index j takes a value of Y or N. Index i describe the assigned node cost function and Index j describe the inclusion of Phase 2. The four cases are as follows.

– $0Y$ - Multipoint nodes have cost 0 and the other nodes have cost 1. Phase 2 is included.
– $0N$ - Multipoint nodes have cost 0 and the other nodes have cost 1. Phase 2 is skipped.
– $1Y$ - The cost of both multipoint and non-multipoint nodes are same. Phase 2 is included.
– $1N$ - The cost of both multipoint and non-multipoint nodes are same. Phase 2 is skipped.

In Section 7, we analyze the performance of these four variations of our heuristic algorithm in terms of average cycle length and percent blocking in different graph types.

5 ILP Formulation to Find Multipoint Cycles

This section describes an ILP that solves the multipoint cycle routing problem. In most of this paper we have treated the network graph as undirected. In formulating the ILP it was more convenient to assume directed edges. This does not give rise to inconsistencies, since a directed cycle can be converted to a bidirectional cycle by including the edge between nodes in the cycle that go in the direction opposite to the selected edges.

We solved the multipoint cycle problem using an ILP that minimizes the number of edges in the cycle. This is accomplished by the objective function given as in Equation 1. The set of edges found as output should satisfy the following three constraints: edges traveling in opposite directions cannot be used, the selected edges form a cycle, and all S nodes are included in the cycle. The last property is enforced by a flow constraint. The selection problem is related to the flow constraint by requiring that the flow only uses selected edges. The constraints for controlling the flow are written in terms of the following variables and constants, which are indexed over the set of edges $e \in E$ in the network as well as nodes $n, m \in V$ in the network.

The constant values used to specify the ILP are as follow.

$$j_n = \begin{cases} |S| - 1 \text{ for one arbitrary } s_0 \in S \\ -1 \text{ for } s_n \in S, s_n \neq s_0 \\ 0 \text{ for } v_n \in V, v_n \notin S \end{cases}$$

$$d_{e,n} = \begin{cases} -1 \text{ for entering node } n \text{ from edge } e \\ 1 \text{ for leaving node } n \text{ from edge } e \end{cases}$$

$$x_{e,n,m} = \begin{cases} 1 \text{ if edge } e \text{ connects nodes } n \text{ and } m \in V \\ 0 \quad \text{otherwise} \end{cases}$$

The variables and the values they may take are as follows.

$$f_e = \Big\{ \text{flow on edge } e$$

$$l_e = \begin{cases} 1 \text{ if link } e \text{ is selected} \\ 0 \quad \text{otherwise} \end{cases}$$

Objective Function

$$Minimize \sum_{e=1}^{|E|} l_e \tag{1}$$

The constraints for the ILP are given below:

We setup a flow problem that can only be satisfied if all S nodes are connected by selected links.

$$\sum_{e=1}^{|E|} d_{e,n} f_e = j_n \tag{2}$$

The selected number of incoming links equals the selected number of out going links. This forces the selected links to form one or more cycles. Satisfying the flow problem in 2 forces the selection to be a single cycle.

$$\sum_{e=1}^{|E|} d_{e,n} l_e = 0 \tag{3}$$

Both the edges between a node pair n,m cannot be selected at the same time.

$$\sum_{e=1}^{|E|} x_{e,n,m} l_e \leq 1 \tag{4}$$

If the connectivity enforcing flow uses edge e, then l_e will show that it has been selected. Since the amount of flow is just a book keeping device for checking connectivity, l_e is scaled by the largest possible flow size.

$$|V| l_e \geq f_e \tag{5}$$

We have used this ILP to find the average cycle length for set of random requests in Arpanet and the comparison with that of best performing heuristic is shown in later section.

