

Disjoint Multipath Routing Using Colored Trees

Srinivasan Ramasubramanian, Harish Krishnamoorthy, and Marwan Krunz
 Department of Electrical and Computer Engineering
 University of Arizona, Tucson, AZ 85721

Abstract—Multipath routing (MPR) is an effective strategy to achieve robustness, load balancing, congestion reduction, and increased throughput in computer networks. Disjoint multipath routing (DMPR) requires the multiple paths to be link- or node-disjoint. Both MPR and DMPR poses significant challenges in terms of obtaining loop-free multiple (disjoint) paths and effectively forwarding the data over the multiple paths, the latter being particularly significant in IP datagram networks.

This paper develops a two-disjoint multipath routing strategy using colored trees. Two trees, red and blue, that are rooted at a designated node called the drain are formed. The paths from a given source to the drain on the two trees are link- or node-disjoint. Such an approach requires every node to maintain only two preferred neighbors for each destination, one on each tree. This paper (1) formulates the problem of colored-trees construction as an integer linear program (ILP); and (2) develops the first distributed algorithm to construct the colored trees using only local information. We demonstrate the effectiveness of the distributed algorithm by evaluating it on grid and random topologies and comparing to the optimal obtained by solving the ILP.

I. INTRODUCTION

Multipath routing (MPR) is an effective strategy to achieve robustness [1], load balancing [2], congestion reduction [3], low power consumption [4], and increased throughput. It operates by transmitting data over multiple paths. In general, the multiple paths from a source to a destination may have common links (or nodes) as long as the shared links (or nodes) have sufficient resources. To improve the transmission reliability and avoid shared-link (or node) failures, the multiple paths can be selected to be link- or node-disjoint. In this case, the MPR approach is referred to as *disjoint multipath routing* (DMPR). DMPR provides better robustness compared to the generic MPR. However, it may be inefficient with respect to other metrics such as the overall energy consumption [4].

DMPR has been extensively studied in the context of wired networks [5], [6], where the multiple paths are often employed for failure resiliency purposes. Only one of the paths, referred to as the primary path, is used at any instant. Upon a failure, the connection is rerouted over a backup path. If the backup path is the same for any link (or node) failure that affects the primary path, then the primary and backup paths must be link- (or node-) disjoint¹. In applications such as transmission of multiple description encoded video streaming, the two link-disjoint paths are used simultaneously. Two independently encoded video streams are transmitted along

two link-disjoint paths [7]. If multiple paths are employed for increased throughput, then the data may be split over multiple paths.

A. Motivation

Implementation of generic MPR and DMPR poses two main challenges. The first is related to the computation of loop-free multiple paths. Several centralized algorithms (or equivalently those that assume a global network knowledge) have been proposed for the DMPR problem in the context of failure resiliency in wired connection-oriented networks. Because of their centralized nature, these algorithms are not directly applicable to large-scale (wired or wireless) networks. In such cases, a distributed solution that relies only on local information is preferred. Distributed multipath routing algorithms in the literature are developed purely in the context of wireless networks. MPR approaches based on Dynamic Source Routing (DSR) [8], [9], [10] require the destination to select maximally disjoint paths among the received route requests. MPR approaches based on AODV routing [11], [12], [13], [14], [15] do not guarantee finding disjoint paths. The only well-known generic multipath routing employed in the wired datagram network is the OSPF algorithm, where the choice of paths is limited to paths of equal cost.

The second challenge of implementing MPR (or DMPR) techniques is related to the forwarding of data over the multiple paths. In typical connection-oriented networks, the end-to-end path is clearly identified using, for example, connection identifiers or labels. In such networks, nodes maintain a routing table that specifies the output port for each label. Each path in the set of multiple paths requires a unique set of connection identifiers. Hence the size of the routing table is directly proportional to the number of multiple paths. In contrast, datagram networks rely on the destination address in the packet header for the purpose of forwarding packets over a single path. To implement MPR or DMPR techniques in such networks, every node must maintain a set of preferred neighbors to reach a destination, such that the paths are loop-free (and disjoint, if needed). Forwarding of packets to meet such constraints must be based on destination address and some “additional” information (e.g. source address). The intermediate nodes must be aware of this additional information or otherwise, it must be carried in every packet header. The choice of what additional information is used to achieve forwarding along multiple paths determines the overhead of the scheme.

