2011

Impairment aware cycle based routing algorithm (IACBRA)

Suresh Sankaran
Iowa State University

Impairment aware cycle based routing algorithm (IACBRA)

by

Suresh Sankaran

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OFSCIENCE

Major: Computer Engineering

Program of Study Committee:

Arun Somani, Major Professor

Manimaran Govindarasu

Daji Qiao

Iowa State University

Ames, Iowa

2011

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENTS

I take this opportunity to express thanks to those who made a difference in my life for the last couple of years in Ames. First and foremost, I thank Dr. Arun Somani for giving me an opportunity to do research at Iowa State University. He gave me complete freedom and guided me from start till the end. He was more than a research advisor to me, by teaching important lessons which will help me both professionally and personally. In each and every meeting, his talk inspired and motivated me to raise the bar and pursue higher goal. I would like to thank Dr. Daji Qiao for teaching the course Wireless Sensor Networks entirely from research perspective. The challenging syllabus of this course helped to sharpen my research oriented skills. I thank Dr. Manimaran Govindarasu for giving moral support and accepting my request to be a member of my POS committee. I also thank Dr. Lei Ying for his guidance and help in peer to peer project.

I specially thank David Lastine for sharing his expertise and for valuable discussions we had while solving various research problems. I thank Prasad and Pavan for their comments and suggestions on my research paper. I thank Prem and Indian Students Association for all their help during my initial days at Iowa State University. I also thank Koray, Harini, Cory, Ding, Lizandro and other members in our research group for their enlightening talks on current research topics.

This research in part was funded by NSF project CNS 0626741 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 and his MS thesis supervisor and do not necessarily reflect the views of the National Science Foundation or the other funding sources.

ABSTRACT

A cycle could support a Synchronous optical networking (SONET) ring, or a reliable multicast, or a multipoint to multipoint traffic request, or a p-cycle protecting connections between any two-node pair that belong to the cycle. Determining a cycle passing through a specific set of nodes is a complex process. This paper presents a novel heuristic algorithm to find a least cost possible simple or non-simple cycle with multiple must-include nodes. In case the required cycle must be a simple cycle only, then it is apparent that our problem is similar to a Travelling Salesman Problem, in which the objective is to find the least cost simple cycle with all the nodes in a given travel network. In our algorithm, we allow complex cycles in the sense that links are allowed to be used only once, whereas a node may appear multiple times. This kind of cycle is called a non-simple cycle and it can increase the success rate of finding the cyclic route with desired set of nodes. The heuristic algorithm in this paper utilizes Breadth First Search and Tree construction algorithms to find a least cost and shortest route between nodes. The experimental results show that our algorithm has lower failure rate of requests than existing heuristic algorithms like Optimized Collapsed Ring (OCR) and Enumeration. The fault-tolerance for single link failure can be achieved by routing the traffic in both the directions of the cycle.

Chapter 1 Introduction

The need for creation of cycles in a mesh network involving some targeted set of nodes occurs in many different contexts in connection routing. In Metropolitan core networks sometimes the traffic has to be routed through specific set of nodes which has wavelength conversion or grooming functionality. Also in SONET ring architecture, creation of disjoint cycles form integral part of protection. An example of SONET ring two fiber unidirectional protection is shown in Figure 1-1, where all data from Digital Cross Connects (DCS) or Add Drop Multiplexer (ADM) are transmitted on the working or active path and the standby path (protection path) lies in waiting. When a failure in the active path occurs, the two network nodes affected immediately switch to the standby line, provided the standby path is not in close proximity to the first and also damaged. So the cycle based routing finds significant application in Core networks.

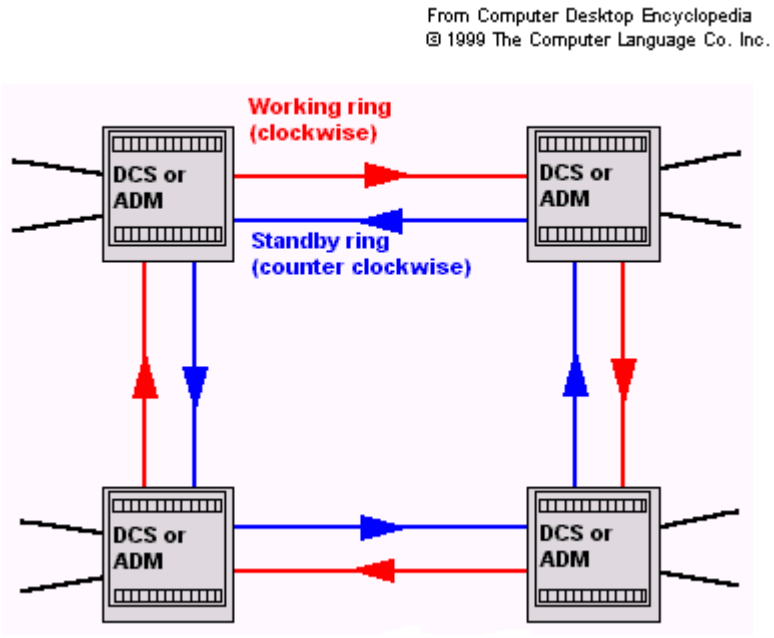

Figure 1-1: Two Fiber Unidirectional SONET ring architecture

.

In some cases the cycle may be needed while satisfying the requirement of failure location independent protection. For example, in protected point to point connection, the cycle is created implicitly when both working and protected paths are concatenated together. Consider the path protected point to point connection shown in Figure 1-2. The primary path that carries traffic between nodes A and B is shown in dashed line. The backup disjoint path is shown in dotted lines. These two paths implicitly form a cycle passing through nodes A and B.

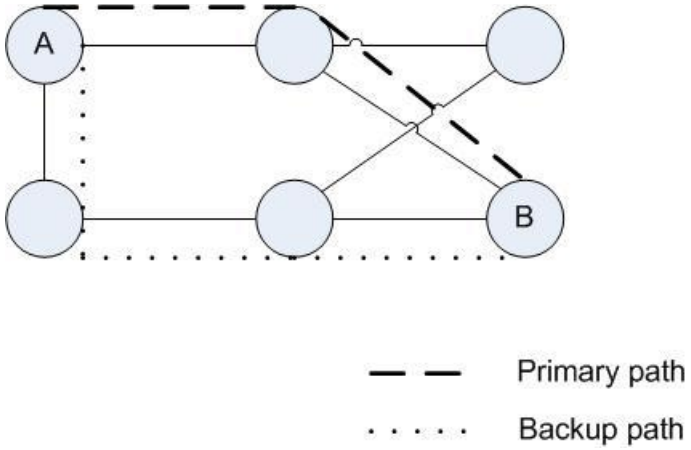

Figure 1-2: Need for a cycle for a point to point connection

A cycle creation is also simple and useful way to share path level protection amongst many point to point connections. Consider Figure 1-3, which depicts a network with nodes A, B, C, and D have the following connections between them: A-B, B-C, C-D, A-C, and B-D. All these connections can be protected from single link failure in the network if nodes A, B, C, D are connected by a cyclic route (A-B-D-C-A) that has enough unused bandwidth. This cycle provides protection in both the cases of a single link forming the entire path (as is the case with A-B) or in the case of a multilink path such as the connection A-X-C.

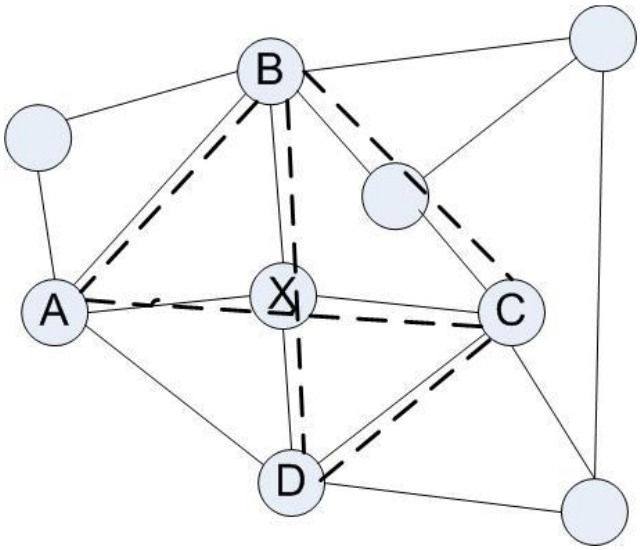

Figure 1-3: Cycle creation for many point to point connections

In this paper, we consider only Multipoint communication where we wish to create a cycle that includes all nodes of interest. Multipoint communication between a set of nodes say {A, B, C} can be defined as multiple simultaneous multicast sessions. In each of the session, at a time either of A or B or C is a source node and the others nodes are destination nodes. Multipoint communication has numerous applications on the Internet including distributed database applications, negotiation and e-commerce systems, online games, and various forms of video-conferencing. Most of these applications are real-time and require the routing to satisfy a specified upper bound on the end-to-end delay. We only focus on multipoint communications, which has predefined fixed set of participating nodes and require a possible least cost route, like in Metropolitan Core Networks (SONET/DWDM).

We propose a heuristic algorithm which solves the multipoint communication by presenting single route for all multicast sessions in a single multipoint communication. Our heuristic algorithm tries to find a least cost and possibly a shortest cycle passing through all must-include nodes. Internally our algorithm uses a Breadth First search approach to construct a tree and to find the shortest and least cost path between two nodes as described in later sections.

For fault tolerance, we transmit the traffic in both directions of the cycle. This is similar to Unidirectional Path Switched Ring (UPSR) protection. This ensures that in case of a link failure, the traffic reaches all nodes in the cycle without interruption. The transmitting node is responsible for ensuring that the signal is dropped upon return after travelling the complete cycle. All nodes listen to signals 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.

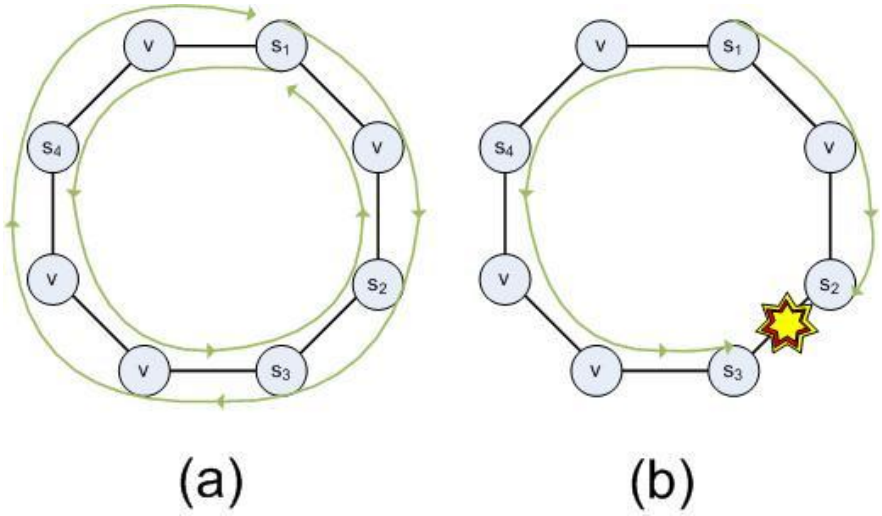

Figure 1-4: Fault-tolerant protocol

An example of this fault tolerance protection with s_1 as source node and s_2, s_3, and s_4 as destination nodes is shown in Figure 1-4 (a). When a fault occurs, as shown in Figure 1-4 (b), all the destination nodes still receive the transmitted information, but only once, without any knowledge of failure in the network and without any added failure restoration delay. By this kind of protection scheme, we eliminate the cost of a dedicated disjoint backup path and also reduce the latency since the traffic is transmitted in both the directions. When compared with other protection mechanisms like dual multipoint tree and p-cycle, the multipoint cycle based protection scheme has advantages like no loss of traffic, faster restoration, and no need to detect the fault location.

Chapter 2 Review of Literature

In [1], Bertsekas defines the routing function in two parts. The first part selects a route for the session during the connection establishment phase, and the second part ensures that each packet of that session is forwarded along the assigned route. In this paper, we consider only the route selection algorithms. As described in [2], Multicast routing algorithms can be classified into one of two categories. The first category is the shortest path algorithms which minimize the cost of each path from the source node to a multicast group member node. Bellman–Ford's algorithm and Dijkstra's algorithms are two well-known shortest path algorithms; they are the basis of the distance vector and link state routing protocols, respectively. The other category is the minimum Steiner tree algorithms. Their objective is to minimize the total cost of the Multicast tree. Efficient minimum Steiner tree heuristics are given in [3], [4] and [5].