6 Random Graph Generation for Heuristic Evaluation

To evaluate our algorithms, we have generated random graphs with the following characteristics. Fault-tolerant connections are not possible in 1-connected graphs; hence we needed to ensure that the random graphs are at least 2-connected. We achieve this 2-connectedness by starting with a Hamiltonian cycle passing through all the nodes. Forcing a graph to have a Hamiltonian cycle is a stronger condition than necessary to create a 2-connected graph. Since we start with a Hamiltonian cycle in all the random graphs, a solution always exists to any multipoint request. This characteristic of random graph helps us evaluating the performance of our algorithm. This is because a cycle solution always exists. The blocking cases reported in later section are due to the failure of our algorithm.

We assumed that the physical distance between nodes affects the probability of placing a link between them. Nodes are randomly placed inside a 20x20 grid. On top of this Hamiltonian cycle, we add the rest of the links in the graph randomly with probability given by Equation 6 following the Waxman model. In Equation 6, s and d are nodes, $D(s,d)$ is the distance between the nodes computed by using their coordinates in the grid. α and β are parameters that can be adjusted.

$$P(s,d) = \beta e^{\frac{-D(s,d)}{L\alpha}} \tag{6}$$

The parameters α and β can be adjusted to control the expected density and the average link length of a generated graph. Consider a graph of n nodes and kn edges. We consider a graph as sparse if k is about two, medium density if $k = (2 + \log_2 n)/2$ and dense if $k = n \log_2 n$. To achieve a target density the values of α and β must depend on the selected value of n. We specify six types of graphs in terms of link density and length, as discussed in Table 1. Values of α and β used to realize the target link density and lengths for 64 node Type 0 to Type 5 networks are also given in the table.

Table 1. Graph Types With α, β for Sixty Four Node Networks

Type	Density	Length	α	β
0	sparce	short	0.2	0.2
1	sparce	long	0.8	0.05
2	medium	short	0.2	0.57
3	medium	long	0.8	0.15
4	dense	short	0.2	0.8
5	dense	long	0.8	0.27

7 Numerical Analysis

In this section, we will analyze the performance of four variations of our heuristic algorithm in different types random graphs, based on cycle length and percent blocking as evaluation metrics. We consider one algorithm to be best performing which has least percent blocking and cycle length close to the best solution found by all variations of our algorithm.

All four variations are executed on six types of random networks (sample size of 12 for each type mentioned in Table 1) of size 65 nodes for 10,000 requests each of which had 10 multipoint nodes. The average length of cycle generated by these variations for all the networks is shown in Figure 7. The variations $1N$ and $0N$ outperform other two heuristic variations in terms of cycle length, but it can be seen from Figure 8 that the blocking performance is too high in comparison to other algorithms. High blocking percentage nulls the advantage of getting smaller cycle length using $1N$ and $0N$ algorithms. Since success rate of the request is an important criteria for any algorithm, the algorithms $0Y$ or $1Y$ are preferred as they have lower blocking percentage although the average cycle length is slightly more than that yielded by $0N$ and $1N$ algorithm. We notice that node cost attribute have minimal impact on the average multipoint cycle length in all random networks.

The performances of four variations of heuristic algorithm are analyzed for these six types of networks and also for different network sizes. The average cycle length generated by all four variations of heuristic in six types of random graphs increases with the network size. This is because for large networks, multipoint