Our work is motivated by the lack of any known distributed algorithm that employs only local information for computing

¹Multiple backup paths may be employed, where each backup path corresponds to a particular link failure in the primary path. Upon a failure, a connection is rerouted on one of the backup paths depending on the failure location. In such an approach, it is possible that no two multiple paths are disjoint with the primary path.

disjoint multiple paths for effectively forwarding packets in a datagram network. As a first step, this work focuses on transmitting data from every node in the network to a designated node called the *drain* over two link- or node-disjoint paths. Hence, the communications paradigm considered is of multi-point to a point. The approach developed for this scenario may be used to support all-to-all routing by independently considering every node as a drain node. To this end, three possible approaches for the DMPR problem are reviewed.

B. DMPR Approaches

DMPR approaches may be classified into three broad categories, depending on the information used for forwarding individual packets.

Explicit forwarding. In this approach, every node independently finds two link/node-disjoint paths to the drain using well-known disjoint-path algorithms [5]. Once the disjoint paths are obtained, the path information is embedded in every packet and is used by intermediate nodes to route individual packets. No routing tables are needed at intermediate nodes. MPR algorithms based on DSR (for mobile ad hoc networks) fall under this category. The embedding of path information in the packet header is a significant overhead that limits the application of this approach.

Source-based forwarding. The communication cost of explicit routing may be reduced by transferring the disjoint path information embedded in the packets to intermediate nodes along the paths. Every node then maintains a routing table and forwards a packet based on the destination and source addresses provided in the packet header. Figure 1 shows two node-disjoint paths computed by nodes A and B. Consider the situation at node C. This node receives packets from one incoming link (ℓ) and forwards them to one of two distinct outgoing links, based on the source address. The routing table at node C will have two entries for the drain, corresponding to the two sources.

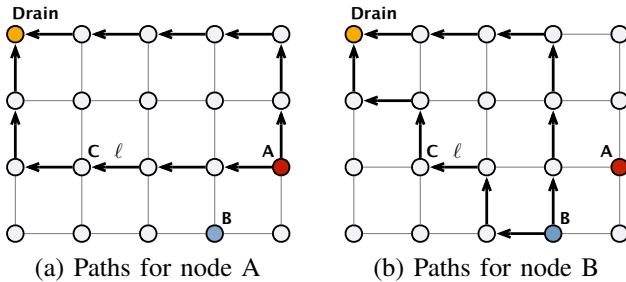


Fig. 1. Two node-disjoint paths from nodes A and B to the drain, computed independently by each node.

If the average sum of the lengths of the two paths from a source to the drain is \bar{H}_d , then the total number of routing entries in the network for a given drain is $(N - 1)\bar{H}_d$, where N denotes the number of nodes in the network. Note that a typical single-path routing would require only $(N - 1)$ entries for a particular drain. The sum of any two *disjoint paths* between a pair of nodes must be at least of length three, $(N - 1)\bar{H}_d \geq 3(N - 1)$. The value of \bar{H}_d may range anywhere from $O(1)$ for a fully connected network to $O(N)$ for a sparsely

connected network. Given that there may be $O(N)$ entries for each drain, in the worst-case, routing a packet based on both destination and source addresses will increase the table lookup time significantly.