In most of the existing research, the multipoint routing problem has been studied in the multicasting domain for transmission from one source to a given set of receivers, thereby restricting the consideration of multiple sources for the performance analysis of the proposed solutions.

In [2], the authors have proposed a heuristic algorithm to find the shortest simple path with multiple must-include nodes, by finding the shortest path between each consecutive pair of nodes in must-include set and concatenating those segments to create a simple path. They didn't define on what basis the nodes are reordered in the must-include nodes (I) set. Moreover the time complexity of this heuristic is exponential in terms of size of the set, I. Their results show very high blocking percentage of more than 20% of requests failed for normal network scenario. The routing of multipoint connections in a large scale packet switched network is addressed in [7]. It solves the multipoint routing with the shortest sub tree which contains the set of must-include nodes. Finding the shortest tree or total cost optimized tree is Steiner minimal Tree (SMT) problem, a well-known problem, which is NP-Complete [8]. In **[**9**]**, the authors explored

the potential solutions for the problem of delay-constrained multipoint communication with multiple sources in a distributed environment by optimizing the number of cores for tree construction. They extended the existing solution for multicast tree routing to the case for multiple sources with improved efficiency in their approach.

Although there are many tree based heuristic algorithms for multicast routing in literature, those algorithms have their own disadvantages when extended to fault tolerant multipoint communication problem. Most of the tree based heuristics have high computation complexity for constructing a tree for each source. This approach might be optimal for multicast traffic but not for multipoint routing. Also it is not simple to provide fault tolerance in multipoint routing via trees.

Chapter 3 Cycle Based Routing Problem

3.1 Network Model

The network is modeled using a connected undirected graph G (V; E), V is the set of vertices representing nodes in the network, E is the set of edges (u, v) ϵ E where u, v ϵ V. An edge (u, v) represents the physical link that allows communication between nodes u and v. The Multipoint request contains a set of nodes, *S* ($S \subseteq V$). The final cyclic route found by the algorithm should include all the nodes in this set *S*. We refer to each element or nodes in the set *S* as S-nodes. Let *m* represent the size of set *S*. Let *s (p)* be the function which returns number of nodes belonging to set *S* in the path *p*.

3.2 Problem Definition

Given a graph G (V, E) and *S* a subset of nodes in *V* and *k* be an integer with $k <= |E|$,

Is there a simple or non-simple cyclic route p (V', E'), where V' is the set of nodes in p and E' is the set of edges in p, such that $S \subseteq V'$ *and* $|E'| <= k$*?*

This cyclic route could also pass through the nodes which are not in set *S*, as required based on network topology. If such a cyclic route cannot be established, then the request is rejected. This rejection is referred as blocked requests. Finding the smallest cycle that includes a specified set of nodes is a hard problem. We show that this problem is hard and decision version of it is NP-Complete.

Proof: We first note that CBR is in class NP. Given a valid solution, it is possible to provide a verification that it is a valid cycle and $|E'| <= k$ in polynomial time. To demonstrate that the decision version of CBR problem is NP-complete, we show that an instance of the

Hamiltonian cycle problem reduces to an instance of the CBR problem in polynomial time. The Hamiltonian cycle problem asks if a graph G (V, E) has a simple cycle that visits every $v \in V$ in graph G. This can be reduced to an instance of CBR problem as follows. Let's build graph for CBR problem with $S = V$ and $k = |V|$. Then CBR problem for graph G'' (V'', E''), k, and S is equivalent of finding a Hamiltonian cycle in G.

Assume G has a Hamiltonian Cycle. Then the Hamiltonian cycle is the required solution for CBR problem which goes through all nodes in Graph G and satisfies the constraint of $|E'| = k$. Also, note that since $k = |V|$ the solution for CBR problem can only be a simple cycle, as n nodes can be connected with n edges only in a simple cycle, i.e., each node can appear only twice as the end nodes of edges. No non-simple cycle can satisfy $k = |V|$ and $S = V$ simultaneously. The solution for CBR problem cannot have $|E'| < k$, since there would exists a node $s_i \in S$ and not present in V'. Assume there is a solution for CBR problem with $S = V$ and $k = |V|$. This implies every node is visited once and all the nodes are included in the cycle. Thus it reduces into solution for Hamiltonian cycle problem.

Since the reduction can be obtained in polynomial time and a solution can be verified in polynomial time, CBR problem is NP-complete. Therefore, we develop a heuristic algorithm to find an appropriate cycle to route the given multipoint requests.

Chapter 4 Cycle Based Routing Algorithm (CBRA)

4.1 Algorithm Description

First, we developed a heuristic algorithm called Multipoint Cycle Routing Algorithm (MCRA). This algorithm is described in [6]. Given a graph, G and a set *S*, where {s_i represents *S-nodes*; for all $s_i \in S$}, as input, MCRA finds the near-optimal cycle in three steps.

1. In Step-1, MCRA finds an *Initial* path.
2. MCRA searches for *Disjoint*-path between the two end nodes of *Initial*-path in Step-II. If Step-II is successful, a cycle is formed passing through some of *S-nodes.* If all the *S-nodes* are included, then the algorithm terminates successfully thereby skipping Step-III.
3. In Step-III, S-nodes which are not included in steps I and II are inserted by replacing the existing segments in the cycle. If all the *S-nodes* are included in the cycle then the algorithm terminates reporting request is accepted otherwise the request is blocked.

Although MCRA algorithm has low time complexity, it has a high request blocking in finding the cycles. To improve the performance of the algorithm, we developed a second algorithm called Cycle Based Routing Algorithm (CBRA). CBRA also finds the near-optimal cycle in three steps as depicted in Figure 4-1. The main difference between MCRA and CBRA is the way path between two nodes is found in all three steps based on the method suggested in [11]. We will be using the term "*S-Optimal Shortest (SOS) path*" throughout the discussion to refer to the shortest path between s_i and s_j (s_i, $s_j \in S$) with maximum number of *S-nodes* on it. The flowchart of CBRA algorithm is shown in Figure 4-1 and the pseudo code is shown in Figure 4-5.

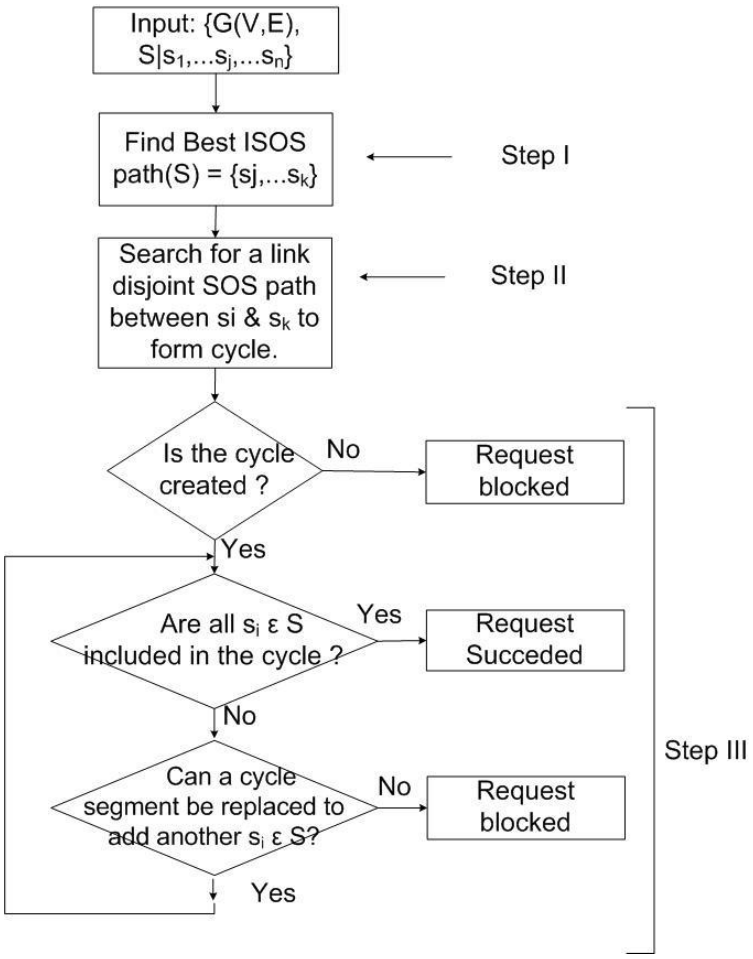

Figure 4-1: CBRA Flowchart

4.2 Evolution of Path Algorithm

In all three steps of our algorithm, we need to find a shortest path between two nodes. Since our goal in finding a cyclic route is to include as many S-nodes as possible, the path finding algorithm evolved over two different approaches to reach the successful Method 3 - Modified Breadth First Search algorithm. The three methods of path finding algorithm, evolution order of Method 1 to Method 3, are as follows.

1. Method 1 – Using Bellman-Ford algorithm

In method 1, we find a shortest path between two nodes using well-known Bellman-Ford algorithm. In this method we do not pay attention to S-node while finding the path, so we enhanced it with method-2 algorithm.

2. Method 2 – Modified Bellman Ford Algorithm

Using a suggestion from my fellow colleague, David Lastine, we modified Bellman-Ford algorithm by assigning weights of value zero to S-nodes and one to non-S nodes, and then we find a shortest path between two nodes which has lesser total cost. This method improves the path selection towards our goal of including as many S-nodes as possible. But this is further enhanced with method-3.

3. Method 3 – Modified Breadth First Search

In this method, using the suggestion from [11], we use Modified Breadth First Search algorithm to find a path between two nodes by specifically preferring S-nodes in the path. Modified Bread-First Search (BFS) algorithm is used to find "*S-Optimal Shortest (SOS) path"* in all the steps of CBRA. Since Modified-BFS algorithm forms an integral part of CBRA, it will be described first, before explaining the three steps of CBRA.

4.3 Modified Breadth First Search (MBFS)

4.3.1 S-optimal Shortest (SOS) path

The *S-Optimal Shortest (SOS) path* is found by Modified BFS algorithm. The algorithm begins at a node $s_i \in S$. MBFS traverses the entire graph starting with root node s_i in a breadth first search manner, but with the goal to create a shortest-path tree. The shortest-path tree is defined as a tree with the shortest path from root node to any child nodes. This tree is created in the following manner.

Let's say the root node s_i is at Level 0. This root node is connected to all of its neighboring nodes. These neighboring nodes are located at Level 1. We connect each of the nodes in Level 1 to all its remaining neighboring nodes, which are not already included in upper level of the tree. We ignore the edges connecting to the neighboring nodes which are present on the same level. These newly connected neighboring nodes of Level 1 are located at Level 2. The entire shortest path tree is constructed in similar fashion. The child nodes with more than one parent are resolved by choosing the parent with the higher number of S-nodes on its path to root. The pseudo-code for the tree-construction algorithm is shown in Figure 4-4.

We can verify that the MBFS finds *S-Optimal shortest path* by extending arguments for the correctness of a shortest path. The correctness of a shortest path found by MBFS comes from the fact that every segment of a shortest path must also be a shortest path; otherwise the segment could be replaced with a shortest path lowering the total path length. The length of a shortest path cannot be lowered, so no segment can be a non-shortest path between its end points. The introduction of a tie break means this is still the case, and a similar argument now applies to the number of *S nodes*. If any segment of a shortest path can be replaced with a segment of equal length that has more *S nodes*, then the path could not have been S-Optimal.

4.3.2 Illustration with an example

The MBFS and its effect of the tie break are illustrated in Figure 4-3. Here MBFS is executed on Arpanet with set *S* containing nodes {2, 5, 13, 14, 17, and 19} as shown in Figure 4-2. *S-nodes* are shown with darkly filled circles. The root node and destination nodes are 2 and 19 respectively. Part (a) of Figure 4-3 shows the Arpanet with root node at the top and rest of the neighboring nodes connected to it at one level. Also the links connecting nodes on same levels are removed. Similarly MBFS traverses the nodes in each level in Part (a) of the figure from left to right and constructs the tree. At each node MBFS selects its parent node, the one with shortest path to the root node; thereby it creates the shortest path tree. In Figure 4-3 (a) initially node 6 would have its parent set to node 1. Later when the possible children of node 5 are considered, the parent of node 6 would be changed to node 5 since the path from node 5 to the root has more S-nodes. This is indicated by the link between nodes 1 and 6 being crossed out. The entire shortest path tree is shown in Figure 4-3 (b). After the tree construction, the *S-Optimal shortest path* between nodes 2 and 19 can be found by back-tracking the parents of nodes from node 19 till it reaches root node 2. In MCRA, this S-optimal shortest path was found using Bellman-Ford algorithm.