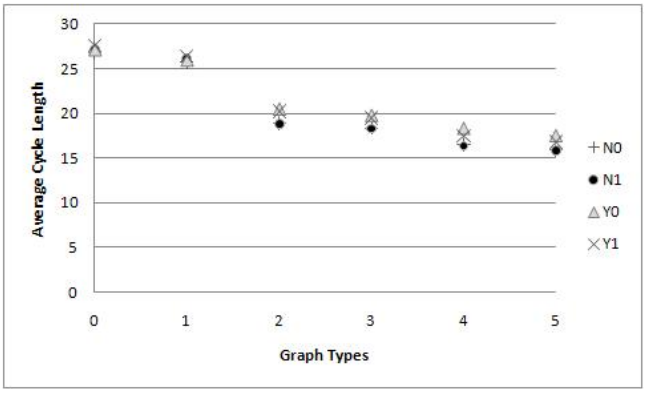

Fig. 7. Average Cycle length for 65 nodes Network

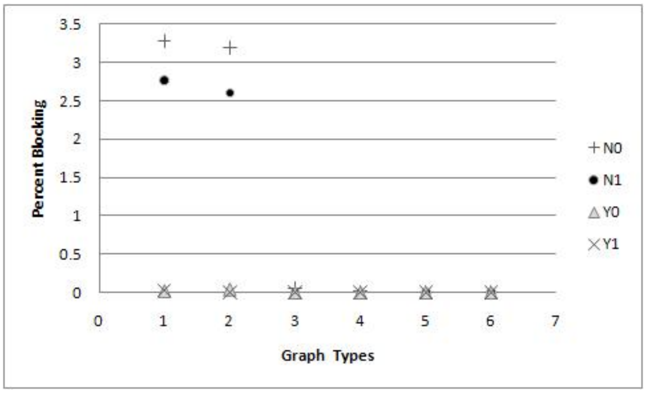

Fig. 8. Blocking Percentage for 64 nodes Network

nodes can be distributed far from each other. Figure 9 depicts that variation $0N$ gives the lowest average cycle length for most network sizes in case of sparse link density (Type 0). For networks with link density medium or higher, variation $0Y$ gives longer cycle length for most network sizes.

Figure 10 shows the average cycle length of all four variations for Type 5 network. The cycle length proportionate to the network size characteristic of algorithm persists for all types of random graphs. Since Type 5 networks have denser link distribution, the maximum cycle length tends to be smaller than that of Type 0 networks.

Figure 11 shows that in case of sparser (Type 0 and Type 1) and medium link density (Type 2 and Type 3) networks the variations $0N$ and $1N$ have many blocked requests whereas the blocking percentage of $0Y$ and $1Y$ algorithms is very close to zero. For dense (Type 4 and Type 5) networks, there is no blocking

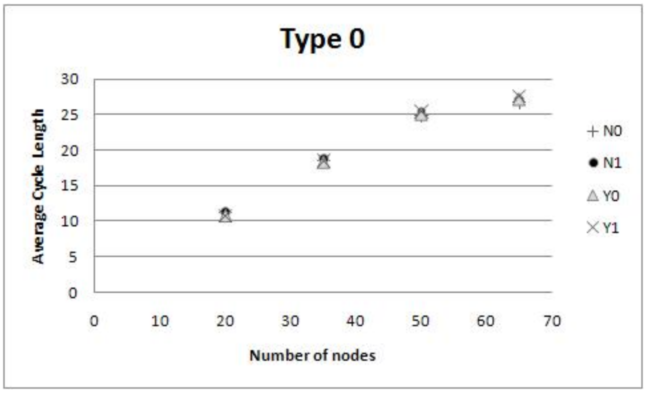

Fig. 9. Average Cycle Length for Type 0 Network

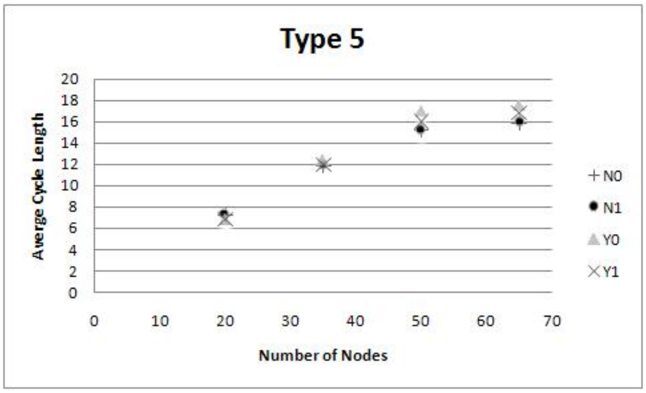

Fig. 10. Average Cycle Length for Type 5 Network

for all variations of algorithms. $1Y$ algorithm always finds a solution for large 65-node networks of all types except for Type 0.