Colored trees. To reduce the routing-table overhead, hence reduce lookup time, this paper develops a novel multipath routing strategy called *colored trees*. Every node in the network has two preferred neighbors to the drain: *red* and *blue*. A packet transmitted from a source is marked with one of the two colors. An intermediate node that receives the packet forwards it to its preferred neighbor based on the color of the packet. Thus, the routing table at a node has only two entries (for every drain node). The network may be viewed as two trees (red and blue) that are rooted at the drain, where the paths on these trees are directed towards the drain. The two trees have the property that the two paths from a given source to the drain on the two trees are link/node-disjoint. The number of routing table entries for a drain in a network of N nodes is $2(N - 1)$. Compared with single-path (source-based) forwarding, the colored tree approach reduces the routing table by a factor of $\frac{\bar{H}_d}{2}$. Figure 2 shows the colored trees for the example network in Figure 1.

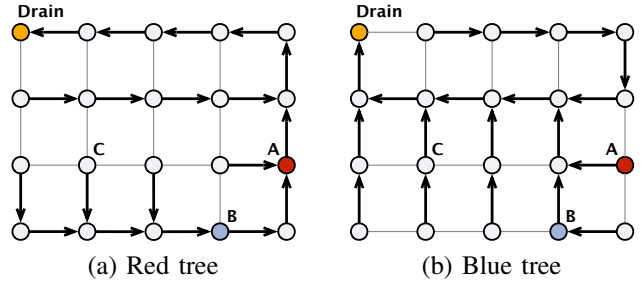


Fig. 2. Two colored trees (any two paths from a node to the drain on the two respective trees are node-disjoint).

C. Scope and Contribution

The goal of this paper is to develop a distributed algorithm for the CT-LD and CT-ND problems and to compare the performance of this algorithm with the optimal solution obtained by formulating the problem as an integer linear program. The contributions of this work are as follows:

- *ILP formulation:* To the best of our knowledge, this work is the first attempt at obtaining an optimal solution to the CT-LD and CT-ND problems. While centralized *heuristic* algorithms developed in the context of robust multicasting [16], [17], [18] may be adapted to this problem, there has not been any effort to compare the results to the optimal solution.
- *Distributed algorithm with local information:* This paper develops the first distributed algorithm to constructing two colored trees, employing only local (neighborhood) information. The algorithm is guaranteed to find a solution if one exists. In addition, a technique to reduce the average path length is also developed and evaluated.

The rest of the paper is organized as follows. Section II describes the network model, problem definition, and ILP

formulation. Section III discusses the related work in obtaining multitrees using heuristic approaches that were originally developed in the context of robust multicasting. This section also points out key observations in the centralized solutions that are essential for a distributed solution using local information only. Section IV develops the distributed algorithm for constructing two colored trees and techniques to arrange the neighbors in order to minimize the average path length from a source to the drain. Section V presents the performance of the distributed algorithm and compares it with the optimal solution. Conclusions and future work are presented in Section VI.

II. PROBLEM FORMULATION

Consider a network $\mathcal{G}(\mathcal{N}, \mathcal{L})$ composed of a set of nodes \mathcal{N} and a set of links \mathcal{L} . The links are assumed to be bi-directional. The terminology of *arc* is used to refer to a directed link between two nodes. An arc from node i to j is represented as $i \rightarrow j$. For wireless networks, the set of links indicates that two nodes are in the receiving range of each other. Let C_{ij} refer to the cost of the arc $i \rightarrow j$. The cost is assumed to be symmetric, i.e., $C_{ij} = C_{ji}$.

Problem definition. Given a drain node $d \in \mathcal{N}$, the goal is to construct two colored trees \mathcal{R} and \mathcal{B} (referred to as the red and blue trees, respectively) rooted at d that minimize the average path length from a source to the drain such that the CT-LD and CD-ND versions of the problem satisfy the link-disjoint and node-disjoint path constraints, respectively. These constraints are stated as follows.

Let $\mathcal{P}_{sd}^{\mathcal{R}}$ and $\mathcal{P}_{sd}^{\mathcal{B}}$ denote the paths from a node s to the drain d on trees \mathcal{R} and \mathcal{B} , respectively.