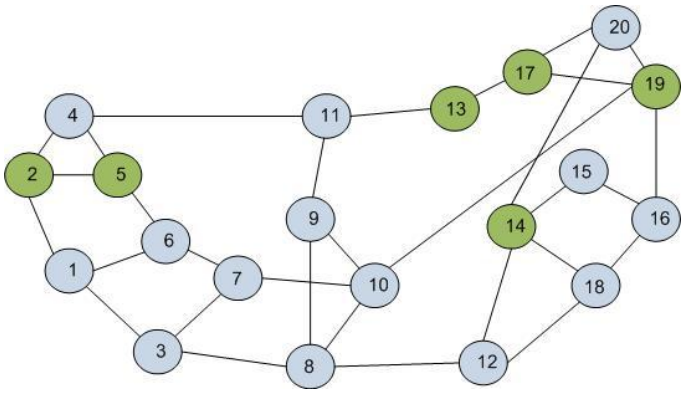

Figure 4-2: Arpanet network

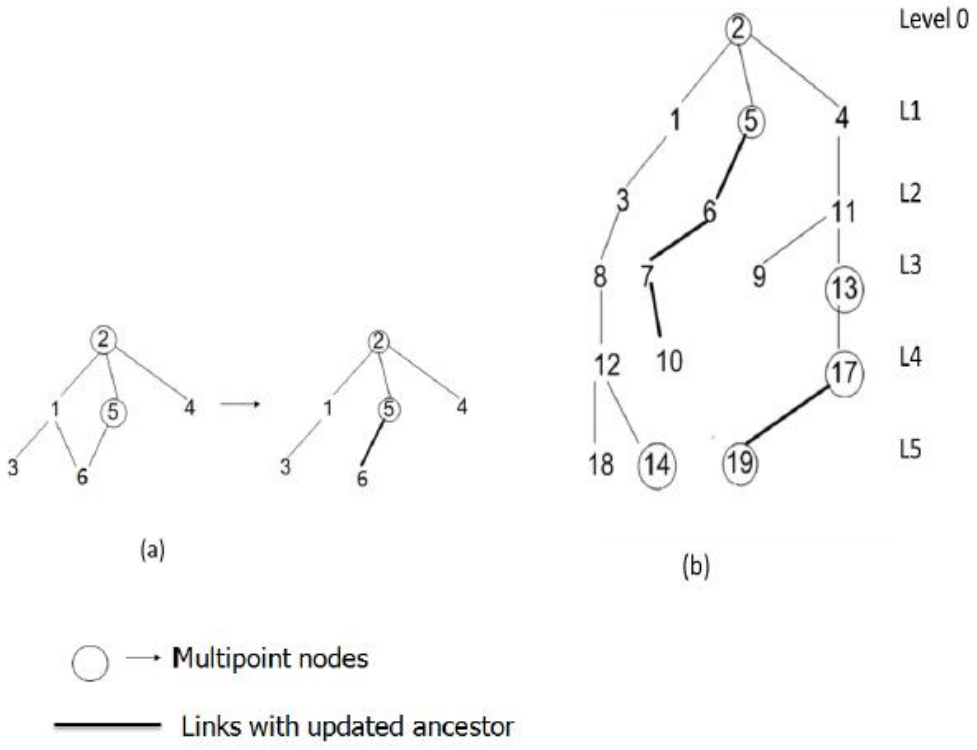

Figure 4-3: Construction of Shortest Path Tree

Once the shortest distance from the source to the destination is known, then extending the shortest path tree for all the remaining nodes is a waste of effort. Therefore our Modified Breadth First Search stops running, when the level with the possible parents for the destination node is processed. Even if the destination node is discovered early in the traversal of the level, that entire level must be processed since the parent node does not stabilize until the last node in that level is processed. In some cases like in Step-I of the CBRA algorithm, we don't exactly know the destination node beforehand to find the *S-Optimal shortest path* and the destination parameter for MBFS will be passed as NULL. Hence the entire shortest path tree (going through all the levels in the tree) has to be constructed by MBFS to select the destination node, $s_j \in S$ which has maximum number of S-nodes on its shortest path to root. If the destination parameter is NULL, MBFS returns the destination node, s_j and its *S-Optimal shortest path.*

```
Input: s_j, s_k
Output: A path from s_j to s_k
L_0 = s_j
L_1 = NULL
DoNextLevel = true
s_L = s_j
forall the s_h ∈ G do
    NS[s_h] = 0
    parent[s_h] == NULL
end
if s_k ∈ S then
    NS[s_k] = 1
end
else
    NS[s_k] = 0
end
while L_0 != NULL and DoNextLevel do
    foreach s_h ∈ L_0 do
        if s_h ∈ S then
            I = 1
        end
        else
            I = 0
        end
        foreach s_n ∈ Neighbor(s_h) do
            if parent[s_n] == NULL then
                SN[s_n] = SN[s_h] + I
                Level[s_n] = Level[s_h] + 1
                L_1 = L_1 ⋂ s_n
            end
            else if Level[s_n] > Level[s_h] and
            SN[s_n] < SN[s_h] + I then
                SN[s_n] = SN[s_h] + I
                Level[s_n] = Level[s_h] + 1
            end
            if s_n = s_k then
                DoNextLevel = false
            end
            if SN[s_L] < SN[s_n] then
                s_L = s_n
            end
        end
    end
    L_0 = L_1
    L_1 = NULL
end
if s_k == NULL then
    s_k = s_L
end
Return M(s_k)
```

Figure 4-4: Pseudo code of MBFS module

```
Input: Graph G, Subset S
Output: CompleteCycle - Complete Cycle
ISOS = NULL;
P = NULL;
//Step − I starts
foreach s_i ∈ S do
    P = MBFS(s_i, NULL);
    if # of S-nodes in P > # of S-nodes in ISOS then
        ISOS = P;
    end
end
T_i = NULL;
∀e ∈ ISOS  capacity(e) = unavailable;

//StepI ends; StepII starts
subpath = NULL;
s_j = First node in ISOS;
s_k = Last node in ISOS;
T_i = MBFS(s_j, s_k);
subpath = M(s_k, T_i);
if subpath == NULL then
    return NULL; request blocked
end
Insert subpath in Initial ISOS; ISOS becomes a
cycle(IC)
T_i = NULL;
//StepII ends; StepIII starts
while S ⊈ IC do
    foreach s_i ∈ S − IC do
        subpath = Select − SOSPaths(s_i, GraphG);
        if # of not-yet-included S-nodes in subpath > #
        of S-nodes in BestSOSpath then
            BestSOSpath = subpath; pick new segment
        end
    end
    if BestSOSpath == NULL then
        return NULL; request blocked
    end
    Insert BestSOSpath into P(T);
end
Return P(T); //StepIII ends
```

Figure 4-5: Pseudo code of CBRA

The variables used in the MBFS implementation module are described as follows.*L0* and *L1* hold two adjacent levels of the Shortest Path tree. *Parent* is an array storing node parents, *SN[s_i]* is an array storing the number of S-nodes in the path starting at the root and ending at nodes. *Level* is the array containing the distance of anode from the root node. The node s_L is a node that has the largest value in *SN*. If the destination parameter of MBFS is passed as NULL, then s_L is returned as the destination node. The pseudo code for M (s_k) is not provided as it is simply a tree traversal algorithm. It returns the path from s_j to s_k.

4.4 CBRA – Step I

4.4.1 Find Initial S-Optimal Shortest Path (ISOS)

In step I of CBRA, for a given set of *S-nodes*{s_1..,s_m}, we find the *S-Optimal shortest paths* between all possible pair (s_i, s_j) in the set S and select the path that contains maximum number of *S-nodes* in it. This special shortest path is called *Initial S-Optimal Shortest path (ISOS).*

We can find *ISOS* with the help of MBFS algorithm. For each *S-node*s, we invoke the MBFS code with the arguments s_j and s_k = NULL. The *S-Optimal shortest path* returned by first iteration is set as *ISOS* initially. Then we update *ISOS* for every other iteration, if the returned *S-Optimal shortest path* has more *S-nodes* than that present in current *ISOS* path. If the returned *S-Optimal shortest path* has same number of *S-nodes* as that in current ISOS path, then we select the path with least number of total nodes to be new*ISOS*.Step-1 is completed after all *S-nodes* are considered and outputs final *ISOS.*

4.5 CBRA - Step II

4.5.1 Create an Incomplete Cycle (IC)

In Step II, we find an *S-Optimal shortest path* between end nodes of *ISOS* path and hence try to create a cyclic route which passes through all or few of *S-nodes*. The *S-Optimal shortest path* found in this step is edge-disjoint from the existing *ISOS* path.

The *S-Optimal shortest path* is found by invoking MBFS algorithm with s_j and s_k as the parameters.s_j and s_k denote the end nodes of the *ISOS*. We do not allow the links that are already used in *ISOS* while finding the *SOS* path in this step. By doing so, we find an alternate, disjoint *SOS* path between the end nodes of ISOS. These disjoint *SOS* and *ISOS* paths are concatenated to create a cycle. If all the *S-nodes* are included in the cycle, then it is a Complete Cycle and CBRA terminates. Otherwise the cycle created is an incomplete cycle (IC) and the CBRA algorithm proceeds to Step III.

4.6 CBRA - Step III

4.6.1 Create a Complete Cycle

In Step III, we find *S-nodes* which are not yet included in Incomplete Cycle (IC) and try to insert them into IC, so as to create a complete cycle. Let's assume that these not-yet-included *S-nodes* are stored in set *S'*. We try to replace a segment connecting consecutive *S-nodes* in Incomplete Cycle (IC), with *S'-Optimal shortest path* that passes through one or more *S-nodes* in S'. Similar to the previous definition, *S'-Optimal shortest path (S'OS)* refers to the shortest path between two S-nodes with maximum number of S'-nodes on it. Then newly included S-nodes (ϵ S') are removed from S'. The above said mode of segment replacement is continued till the set S' becomes NULL and hence creating a complete cycle.

The segment in IC, which is replaced in every iteration to include one or more nodes from S', is selected as follows. For each node s_i in S', we determine the candidate segment connecting consecutive S-nodes in IC, which is to be replaced with a path passing through s_i. This is done by the module *Select - S'OSPaths*. This module finds the *S'-Optimal shortest path* between all valid consecutive *S-nodes* (s_j, s_k) in IC that passes through s_i; This S'OS path can reuse some of the links in the corresponding segment which it is about to be replaced, but not the other links present in IC; This S'OS path passing through s_i and between consecutive S-nodes (s_j, s_k) is found as two sub paths using the function *Find - S'OSPath*; one sub path between (s_j, s_i) and other sub path between (s_i, s_k); these two sub paths are nothing but *S'-Optimal shortest path* between the mentioned two end nodes. After finding all S'OS paths between every consecutive *S-nodes* segment in IC, which passes through s_i, we select one consecutive *S-node* segment as a candidate segment for s_i based on the maximum number of *S' nodes* present on that S'OS path.

Similarly we find the candidate consecutive S-node segments and its S'OS paths for all the nodes in S'. Then we select the final single candidate segment based on its S'OS paths that includes the largest number of nodes in S'. This final candidate segment is replaced by its S'OS path in IC. The S' nodes in this S'OS path are removed from the set S' and the Step III reiterates till it reaches one of the two cases.

Case one -The set S' is NULL. This implies that we have succeeded in finding the Complete Cycle for the given request.

Case two -There is no valid final candidate segment and S'OS path for any of the nodes present in S'. In this case the request will be blocked due to failure in finding the complete cycle with given set of nodes.

4.7 Functions used in CBRA

4.7.1 Select – S'OS Paths

Function *Select – S'OS Paths* is responsible for finding the candidate segment and its S'OS paths for a node in the set *S'*. One node from set S' i.e., s_i and the current Incomplete Cycle (IC) are input to the function *Select - S'OS Paths*. It performs the necessary work to loop over all consecutive S-nodes pairs on Incomplete Cycle (IC) starting with one pair of S-nodes s_j and s_k to include node s_i. It uses the function *Find - S'OS Path* to find the S'OS path between consecutive S-node pair on IC, s_j and s_k, passing through s_i. For each s_i, function *Find - S'OS Path* is called r times by this function *Select - S'OS Paths*, where r is the number of S-nodes currently in IC. The maximum value of r can be |S|- 1. Thus this function finds the S'OS paths for all the segments and returns the best of all the S'OS paths based on the one with maximum number of nodes in the set S'.