Since the blocking percentage of the algorithms $0N$ and $1N$ is very high for sparser networks (Type 0) and the cycle length found by $1Y$ algorithm does not differ much from $0N$ and $1N$ algorithms, we will be considering $1Y$ algorithm to compare with the optimal results founds by ILP. The reason for selecting $1Y$ over $0Y$ is that we have a done studies of their relative performance for dynamic traffic and $1Y$ had lower blocking. Hence $1Y$ is our recommended heuristic.

7.1 Comparison between Heuristic and ILP

An idea commonly employed by routing algorithms is that a request should be routed with the minimum amount of resources possible. Integer Linear Program (ILP) is used to find optimal routes for some randomly generated requests.

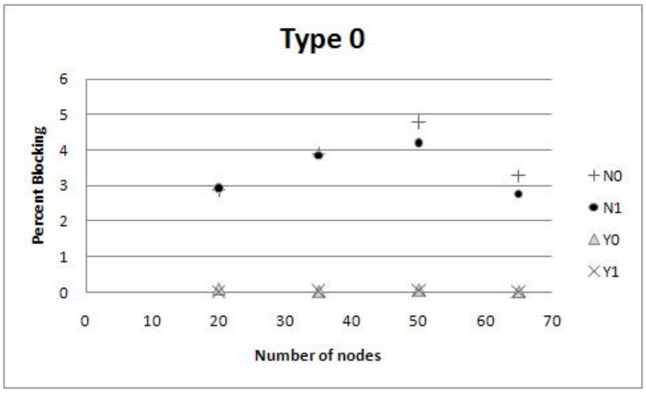

Fig. 11. Blocking Percentage for Type 0 Network

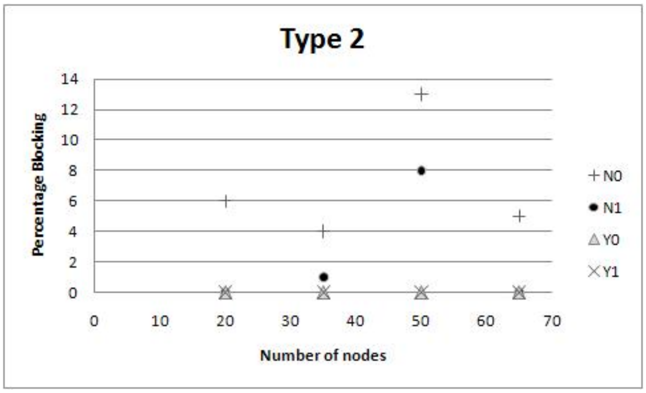

Fig. 12. Blocking Percentage for Type 0 Network

We use CPLEX to solve our ILP. A thousand requests are randomly generated and then routed using ILP. The requests are also routed using our MCRA algorithm. For each of these thousand requests, the ratio is found between the cycle lengths of heuristic and ILP. All results in this section are for Arpanet (20 Nodes, 31 Links). Arpanet has a Hamiltonian cycle so there exists a solution to any possible multipoint request.

This experiment is repeated for requests with different number of multipoint nodes, which are five, seven, nine and eleven. The curves are marked as $S5$, $S7$, $S9$, and $S11$ in Figure 13 respectively for 5,7,9,11 MP-node cases. The list of cycle length ratios of each experiment is sorted and displayed in Figure 13. The X-axis refers to thousand requests used in the experiment and the Y-axis refers to their cycle length ratios. When a quarter of the nodes are multipoint