Link-disjoint path constraint: If the path from s to d on the red tree traverses $i \rightarrow j$, then the path from s to d on the blue tree traverses neither $i \rightarrow j$ nor $j \rightarrow i$, i.e., $\forall s \in \mathcal{N} \setminus \{d\}$ and $\forall i, j \in \mathcal{N}$

$$i \rightarrow j \in \mathcal{P}_{sd}^{\mathcal{R}} \Rightarrow (i \rightarrow j \notin \mathcal{P}_{sd}^{\mathcal{B}}) \wedge (j \rightarrow i \notin \mathcal{P}_{sd}^{\mathcal{B}}).$$

Node-disjoint path constraint: If the path from s to d on the red tree traverses node i , then the path from s to d on the blue tree does not traverse node i , i.e., $\forall s \in \mathcal{N} \setminus \{d\}$ and $\forall i \in \mathcal{N} \setminus \{s, d\}$

$$i \in \mathcal{P}_{sd}^{\mathcal{R}} \Rightarrow (i \notin \mathcal{P}_{sd}^{\mathcal{B}}).$$

A network must be two-node-connected to obtain a solution to the CT-ND problem. Similarly, the network must be two-edge-connected in order to obtain a solution to the CT-LD problem [17].

ILP formulation. The ILP formulation for the construction of two colored trees is shown in Figure 3. The tree constraints T-1 and T-2 specify that every node must have exactly one parent on each tree. The flow constraints F-1 through F-4 specify that every node s sources one unit of traffic to the drain. Each link is assumed to have unit capacity in each direction, if it is present on a tree; otherwise, none. The link-disjoint path constraints LD-1 and LD-2 specify that the paths from node s to d on the two trees may not use a link in the same or opposite directions. Note that if every node is assumed to generate

traffic, then LD-1 also implies $\alpha_{ij} + \beta_{ij} \leq 1, \forall i, j \in \mathcal{N}$. Hence, the red and blue trees are arc-disjoint when every node sources traffic. The node disjoint path constraint ND specifies that the two paths from node s may not pass through the same node more than once. The ILP is solved with either the LD-1 and LD-2 constraints or the ND constraint to compute colored trees for link-disjoint or node-disjoint paths, respectively.

The ILP formulation helps in identifying an optimal solution for small networks. However, its application to large networks is impractical due to its prohibitive computational time. It can still serve as a benchmark for evaluating the effectiveness of heuristic approaches in reasonable-sized networks, thereby aiding in developing heuristic solutions for large-scale networks.

III. RELATED WORK

The construction of colored trees for wired networks was considered in the context of robust multicasting to withstand at most single-link failures. The trees constructed for multicasting are directed from the source towards multiple destinations. The techniques employed for multicasting may be applied to the problem at hand by simple arc reversal. Note that the routing table information maintained for multicast routing is quite different from that required when multiple nodes transmit to a single drain. Two centralized approaches have been proposed in the literature for multicast routing, one based on s - t numbering [16] and the other on generalized path augmentation [17], [18].

A. s - t Numbering

Given a graph with N nodes and an edge s - t in that graph, an s - t numbering is a numbering of the vertices of the graph such that the following two properties are obeyed:

- Node s has the number 1 and node t has the number N .
- Every node i other than s and t has a number v_i , $1 < v_i < N$, such that node i has at least one neighbor with a higher s - t number than v_i and at least one neighbor with lower s - t number than v_i .

Itai et al. [16] employed the s - t numbering technique to obtain a centralized solution to the CT-ND problem. The major drawback of the s - t numbering approach is that it is applicable only to networks that are two-node-connected. Hence, it is not possible to obtain a solution to the CT-LD problem using this approach.

B. Generalized Path Augmentation

The path augmentation algorithm starts by choosing an arbitrary directed cycle (d, v_1, \dots, v_k, d) in \mathcal{G} with at least three nodes ($k \geq 2$). If this cycle does not include all the nodes of \mathcal{G} , then a path that starts and ends on that cycle and that passes through at least one node not on the cycle is chosen for augmentation. The algorithm continues with path augmentation until all the nodes in the network are considered.