Select-SOS Paths

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

Find-SOS Path

Input: $P(T)$, s_i, s_j, s_k, Graph G
Output: $SOSpath$
$Subpath_1 = NULL$; $Subpath_2 = NULL$;
$Subpath_1 = Find - SOS - path - Helper(P(T), s_i, s_j, s_k$,Graph$G)$;
$Subpath_2 = Find - SOS - path - Helper(P(T), s_i, s_k, s_j$,Graph G); switches order of nodes
if $s(Subpath_1) > s(Subpath_2)$ **then**
  $SOSpath = Subpath_1$;
end
if $s(Subpath_2) > s(Subpath_1)$ **then**
  $SOSpath = Subpath_2$;
end
if $s(Subpath_1) == s(Subpath_2)$ **then**
  if $Subpath_1.cost > Subpath_2.cost$ **then**
    $SOSpath = Subpath_2$;
  end
  else
    $SOSpath = Subpath_1$;
  end
end
return $SOSpath$;

Find-SOS-path-Helper

Input: $P(T)$, s_i, s_j, s_k, Graph G
Output: $Subpath$
$T_i = NULL$; $T_j = NULL$; $path_1 = NULL$;
$path_2 = NULL$;
Set $s_k.Cost = |V(G)|$;
$path_1 = MBFS(s_j, s_i)$;
Set $s_k.Cost$ = 1;
$\forall e \in P_1$ capacity(e)=0;
Set $s_j.Cost = |V(G)|$;
$path_2 = MBFS(s_i, s_k)$;
Set $s_j.Cost$ =1;
$\forall e \in P_1$ capacity(e)=1;
if $path_1 = NULL$ *or* $path_2 = NULL$ **then**
  return $NULL$;
end
else
  return $path_1 \bigcup path_2$;
end

Figure 4-6: Pseudo-code of functions used in CBRA implementation

4.7.2 Find–S'OS Path

In this function we find S'OS path between consecutive S-nodes, s_j and s_k, in IC passing through a node in set S', s_i. This is accomplished with help of function Find – S'OS – path - Helper. We maintain the following condition throughout the process of finding the S'OS path. No link that is used on IC is available to the S'OS path s_j– s_i- s_k except the links on path s_j - s_k on IC. We find the path between nodes s_j and s_i by creating a shortest path tree rooted at s_j using the function MBFS (s_j, s_i). Similarly we find the path between s_i and s_k by creating a tree rooted at s_i. The union of the two paths forms the S'OS path for a not-yet-included node s_i which is to be inserted between consecutive S-nodes s_j and s_k. For the network shown in Figure 4-2 the IC is found to be 19, 17, 13, 11, 4, 2, 5, 6, 7, and 10 at the end of Step 2 of CBRA. The S'OS path between nodes 17 and 19 passing through node s_i i.e., 14 is found using trees in Figure 4.7 (a) and 4.7 (b). The union of these two paths passing through node 14 is 17, 20, 14, 15, 16, 19. We find this S'OS path for s_i in two directions s_j– s_i–s_k, s_k-s_i-s_j and return the path with maximum number of nodes in set S' and low total node cost.

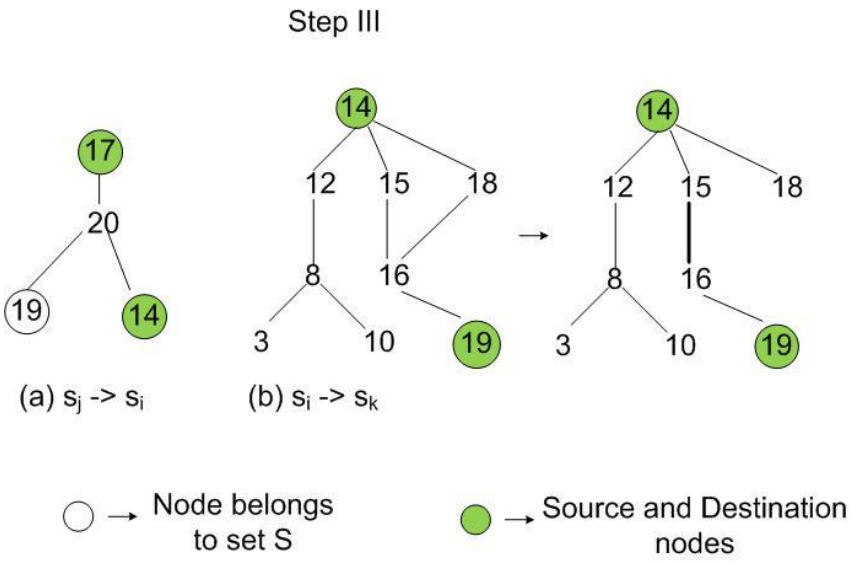

Figure 4-7: Step III of CBRA

4.8 CBRA Example

An example for CBRA on Arpanet is shown in Figure 4-8. The set of *S-nodes* received as input include 2, 5, 13, 14, 17 and 19. The path 2, 4, 11, 13, 17, 19 with solid line in the figure represents the Initial SOS Path (*ISOS*) found in Step 1. In the second step of CBRA, an alternate SOS path 2, 5, 6, 7, 10, 19 between end nodes of *ISOS* is found and inserted to create an Incomplete Cycle (IC) which is depicted in the figure with dotted line. Still the cycle is incomplete, because one S-node 14 is not included in the cycle. The *S'OS* paths between consecutive pairs of S-nodes on IC passing through node14 are found and the path between consecutive pair of S-nodes 17, 19, which is shown in dashed line in the figure, is selected, as it is the shortest and most optimized path. This path 17, 20, 14, 15, 16, 19 is inserted into IC to create a complete cycle and then the CBRA algorithm terminates successfully.

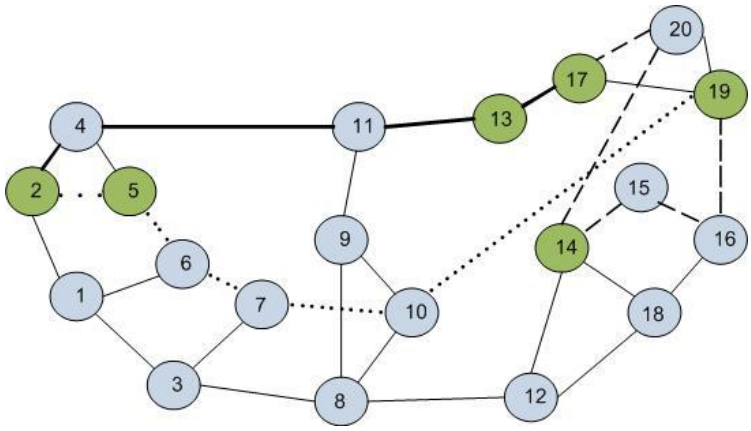

Figure 4-8: Arpanet with routes found in Steps I, II and III

4.9 Time Complexity Analysis

Let 'n' and 'q' represent total number of nodes and total number of edges in the network respectively. As we earlier mentioned 'm' represents number of multipoint nodes (S-nodes) taking part in the request. The time complexity for our algorithm in Step-1, Step-2 and Step-3 are O (mq), O (q) and O (m^2q), respectively. Hence the summed up complexity of our CBRA algorithm is O (m^2q).

4.10 Additional enhancements done over CBRA

Although we found that the blocking cases of CBRA are quite low when compared to MCRA, we enhanced CBRA further to reduce the number of blocking requests to zero. While analyzing the blocked requests for CBRA, we found an interesting observation. The most of the blocked cases are because of one or more 2-degree S-nodes could not be included in the cycle. Since 2-degree nodes are too restricted in terms of routes (maximum of one) passing through them, it reduces the probability of those nodes to be included in the incomplete cycle IC when CBRA tries to insert them in the third step. To overcome this, we developed a variation of CBRA in which the Step I of CBRA prefers the *ISOS* path with 2-degree S-nodes as its end nodes. This variation of CBRA algorithm is referred as *2-degree CBRA*. In Figure 4-9, we have shown an example for 2-degree CBRA algorithm. The initial path found in Step-1 has a two-degree S-node as one of its end nodes.

As another enhancement to the algorithm, we examined the process by which unwanted nodes were included in a cycle, in order to reduce the average cycle length. When we selected SOS and Initial SOS paths with a maximum number of *S-nodes* on its path, we noticed that these paths can also be paths with many unwanted nodes. Since this is inefficient, we tried modifying this selection criterion of SOS path in Step-1 to ratio of *S-nodes* to total number of nodes in the

path. The variation of CBRA algorithm with this ratio as selection criteria and also with preference for 2-degree *S-nodes* as end nodes of *ISOS* will be referred as *Enhanced CBRA* (ECBRA). Figure 4-10 shows an example for ECBRA algorithm. The initial path found in Step -1 has a two-degree end node and ratio is of value 1. Although there could be other path with a two degree end node and more S-nodes, but the ratio of S-nodes to total nodes cannot be higher than one, hence the path 13-17 is chosen in Step-1

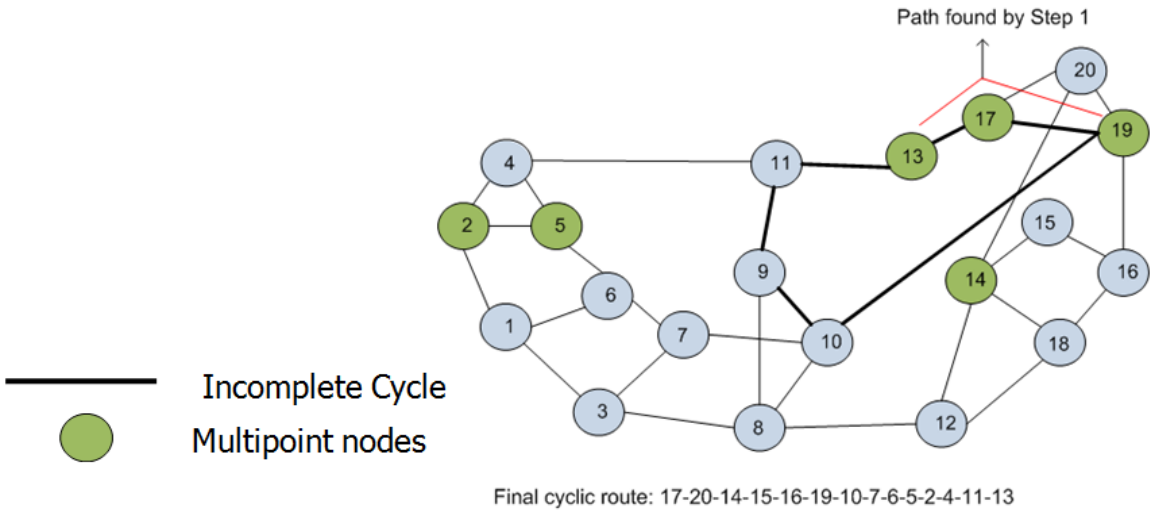

Figure 4-9: 2-Degree CBRA example

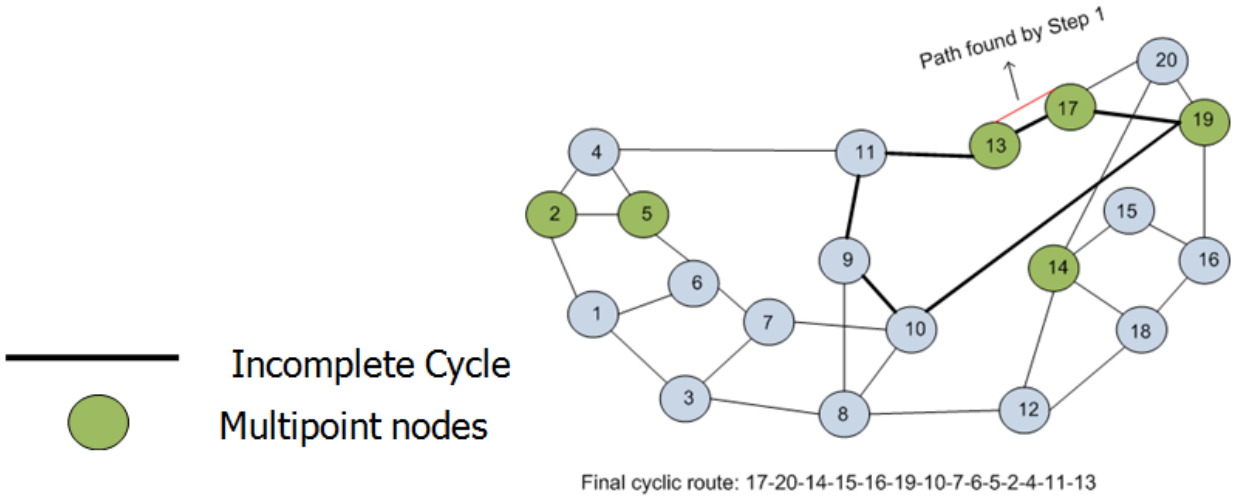

Figure 4-10: ECBRA example

Chapter 5 Other Cycle finding heuristics used for analysis

5.1 Enumerating All Simple Cycles

It is interesting to note that while the number of simple cycles in a graph can be exponential in terms of number of nodes, but for practical networks considered by researchers such as COST239, Arpanet, and NSFNET, the number of actual cycles, as determined by enumeration, in terms of the number of nodes is approximately $|V|^{3.5}$, $|V|^{2.25}$ and $|V|^{2.0}$ respectively. Of these three networks, the one with the most cycles is COST239, which has 3531 cycles, whereas Arpanet and NSFNET have 847 and 199 cycles respectively. These numbers are sufficiently small that grouping the cycles by sizes and checking each group for a cycle that can satisfy a request can be done in a reasonable time. Therefore, we chose to use this algorithm and compare its results with CBRA. Notice that since we are only enumerating simple cycles, it is possible we may not succeed for all requests. This algorithm is referred as Enumeration or Enum.