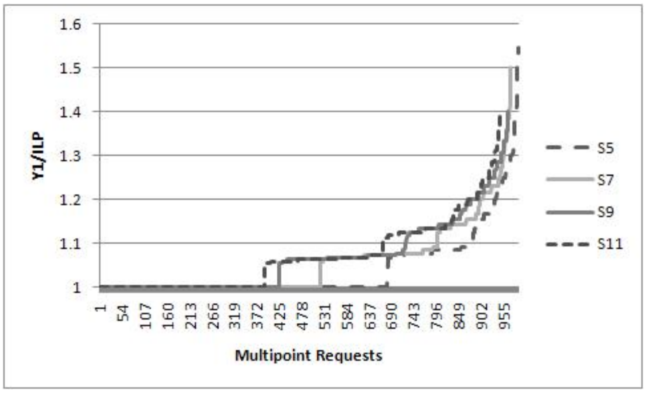

Fig. 13. Ratio Of Cycle Lengths Y1:ILP

nodes ($S = 5$), optimal resource usage occurred for more than 50% of the total requests. When just over half the nodes in the network are multipoint nodes, optimal resource usage occurred over thirty percent of the time. The figure shows that heuristic ($1Y$) algorithm yielded performance within a factor of 1.2 of the optimal performance over 80% of the time. Requests, where the MCRA failed to find a cycle, are obviously not included in the Figure 13.

8 Conclusion

Multipoint communication can be protected in such a way that single link failures are automatically restored by the communication protocol without the time consuming process detecting and reacting to the link failure. This is accomplished by forming a bidirectional cycle inside a mesh network and transmitting both directions along the cycle. In this paper we developed a multipoint cycle routing algorithm (MCRA) and provided various heuristic to find a cycle including nodes wishing to form the multipoint session along with a small number of extra nodes. The small number of extra nodes is forced by the network topology. Since we are trying to control the computational complexity of the algorithm, the number of extra nodes included in the cycle are not minimal. We compared the performance of our algorithm to the results of an ILP with the same set of random requests on Arpanet. The algorithm yielded performance within a factor of 1.2 of the optimal performance over 80% of the time.

Acknowledgements. Research funded in part by NSF project CNS 0626741, a Department of Education GAANN Fellowship, and the Jerry R. Junkins Endowment at Iowa State University. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation (or another funding agency). We would like to thank Onur Turkcu for helpful discussion during the project.

References

1. Zhang, F., Zhong, W.: Performance Evaluation of Optical Multicast Protection Approaches for Combined Node and Link Failure Recovery. Journal of Lightwave Technology 27(18), 4017–4025 (2009)
2. Khalil, A., Hadjiantonis, A., Ellinas, G., Ali, M.: Pre-planned multicast protection approaches in wdm mesh networks. In: 31st European Conference on Optical Communication, ECOC 2005, 25-29, vol. 1, pp. 25–26 (2005)
3. Ramamurthy, S., Sahasrabuddhe, L., Mukherjee, B.: Survivable WDM mesh networks. Journal of Lightwave Technology 21(4), 870 (2003)
4. Singhal, N., Sahasrabuddhe, L., Mukherjee, B.: Provisioning of survivable multicast sessions against single link failures in optical WDM mesh networks. Journal of Lightwave Technology 21, 11–21 (2003)
5. YuQing, G., Beijing, C.: Protecting Dynamic Multicast Sessions in Optical WDM Mesh Networks
6. Wen-De Zhong, F.: Applying p-Cycles in Dynamic Provisioning of Survivable Multicast Sessions in Optical WDM Networks. In: Conference on Optical Fiber Communication and the National Fiber Optic Engineers Conference, OFC/NFOEC 2007, pp. 1–3 (2007)
7. Feng, T., Lu, R., Zhang, W.: Intelligent p-Cycle Protection for Multicast Sessions in WDM Networks. In: Proc. ICC, vol. 8, pp. 5165–5169 (2008)
8. Zhang, F., Zhong, W., Jin, Y.: Optimizations of p-Cycle-Based Protection ofOptical Multicast Sessions. Journal of Lightwave Technology 26(19), 3298–3306 (2008)
9. Zhang, F., Zhong, W.: Performance evaluation of p-cycle based protection methods for provisioning of dynamic multicast sessions in mesh WDM networks. Photonic Network Communications 16(2), 127–138 (2008)
10. Wen-De Zhong, F.: p-Cycle based tree protection of optical multicast traffic for combined link and node failure recovery in WDM mesh networks. IEEE Communications Letters 13(1), 40–42 (2009)