Medard et al. [17] developed a centralized algorithm that selects a cycle and successive paths at random. Xue et al. [18]

Notations employed in the ILP formulation.

Symbols	Type	Comment
d	Index	Drain node.
i, j, s	Indices	Denotes nodes in the network.
L_{ij}	Data	Denotes if a arc exists from node i to j . Assume $L_{ij} = L_{ji}, \forall i, j \in \mathcal{N}$.
C_{ij}	Data	Cost of arc $i \rightarrow j$. Assume $C_{ij} = C_{ji}, \forall i, j \in \mathcal{N}$.
α_{ij}	Variable	Binary variable that denotes if arc $i \rightarrow j$ is present in the red tree.
β_{ij}	Variable	Binary variable that denotes if arc $i \rightarrow j$ is present in the blue tree.
γ_{ij}^s	Variable	Binary variable that denotes node s uses arc $i \rightarrow j$ in the red tree.
δ_{ij}^s	Variable	Binary variable that denotes node s uses arc $i \rightarrow j$ in the blue tree.

Objective function:

$$\text{Minimize } \sum_{i,j \in \mathcal{N}} \sum_{s \in \mathcal{N} \setminus \{d\}} (\gamma_{ij}^s + \delta_{ij}^s) C_{ij}$$

Tree constraints:

$$\text{T-1. } \sum_{j \in \mathcal{N}} \alpha_{ij} L_{ij} = 1 \quad \forall i \in \mathcal{N} \setminus \{d\}$$

$$\text{T-2. } \sum_{j \in \mathcal{N}} \beta_{ij} L_{ij} = 1 \quad \forall i \in \mathcal{N} \setminus \{d\}$$

Flow constraints:

$$\text{F-1. } \sum_{j \in \mathcal{N}} \gamma_{ij}^s L_{ij} - \sum_{j \in \mathcal{N}} \gamma_{ji}^s L_{ji} = \begin{cases} 1 & \text{if } i = s \\ -1 & \text{if } i = d \\ 0 & \text{otherwise} \end{cases} \quad \forall s \in \mathcal{N} \setminus \{d\} \text{ and } \forall i \in \mathcal{N}$$

$$\text{F-2. } \sum_{j \in \mathcal{N}} \delta_{ij}^s L_{ij} - \sum_{j \in \mathcal{N}} \delta_{ji}^s L_{ji} = \begin{cases} 1 & \text{if } i = s \\ -1 & \text{if } i = d \\ 0 & \text{otherwise} \end{cases} \quad \forall s \in \mathcal{N} \setminus \{d\} \text{ and } \forall i \in \mathcal{N}$$

$$\text{F-3. } 0 \leq \gamma_{ij}^s \leq \alpha_{ij} L_{ij} \quad \forall s \in \mathcal{N} \setminus \{d\} \text{ and } \forall i, j \in \mathcal{N}$$

$$\text{F-4. } 0 \leq \delta_{ij}^s \leq \beta_{ij} L_{ij} \quad \forall s \in \mathcal{N} \setminus \{d\} \text{ and } \forall i, j \in \mathcal{N}$$

Link-disjoint path constraint:

$$\text{LD-1. } (\gamma_{ij}^s + \delta_{ij}^s) L_{ij} \leq 1 \quad \forall s \in \mathcal{N} \setminus \{d\} \text{ and } \forall i, j \in \mathcal{N}$$

$$\text{LD-2. } (\gamma_{ij}^s + \delta_{ji}^s) L_{ij} \leq 1 \quad \forall s \in \mathcal{N} \setminus \{d\} \text{ and } \forall i, j \in \mathcal{N}$$

Node-disjoint path constraint:

$$\text{ND. } \sum_{j \in \mathcal{N}} (\gamma_{ij}^s + \delta_{ij}^s) L_{ij} \leq 1 \quad \forall s \in \mathcal{N} \setminus \{d\} \text{ and } \forall i \in \mathcal{N} \setminus \{s, d\}$$

Bounds:

$$\text{B-1. } \alpha_{ij}, \beta_{ij} \in \{0, 1\} \quad \forall i, j \in \mathcal{N}$$

Fig. 3. ILP formulation for the construction of two colored trees with link/node-disjoint path constraint.

developed a generalized version of the centralized path augmentation approach (referred to as the XCT algorithm in the rest of the paper) by specifying certain criteria for selecting paths for augmentation, which depend on the problem objective (e.g. minimizing average delay or cost, maximizing bandwidth, etc.).