5.2 Optimized Collapsed Ring (OCR) Heuristic

Optimized Collapsed Ring (OCR) heuristic has been proposed in [10] to protect multicast sessions against any fiber cut in the network. OCR has primary and backup paths for a given request. In order to find the primary path, OCR starts from the source node and finds the closest destination node using shortest path algorithm and removes all the links along the path. Then it resets the source node to be the previously used (last) destination node and finds the shortest path to next closest destination node. Similarly all the destination nodes are included in the primary path. For backup path, OCR finds the shortest path from first source node to final destination node thereby creating a cycle with set of destination nodes and source node. This heuristic algorithm is used to compare the performance of ECBRA in terms of cycle length and percent blocking.

Chapter 6 Numerical Analysis of CBRA, 2-Degree CBRA and ECBRA

6.1 Graph Model Description

To evaluate our algorithms, we 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. We add rest of the links in the graph randomly with probability given by Equation 1 following the Waxman model. In Equation 1, *s* and *d* are two nodes; D(s, d) is the distance between the nodes computed by using their coordinates in the grid, and α and β are two parameters that can be adjusted to control the number of links and their lengths in terms of distance of nodes on the Hamiltonian cycle.

$$P(s,d) = \beta e^{\frac{-D(s,d)}{L\alpha}} \qquad (1)$$

Since we start with a Hamiltonian cycle in all the random graphs, a solution always exists to any request. This characteristic of random graph helps us evaluating the performance of our algorithm. Since a cycle solution always exists, the blocking cases are due to lack of resources or failure of our algorithm.

6.2 Experimental Results for Static Traffic

We explored the performance of all three variations of CBRA in terms of blocking and cycle length on random graphs with various link densities. We ran the simulation on three different random graphs with 100 samples of each type for 10,000 (10k) requests. These three random graph types have different average number of links, 120, 360 and 720 links, respectively. All the random graphs have 60 nodes. Each request consists of 10 S-nodes. Figure 6-1 shows the average cycle length found by three variations of CBRA. We note that there is minimal impact on cycle length caused by enhancements on CBRA for all random graph types. The blocking percentage for random graphs with 120 and 360 links is shown in Figure 6-2. ECBRA has lower percent blocking than other two variations of CBRA for random graph with 120 links. Hence, ECBRA is preferred over other variations of CBRA. In the next section, ECBRA is used to compare the performance with related cycle finding heuristics like Enumeration, OCR for dynamic traffic.

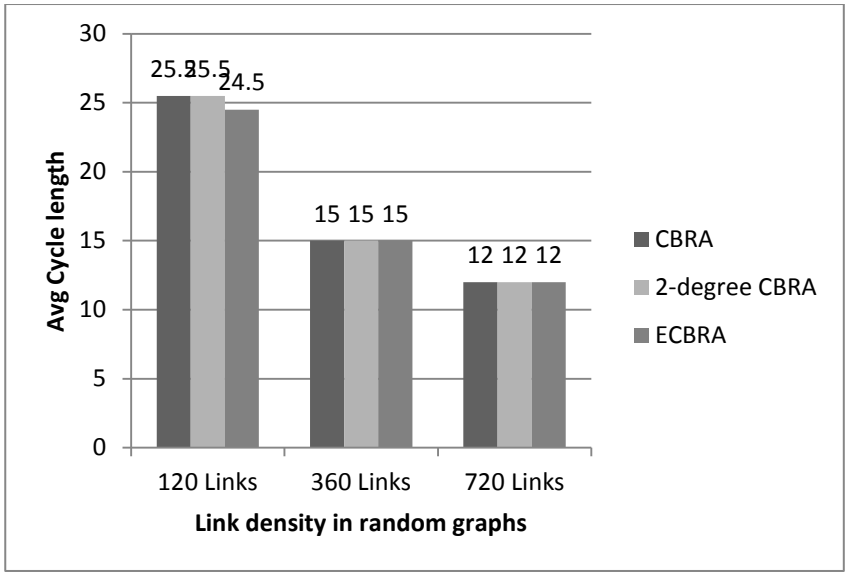

Figure 6-1: Comparison of Average cycle length for random graphs

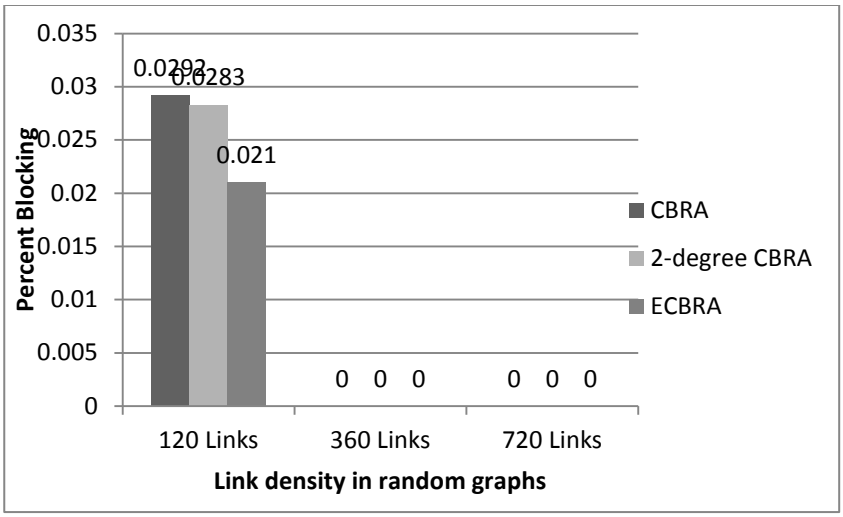

Figure 6-2: Comparison of percent blocking for random graphs

6.3 Results for Dynamic Traffic

6.3.1 Traffic Model

In this section, we assume that traffic arrives and leaves the network for a cycle requests in an optical network deploying wavelength division multiplexing. It is assumed that each link consists of a fiber with multiple wavelengths. Each wavelength is assumed to have a capacity of OC-192. Requests are assumed to be of three different capacities OC-1, OC-3, and OC-12 with equal probability. The request arrival is a Poisson process with rate r and request holding time is exponential with average time of 1. Our Traffic model assumes each link has a capacity of *n* lambdas as specified in user input and the wavelength continuity constraint is required to be maintained for any requests that have to be routed on a network. For performance evaluation, we consider two wavelength selection schemes. In one scheme, called First Fit, we order wavelengths and find cycle in the first lambda that has sufficient free capacity. The other wavelength selection scheme is that we compute the cycle length for each lambda and select the one with shortest cycle length. A tie is broken by selecting the least utilized wavelength.

6.3.2 Comparison with Enumeration Heuristic

We compare the performance of ECBRA heuristic with Enumeration cycle algorithm based on two evaluation metrics, average cycle length and blocking percentage of requests. The experiment is performed on three well-known network topologies, i.e. Arpanet, NSFNET and Cost239, for dynamic traffic arriving at different rates. We assume these networks have three wavelength capacity (3* OC-192) on all links. Since we are considering traffic grooming and the size of each request is far less than a wavelength capacity, each link can accommodate many requests till link's capacity is exhausted. We analyze the performance of the heuristics in two wavelength selection schemes, First Fit (FF) and Shortest Cycle (SC). Each multipoint request size is at most 25% of total nodes and the different arrival rates are simulated with input of 100,000 requests.

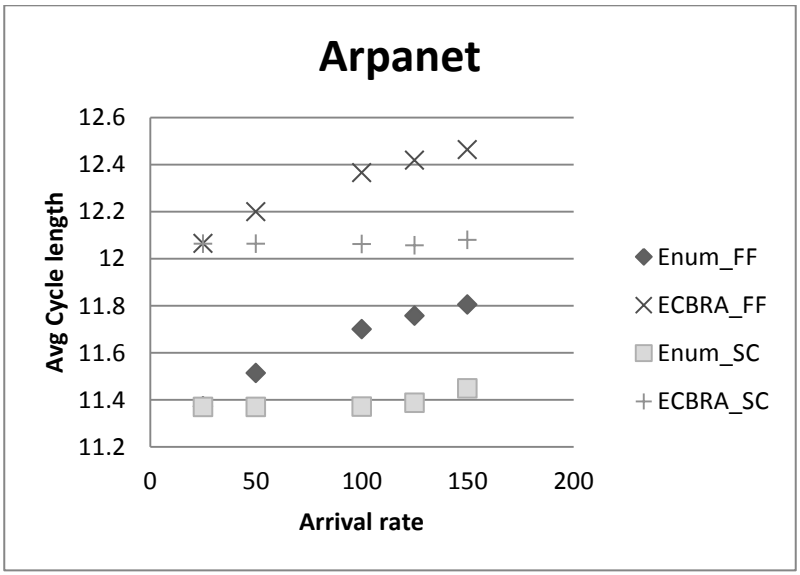

Figure 6-3: Comparison of Enumeration and ECBRA heuristic algorithms on Arpanet network

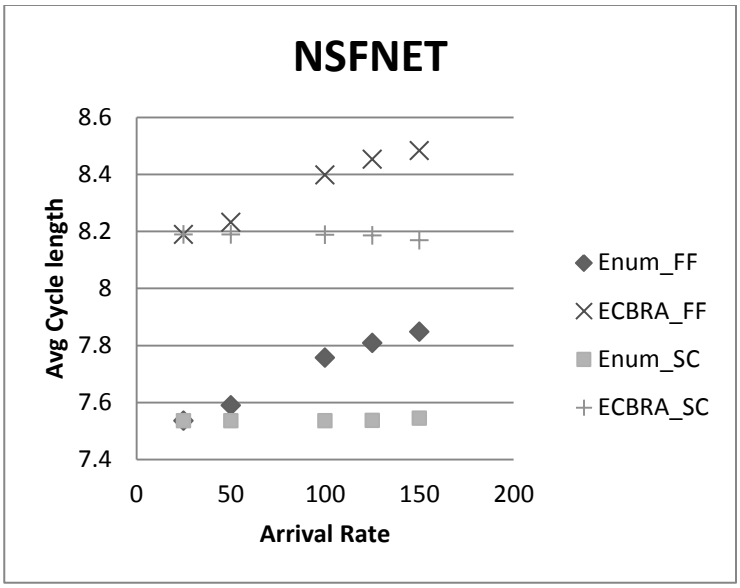

Figure 6-4: Comparison of Enumeration and ECBRA heuristic algorithms on NSFNET network

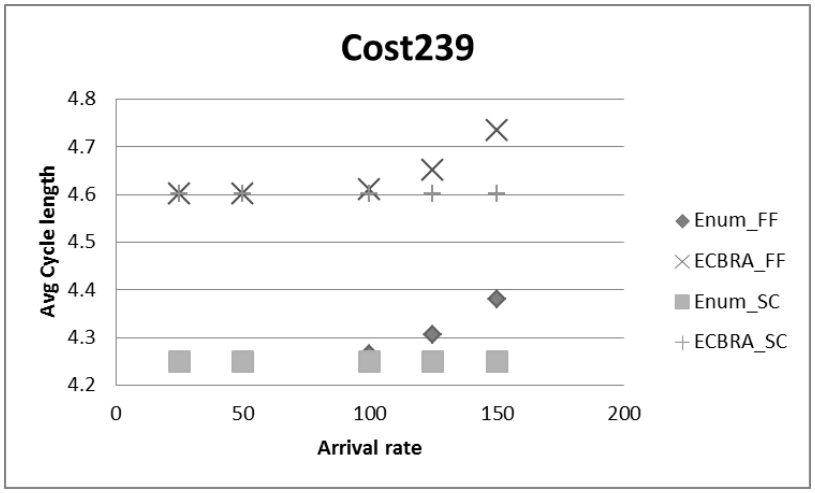

Figure 6-5: Comparison of Enumeration and ECBRA heuristic algorithms on Cost239 network