The XCT algorithm is based on partial ordering of nodes in the network. The partial order \prec of the nodes on the blue tree \mathcal{B} is defined as follows. If $u \rightarrow v \in \mathcal{B}$, then u precedes v in the partial order, represented as $u \prec v$ (the algorithm

in [18] employs partial order on both the red and blue trees for link-disjoint paths. However, the explanation here has been simplified based on the partial order on the blue trees and node-disjoint paths). The partial ordering satisfies the transitive relationship, i.e., if $u \prec v \prec w$, then $u \prec w$. The generalized approach is now described for the construction of two colored trees for the CT-ND problem.

The XCT algorithm for constructing two trees that satisfy the node-disjoint path constraint follows four steps:

- 1) Initialize \mathcal{R} and \mathcal{B} to contain the root node d only.

- **UNVISITED**. This state describes a node that is not part of the trees and is not a part of a path that is being formed for augmentation.
- **VISITED**. A node in this state is being considered as a part of the path for augmentation. However, it is not added to the trees.
- **CYCLE**. A node in this state belongs to both the red and blue trees.
- **TOKEN**. A node in this state has the authority to initiate a path search. After initiating a path search through all its neighbors, sequentially, the node transfers the token to its neighbors.
- **FINISH**. A node in this state is already part of the cycle, has received the token, has initiated a path search along all its neighbors, and has returned the token back to the node from which the token was received. Having all the nodes in the network in this state signifies the termination of the algorithm.

Messages. Let `msg` denote a message exchanged between two nodes. The message has the following fields: (1) `msg.source` is the source of the message; (2) `msg.dst` is the node which the message is intended for; (3) `msg.root` is the root/destination node for which the tree is constructed; (4) `msg.type` is the message type; and (5) `msg.flagNewNodeAdded` is a flag that indicates whether or not the path search through a neighbor resulted in the addition of new nodes. This flag is used to decide if the path search token is to be passed to a neighbor or not, another key element of the distributed implementation.

The algorithm employs the following four different message types:

- **SEARCH**. A node in the **TOKEN** state has the authority to initiate a path search for augmentation. The path search (or the initial cycle) is performed through DFS technique using this message. On receiving this message: (1) a node in the **UNVISITED** state changes itself to the **VISITED** state, (2) a node in the **VISITED** state returns a **FAILURE** message, and (3) a node in the **CYCLE** state returns a **SUCCESS** message.
- **SUCCESS**. When a node in the **CYCLE** state receives a **SEARCH** message, it returns a success message that indicates the termination of a path augmentation. The **SUCCESS** message travels the path taken by the **SEARCH** message in the reverse direction towards the node that initiated the path search.
- **FAILURE**. This message is sent as a response to the **SEARCH** message whenever a path for augmentation cannot be found. When a **SEARCH** message reaches a node that is in the **VISITED** state, then the path search results in a loop, hence the node in the **VISITED** state responds with **FAILURE** message.
- **TOKEN**. This message is the token that is being passed on from the root, giving the authority to a node to initiate a new path search. Only nodes that have been added to the tree (in **CYCLE** state) may receive this message. In addition, only the node that recently received this

message may initiate the search.

- **RETURN**. This message is sent by a node in the **TOKEN** state to the node from which the **TOKEN** message was received. It is sent after the node has initiated a path search along all its neighbors, passed the **TOKEN** to all its neighbors along which a new path was chosen for augmentation, and received the **RETURN** message from all its neighbors to which the token message was sent.

Neighbor list. Every node maintains a list of its neighbors, which can be implemented as a doubly-linked-list. The head and tail pointers of the neighbor list are denoted by `nbrlist.head` and `nbrlist.tail`, respectively. The arrangement of the neighbors from head to tail may be based on any criteria, such as random, increasing advertised cost to the drain (root), etc. For augmenting new paths, the neighbor list is scanned from head to tail. However, when sending the search token, the list is traversed from tail to head. Such a forward-and-reverse traversal guarantees the partial ordering in a distributed fashion.

In this paper, we consider two orderings of the neighbor list at a node: (1) random ordering, where the neighbors are randomly arranged in the list, and (2) Breadth-First-Search (BFS) ordering, where nodes are arranged in an increasing order of their advertised path length to the drain⁵.

A. Working of the Algorithm

An overview of the steps involved in the algorithm is shown in Figure 5. A detailed flowchart is shown in Figure 6.

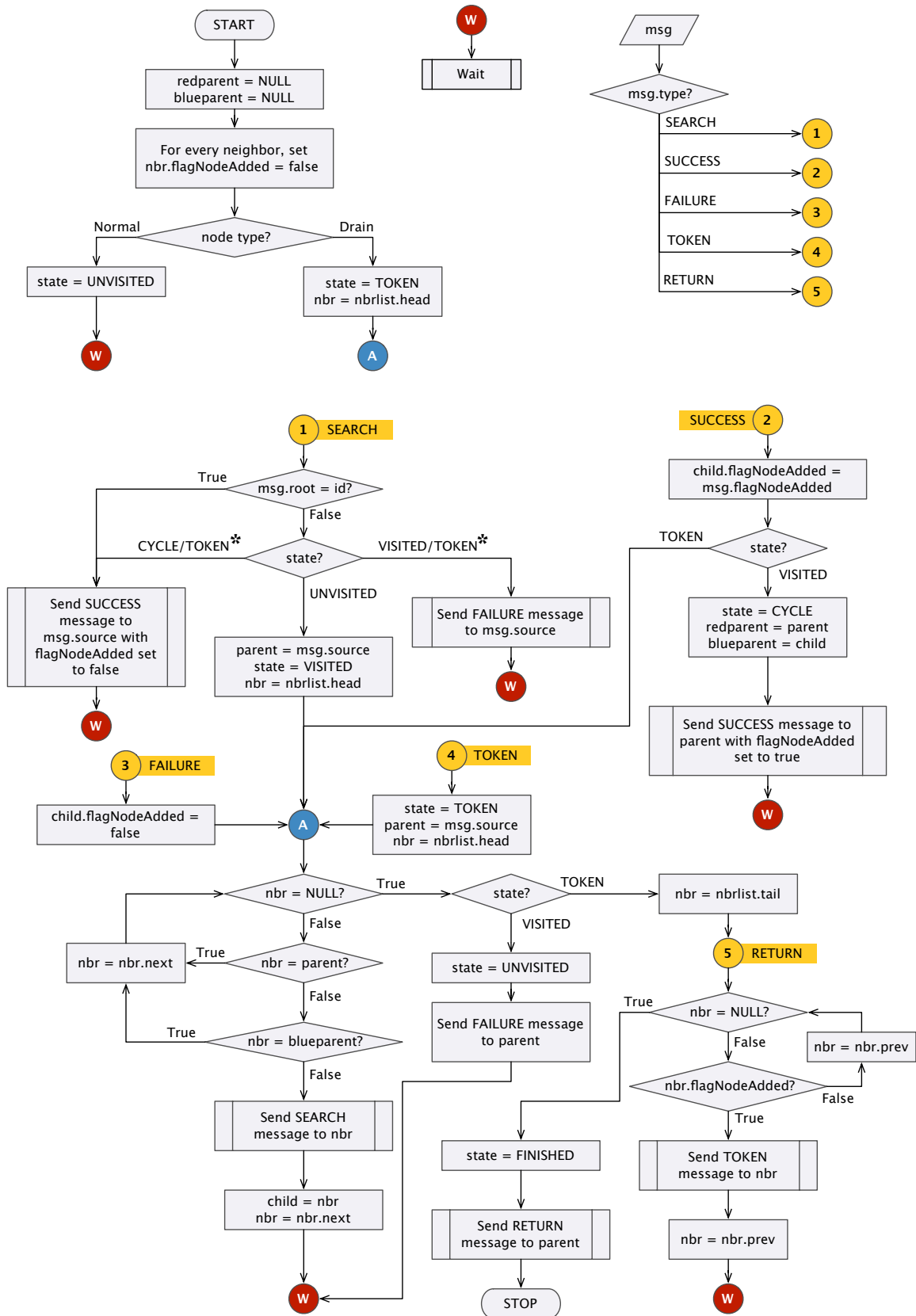
Distributed Path Augmentation Algorithm

- 1) Arrange the neighbors in the neighbor list in an increasing order of their advertised cost (hop count) to the drain.
 - 2) On receiving a **TOKEN** message, initiate path search along every node in the neighbor list, one at a time.
 - a) Every node that receives the **SEARCH** message forwards it sequentially to every node in the neighbor list until it receives a **SUCCESS** message.
 - b) When a **SUCCESS** message returns from a neighbor, the value of `msg.flagNewNodeAdded` is stored next to the neighbor entry in the neighbor list.
 - 3) Forward the **TOKEN** message to every node if the `newNodeAdded` flag for this neighbor is **TRUE**. The neighbor list is traversed in the reverse direction. Every node finishes its operation and sends a **RETURN** message back.
 - 4) After receiving a **RETURN** message from all the neighbors to whom the token message was sent, send **RETURN** message to the parent that sent the **TOKEN** message.
-

Fig. 5. Overview of the steps involved in the distributed algorithm for computing two colored trees.