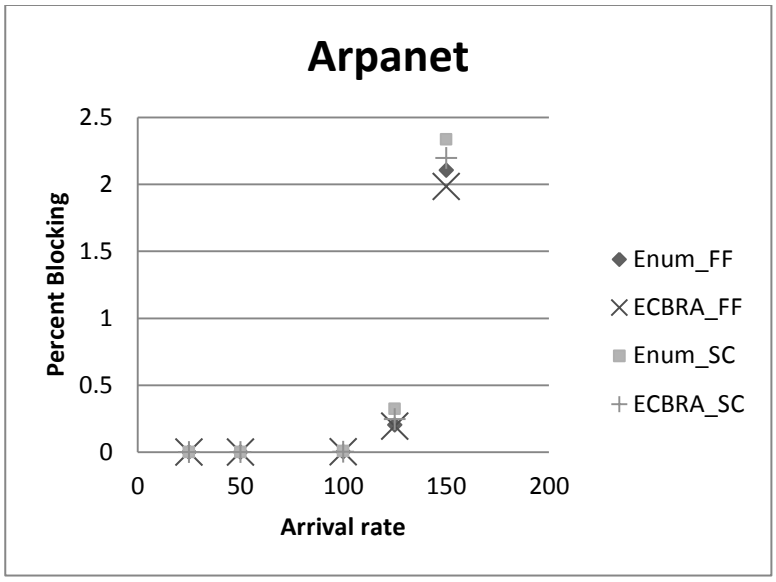

Figure 6-6: Blocking percentage of Enumeration and ECBRA heuristic algorithms on Arpanet network

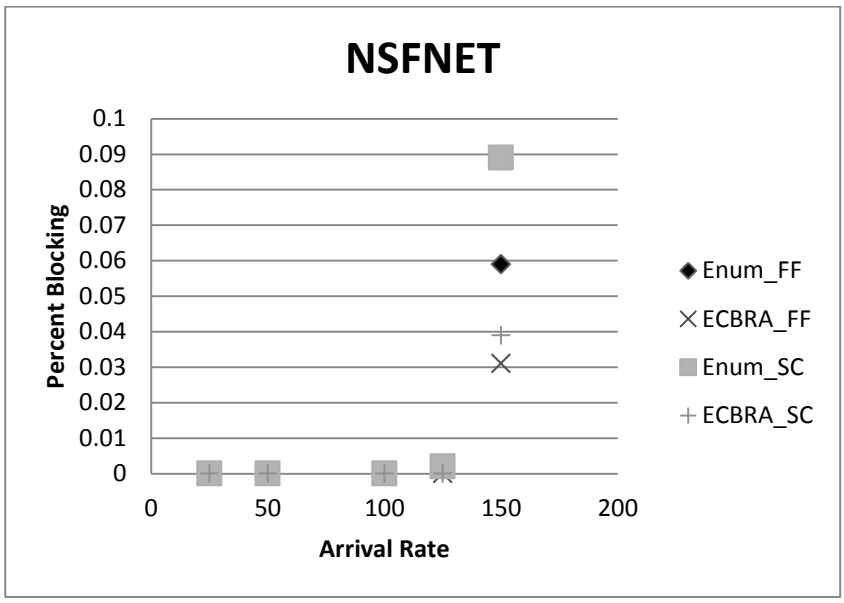

Figure 6-7: Blocking percentage of Enumeration and ECBRA heuristic algorithms on NSFNET network

Figure 6-3 shows the average cycle length found by ECBRA and Enumeration algorithm for Arpanet. The X-axis represents the requests arrival rate and Y-axis denotes the average cycle

length. The average cycle length found by ECBRA is bit higher than that of Enumeration for both wavelength selection schemes. It should be noted that the difference in length between these two heuristics is less than one node, which suggests that the scalable ECBRA performance is very close to that of un-scalable simple cycle generating Enumeration algorithm. The cycles generated with specific to wavelength selection schemes are as expected where First Fit (FF) has higher length than shorter cycle (SC) scheme. The average cycle length comparison of ECBRA and Enumeration algorithm for NSFNET and Cost239 networks are shown in Figure 6-4 and Figure 6-5 in which the behavior of ECBRA is similar to that of Arpanet.

The blocking percentage of requests for Arpanet using ECBRA and Enumeration algorithms is shown in Figure 6-6. It can be seen that blocking percentage is zero for arrival rates up to 100 for both the heuristics. When the arrival rate increases further, network could get overloaded and blocks some requests due to lack of resources. At high arrival rates the blocking percentage of Enumeration algorithm is higher than that of ECBRA; this is because ECBRA tends to find non-simple cycles as well, which increases the probability of finding the cycle. We can see similar behavior of Enumeration and ECBRA algorithms for NSFNET network in Figure 6-7. Considering the performance of ECBRA in terms of Average cycle length is close to that simple cycle and the blocking percentage lower than un-scalable Enumeration algorithm at high arrival rates, ECBRA is a preferable algorithm to find cycles when compared to Enumeration algorithm.

6.3.3 Comparison with OCR Heuristic

We depict the comparison of OCR heuristic with ECBRA in terms of average cycle length in NSFNET and Cost239 networks in Figures 6-8 and 6-9, respectively. We can see that

cycle length found by OCR is independent of request arrival rates and lower than that of ECBRA. But the blocking percentage of OCR heuristic turns out to be very high. We note that more than 50% of the requests are blocked by OCR heuristic at all the arrival rates used in the experiment. This makes OCR heuristic unreliable and not preferred in finding cycles through a given set of nodes when compared to ECBRA.

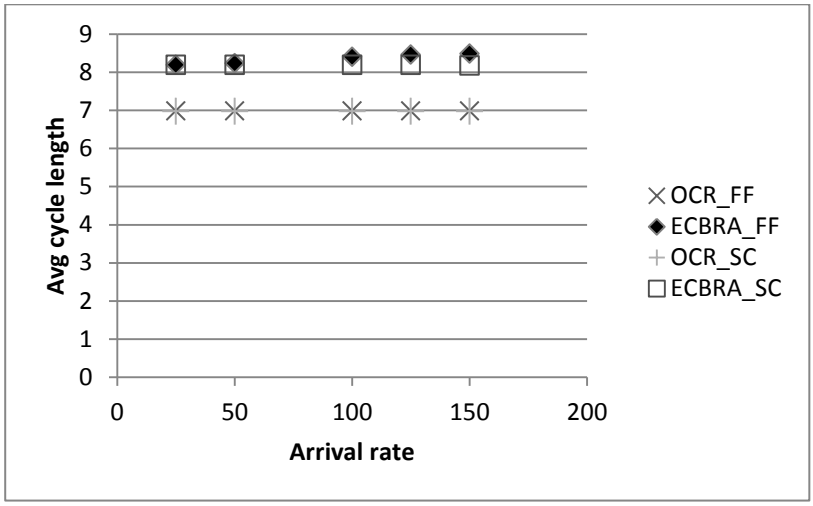

Figure 6-8: Comparison of OCR and ECBRA heuristic algorithms on NSFNET network

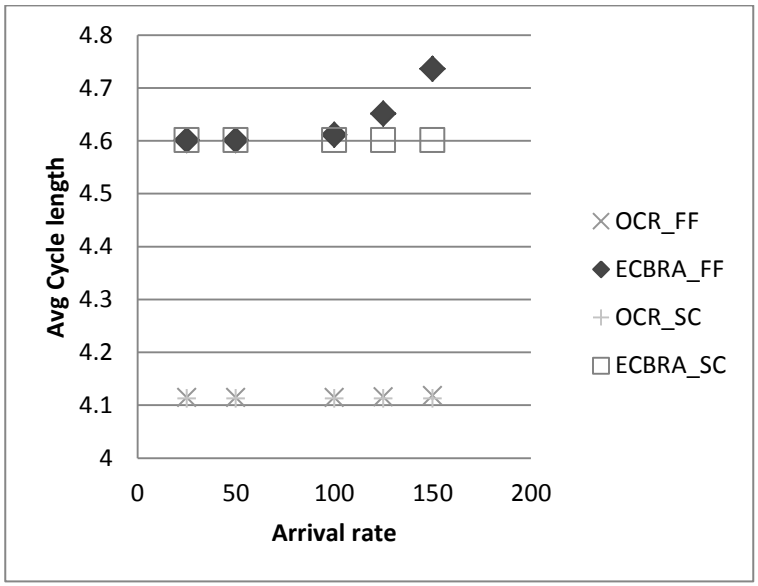

Figure 6-9: Comparison of OCR and ECBRA heuristic algorithms on Cost239 network

Chapter 7 Impairment Aware Cycle Based Routing Algorithm (IACBRA)

7.1 Network Model

In real networks, the links connecting two nodes will have impairment attributes at varying levels, which sometimes negatively influence the network performance. It is imperative that the link impairments should be considered while finding a cycle to route the traffic through specified must-include nodes. In this paper, we associate the impairment in a link to an integral value or called cost such that, higher the level of impairment, the cost associated with that link will also be higher. Hence we develop an algorithm to find a possibly least impaired or least cost cycle passing through given set of nodes. The network is modeled similar to that of ECBRA algorithm, using a graph model G (V, E, L (e)) and set S containing S-nodes that is given as part of multipoint request. L (e) represents the cost of link e, where e ϵ E. The only difference between CBRA and Impairment Aware Algorithm network model is the way costs are associated with the network. In Impairment Aware algorithm, the cost is associated with links. This implies that if link cost L(e) is equal to one for all the links in the network then IACBRA turns into CBRA algorithm.

7.2 Impairment Aware Cycle Based Routing problem (IACBRA)

Given a graph G (V, E, L (e)), S a subset of nodes in *V*, *k* be an integer and L(e)is the cost associated with the links,

Is there a simple or non-simple cyclic route p (V', E'), where V' is the set of nodes in p and E' is the set of edges in p, such that $S \subseteq V'$ *and Total Edge Cost (p),* $\sum_{e \epsilon E'} L(e) <= k$*?*

7.3 IACBRA Description

IACBRA is used to find a near optimal route for a multipoint request in a network with varying link quality. We use the term "*S-Optimal Least Cost (SOLC) path*" throughout the discussion to refer to the least link cost path between s_i and s_j (s_i, $s_j \in S$) with maximum number of *S-nodes* on it. IACBRA implementation is very similar to that of CBRA except Modified Bread-First Search (MBFS). Modified Bread-First Search (MBFS) module used in CBRA is changed to find "*S-Optimal Least Cost (SOLC) path"* in all the steps of IACBRA.

1. In Step-1, IACBRA finds the *Initial-SOLC* path.
2. IACBRA searches for *Disjoint-SOLC* path between end nodes of *Initial-SOLC* path in Step-II. If Step-II is successful, a cycle is formed passing through few or all *S-nodes.* If all the *S-nodes* are included, then the algorithm terminates successfully thereby skips Step-III.
3. In Step-III, the S-nodes which are not included in steps I and II are inserted by replacing the existing segments in the cycle. If all *S-nodes* are included in the cycle then IACBRA terminates successfully otherwise it reports failure.

Since the steps involved in finding the cycle for a multipoint request in IACBRA is same as that of CBRA, we omitted the description of the steps.

7.4 Link Cost Definition

In most of the practical networks, the links will not be all of same quality. The metric for measuring the quality of a link can be end-to-end delay, bandwidth availability, accessibility to

end nodes of a link, bit error rate and so on. In this paper, we have considered the accessibility to end nodes of a link to evaluate the cost of that link and also random allocation of link cost. In our simulation the formula used to calculate the cost of a link connecting nodes u and v is

$$\text{Link cost of e (u, v)} = degree\ (u) + degree\ (v)$$

Degree (u or v) represents the node degree. Link cost represents the degree of the end nodes. We have chosen link cost directly proportional to degree of end nodes because results in CBRA suggested that the nodes left out in blocked requests are nodes with small degree. Hence we force our algorithm to take low cost path and thereby include as many lower degree nodes as possible. Figure 7-1 shows Arpanet network with link cost mentioned on top of all edges.

The nodes in darkly filled circles belong to the set S, which should be included in the cycle found by our algorithm.

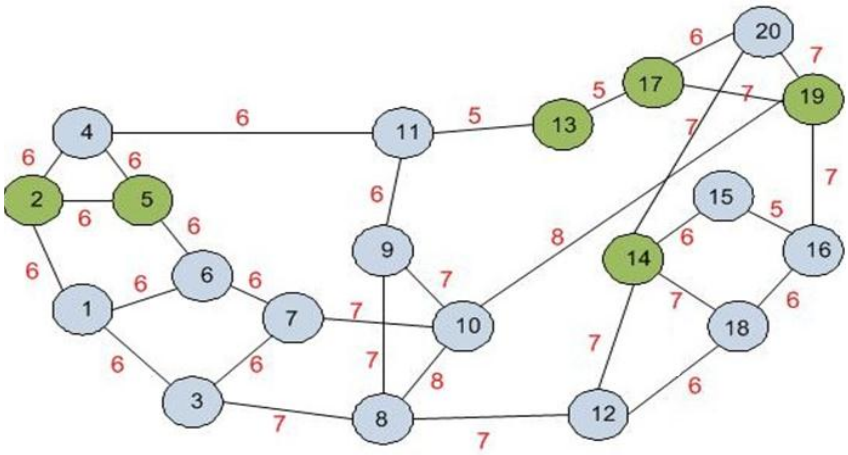

Figure 7-1: Arpanet network with costs mentioned on top of all links

7.5 Modified Bread First Search (IACBRA)

The *S-Optimal Least Cost (SOLC) path* is found by Modified BFS algorithm. The algorithm begins at a node $s_i \in S$. MBFS traverses the entire graph starting with root node s_i in a breadth first search manner but with the goal to create a least link-cost tree. The Least link-cost

tree is defined as tree with lowest total link cost path from root node to any child nodes. This tree is created in the following manner. Let's say the root node s_i is at Level 0. This root node is connected to all of its neighboring nodes. These neighboring nodes are located at Level 1. We connect each of the nodes in Level 1 to all its remaining neighboring nodes, which are not already included in upper level of the tree. We ignore the edges connecting to the neighboring nodes which are present on the same level. These newly connected neighboring nodes of Level 1 nodes are located at Level 2. The rest of the least link-cost tree is constructed in similar fashion. The child nodes with more than one parent are resolved by choosing the parent node, which has lowest total link cost on its path to root node. If there are more than one parent nodes which lead to root node at the same link cost, then the tie is broken by choosing the parent node with more number of *S-nodes* on its path.

For Arpanet network with link weights, shown in Figure 7-1, MBFS creates a least link cost tree as depicted in Figure 7.2. This example network has set *S* with following nodes {2, 5, 13, 14, 17, and 19}. The *S-nodes* are shown with circles in Figure 7.2. In this example, MBFS create a least link cost tree with root node at 5 and returns the destination node which has maximum number of S-nodes in its least link path to root.

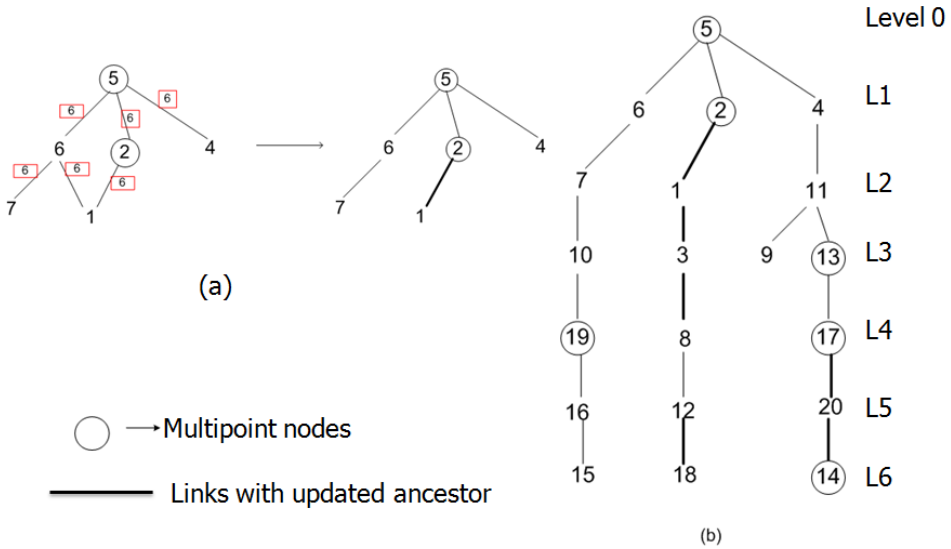

Figure 7-2: Least link-cost tree created by MBFS of IACBRA

Part (a) of Figure 7-2 shows how least link cost tree of Arpanet is constructed. While visiting each node, if the parent of its child node is not already set then MBFS sets the currently visited node as parent for all of such child nodes. If the parent is already set for a child node then MBFS chooses its parent node which has with least cost path to the root node which is shown in Part (a) of Figure 7-2. Initially node 1 would have its parent set to node 6. Later when the possible children of node 2 are considered, the parent of node 1 would be changed to node 2 since its path to root has same total link cost and has more S-nodes. This is indicated by the link between nodes 1 and 6 being crossed out. Similarly other links have been crossed out. Figure 7-2 (b) shows the final least link cost tree with root set to node 5 created for Step-1. Here MBFS returns the node 14 and its path to root node, because it has SOLC path.

Chapter 8 Numerical analysis of ECBRA and IACBRA

8.1 Experimental results for Static Traffic

We mentioned in earlier section that IACBRA is exactly same as ECBRA for graph G (V, E, L (e)) where L (e) = 1. Hence we verified the results generated by both algorithms with L (e) = 1, for same set of multipoint requests and found that they match for every request. In order to determine the performance improvement provided by IACBRA for a network with link costs (> 1), we executed CBRA algorithm on the same network for same set of multipoint requests under static traffic scenario. Then we calculated the total link cost for the each of the cycles generated by both the algorithms, to see the quantity of cost savings provided by IACBRA. We ran ECBRA and IACBRA on Arpanet network, as shown in Figure 7.1, for the same set of 100,000 multipoint requests. Each request is of size 20-25% of total nodes in the network. The difference between total link cost for each of 100,000 cycles generated by ECBRA and IACBRA is shown in Figure 8.1. This difference between ECBRA and IACBRA has been sorted in ascending order, in order to give vivid picture of IACBRA performance.

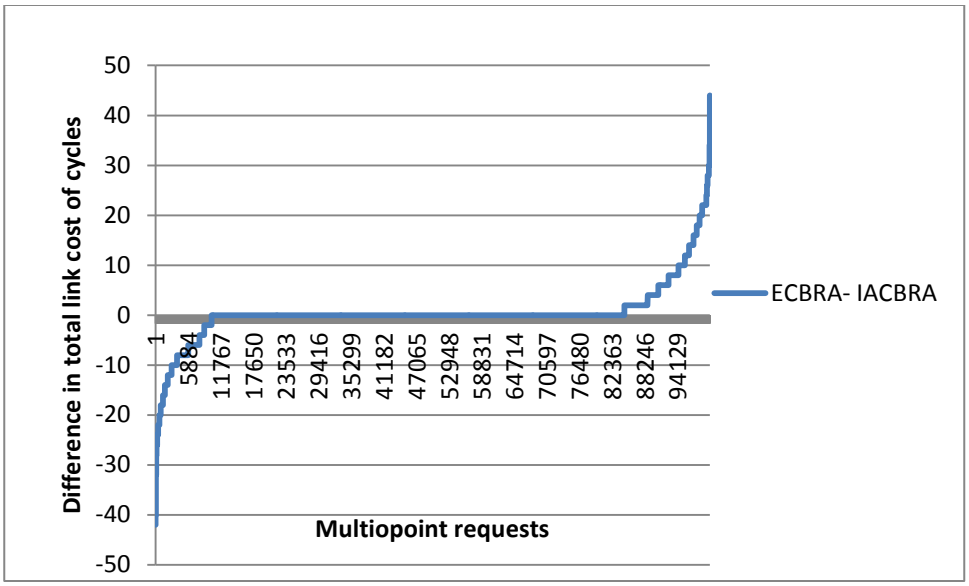

Figure 8-1: Total link cost difference between ECBRA and IACBRA

Figure 8-1 shows around 90% of requests have lower total link cost for the cycle generated by IACBRA algorithm. Also the average total link cost of cycles generated by IACBRA is one less than that of ECBRA. Although this might look to be too small, but when we see the total range of link cost in the network, it gives better perspective. The different values of link cost used in the network are 5, 6, 7, and 8. Hence, given the range of link cost in the network to be mere 3, the improvement given by IACBRA algorithm is significant. We have shown that if the range of link cost in the network is widened, the savings in average total link cost generated by IACBRA increases in next section. Further analysis on the requests for which ECBRA algorithm generated cycles with lesser total link cost, reveals that it is because of the nature of network and the way our algorithm functions. For all the requests, IACBRA finds the path with low total link cost in Step-1, when compared to that produced by ECBRA. But as the algorithm proceeds, IACBRA have to add few extra nodes or take high cost paths with many nodes due to the earlier optimized decision in Step-1 or Step-2. This could be solved by different complex heuristic which takes into consideration of link cost of all the steps but polynomial time execution is doubtful.

8.2 Simulation Results for Dynamic Traffic

The experimental setup for analyzing the performance of IACBRA in dynamic scenario is as follows. The networks have wavelength continuity constraint, links are assumed to have three lambdas, wavelength selection criteria is based on first fit wavelength and incoming requests are with size of 20-25% of total nodes. The request arrival is a Poisson process with rate r and request holding time is exponential in time. The request and link capacity are analogous to OC-3 and 3 * OC-192 respectively.

In order to compare IACBRA performance with ECBRA in dynamic traffic we ran the algorithms for different request arrival rates and compared the average total link cost of the cycles. We considered three different scenarios which differ from the way the costs are associated with the links. In case one, we assigned the costs for links based on the formula mentioned in earlier Link cost section. Figure 8-2 shows the results of ECBRA and IACBRA algorithms for Arpanet network, where the difference between ECBRA and IACBRA algorithms' average total link cost of a cycle is less than 1. In case two, we verified whether the performance improvement by IACBRA increases with widened range of link cost, by arbitrarily assigning low link cost for the links connecting two degree nodes and high link cost for the links connecting three/four degree nodes. The example network for the case two is shown in Figure 8-3. We can see in Figure 8-4 that the difference between IACBRA and ECBRA's performance is high for wider range of link costs in the network. So our IACBRA gain scales up with wider range of link cost used in the network. In case three, we randomly assign the link cost irrespective of the degree of nodes being connected by a link. Figure 8-5 is an example network for the case three. The results for case three is shown in Figure 8-6, where IACBRA finds the cycles with lower total costs when compared to that found by ECBRA algorithm.

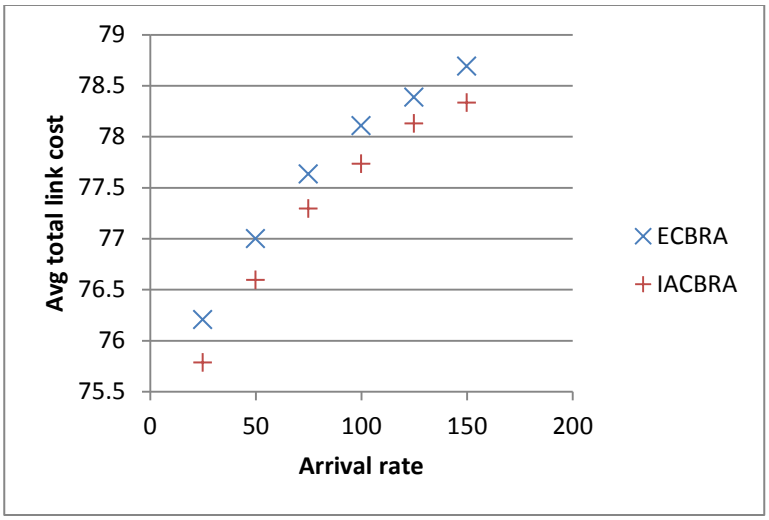

Figure 8-2: Comparison of ECBRA and IACBRA algorithms on Arpanet for case one

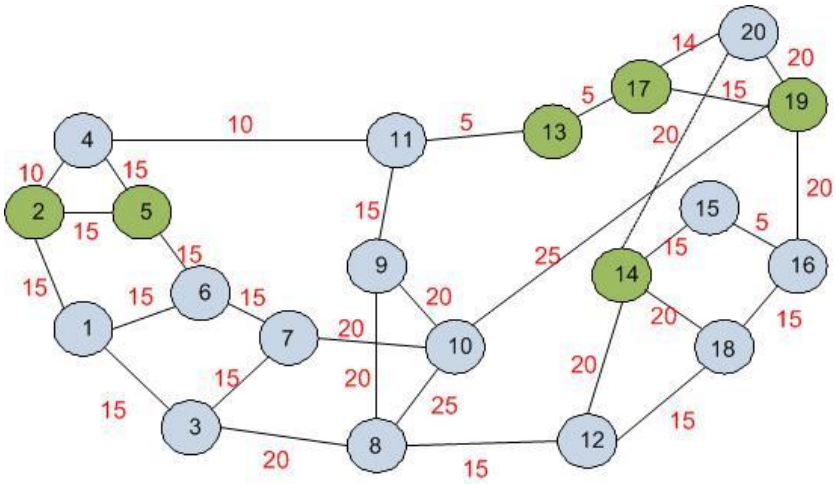

Figure 8-3: Arpanet network with wider range of link costs

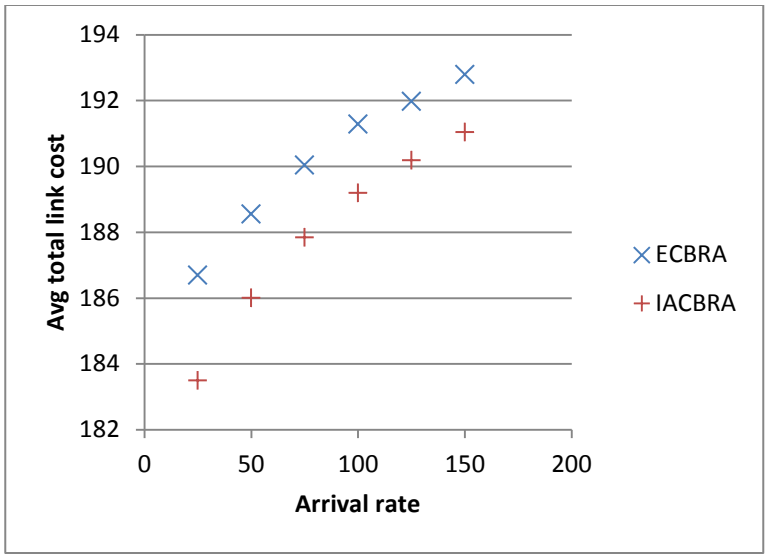

Figure 8-4: Comparison of ECBRA and IACBRA algorithms on Arpanet for case two

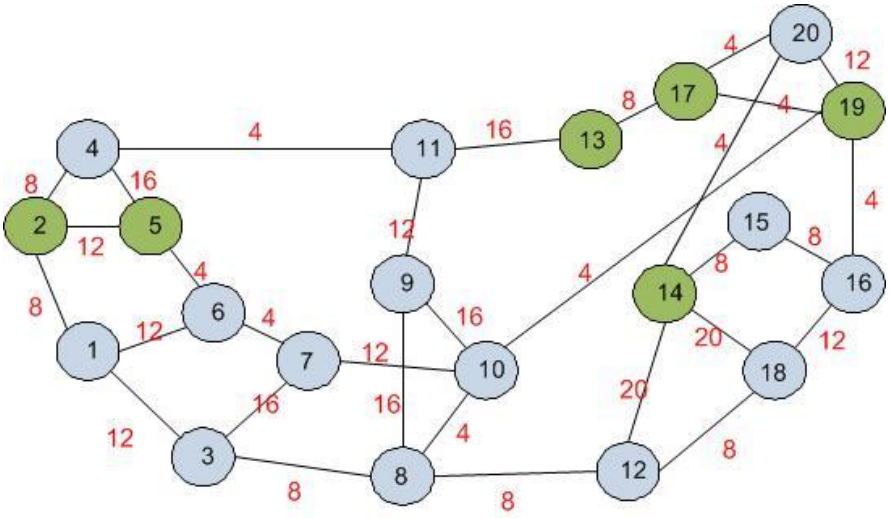

Figure 8-5: Arpanet network with randomly assigned link costs

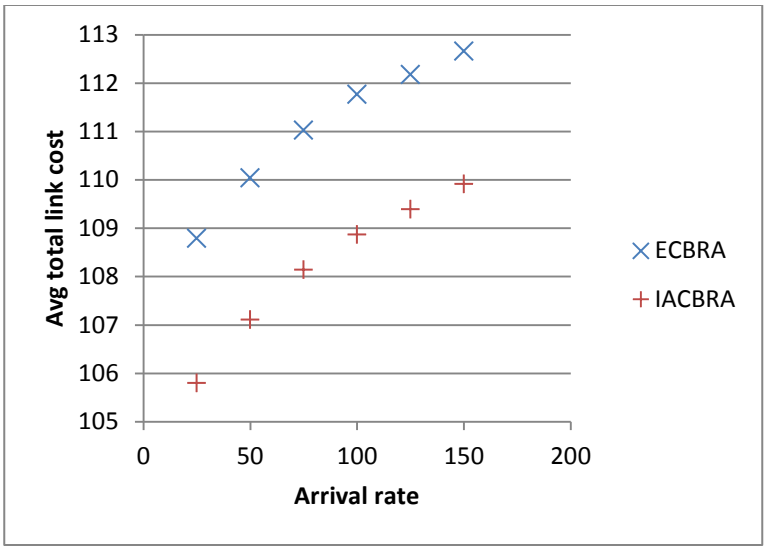

Figure 8-6: Comparison of ECBRA and IACBRA algorithms on Arpanet for case three

We executed our IACBRA algorithm for Arpanet and NSFNET under dynamic traffic, to compare the cycle length generated by IACBRA with that of CBRA. Figure 8-7 shows that the average cycle length generated by IACBRA on Arpanet is very close to CBRA results depicted in Figure 6-3. We can see the similar comparison of average cycle length on NSFNET in Figures 8-8 and 6-4. This suggests that IACBRA improved the total link cost of cycles without compensating on number of nodes in the cycle.

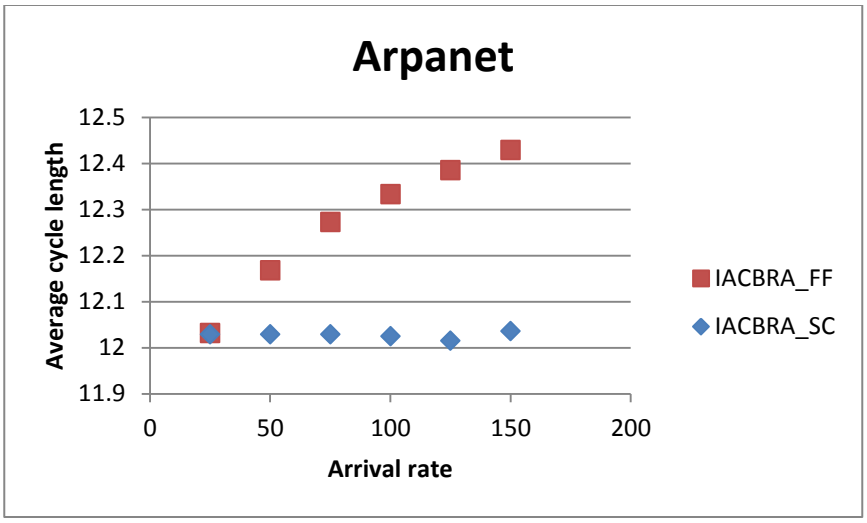

Figure 8-7: Average cycle length generated by IACBRA on Arpanet

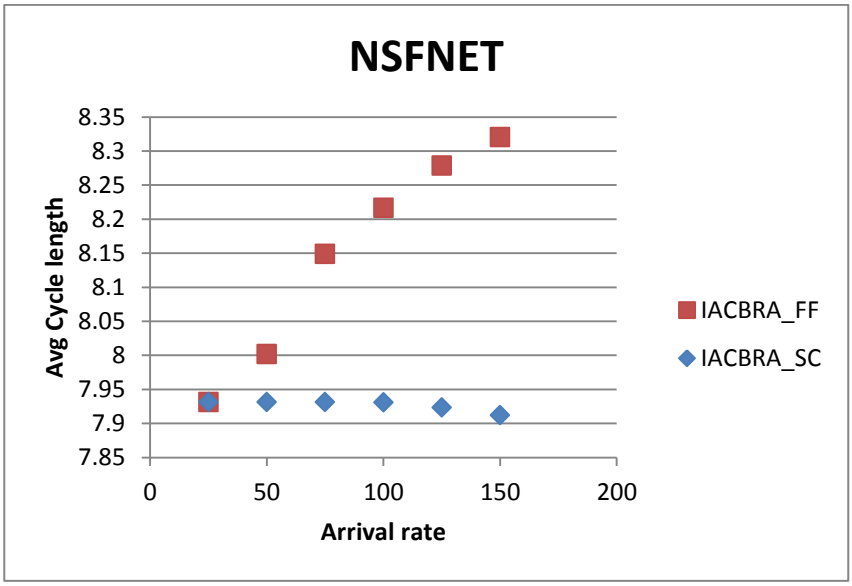

Figure 8-8: Average cycle length generated by IACBRA on NSFNET

We also found that there is not much difference in blocking percentage of requests between IACBRA and CBRA algorithms.

Conclusion and Future work

In this paper, we proposed a novel, fault tolerant cost-oriented cycle based routing algorithm for multipoint communication in core networks. Our algorithm can also be used to find cycle with must-include nodes in cases like reliable point to point communication, multicast and preconfigured protection structures. We extended our shortest cycle algorithm (ECBRA) to least-cost cycle algorithm (IACBRA), which finds many applications in Metropolitan Core networks (SONET/DWDM). Our simulations show that ECBRA has higher success rate when compared to other existing heuristic algorithms. IACBRA improves the total link cost of cycle without increasing the cycle length or blocking percentage, in both the cases where link costs are chosen randomly and the case where the link costs are assigned based on degree of nodes connected by that link.

This research is primarily focused on multipoint communication in core networks. Our simulation traffic model is based on wavelength continuity constraint which could be expanded to more sophisticated network with functionalities like lambda switching and aggregation. This algorithm could also be extended to real-time multipoint communication like video conferencing, online games and other distributed applications where the nodes arrive and leave the network dynamically for a given request. Our algorithm is only cost-oriented, so to apply for real-time applications the algorithm must also be implemented with end-to-end delay constraint between source-destination pairs.

BIBLIOGRPAHY

[1] D. Bertsekas and R. G. Gallager, "Data Networks", 2nd ed. EnglewoodCliffs, NJ: Prentice-Hall, 1992.

[2] H. Vardhan, S. Billenahalli, H. Wanjun, M. Razo, A. Sivasankaran, T. Limin T, P. Monti, M. Tacca, A. Fumagalli, "Finding a simple path with multiple must-include nodes," Modeling, Analysis & Simulation of Computer and Telecommunication Systems, 2009, MASCOTS '09. IEEE International Symposium on, vol., no., pp.1-3, 21-23 Sept. 2009.

[3] S. Ramanathan, "An algorithm for multicast tree generation in networks with asymmetric links," in Proc. IEEE INFOCOM'96, pp.337–344.

[4] L. Kyou, G. Markowsky, and L. Berman, "A fast algorithm for Steinertrees," Acta Info., vol. 15, no. 2, pp. 141–145, 1981.

[5] V. Rayward-Smith, "The computation of nearly minimal Steiner treesin graphs," Int. J. Math. Education Sci. Tech., vol. 14, no. 1, pp. 15–23,Jan./Feb. 1983.

[6] David Lastine, Suresh Sankaran and Arun K Somani, "A Fault-Tolerant Multipoint Cycle Routing Algorithm (MCRA) (An Invited Paper)", BROADNETS, 2010.

[7] B. M. Waxman, "Routing of multipoint connections," IEEE Journal on Selected Areas in Communications, Vol. 6, No. 9, 1988, pp. 1617-1622.

[8] R. Karp, R.M, J.Thatcher, "Reducibility among Combinatorial Problems, Complexity of Computer Computations", Plenum Press, pp. 85-103, 1972.

[9] Ayse Karaman and Hossam Hassanein, "DCMC – Delay-Constrained Multipoint Communication with Multiple Sources", Proceedings of the Eighth IEEE International Symposium on Computers and Communication (ISCC'03), pp. 467—472.

.

[10] A. Khalil, A. Hadjiantonis, G. Ellinas, M.A. Ali, "Pre-planned multicast protection approaches in WDM mesh networks", Optical Communication, 2005, 31st European Conference, pp. 25- 26.

[11] Arun K. Somani, "A Modified Breadth First Search Algorithm to include required nodes," personal communication, May 2010.