Initialization. The drain is initialized to the **TOKEN** state. The other nodes are initialized to the **UNVISITED** state. Every node in the network is assumed to employ a neighborhood discovery protocol. A simple periodic exchange of “Hello”

⁵Node numbers are used to break ties.



* A node is configured either to send a success or fail message if it is in the TOKEN state. A response with a SUCCESS message will provide a solution to the CT-LD case, while a response with FAIL message will provide a solution to CT-ND case.

Fig. 6. Flowchart of the distributed algorithm when executed at a node.

message would suffice for such a neighborhood discovery. The neighbors of a node are arranged in a random order in the neighbor list. This order is later chosen in a specific manner to minimize the average delay along the two trees.

Path search. The drain initiates a path search sequentially along its neighbors. The first search is for a cycle while the others are for a path. On receiving a SEARCH message, a node in the UNVISITED state changes itself to the VISITED state. The SEARCH message is then forwarded to one of the neighbors (based on the neighbor list maintained at the node). The drain always responds to a SEARCH message with a SUCCESS. When a SEARCH message reaches any other node, that node responds with a SUCCESS message if it is in the CYCLE state. The global ordering employed in the distributed algorithm (described in the previous section) guarantees that if a node in the TOKEN state receives a SEARCH message, then the path search must have been initiated by the same node. This property allows simple adaptation of the algorithm for CT-ND and CT-LD cases. If the node in the TOKEN state is configured to send a SUCCESS message, then the paths for augmentation may start and end at the same node, resulting in a solution for the CT-LD case. If the node in the TOKEN state is configured to respond with a FAILURE, then the result would be a solution for the CT-ND case.

The sequence of nodes that are in the VISITED state indicates the path under formation for augmentation. If the SEARCH message reaches any node in the VISITED state, then a loop is formed. Such a loop is avoided by having the node in the VISITED state respond to the SEARCH message with a FAILURE message. When a node in the VISITED state receives a FAILURE message, it forwards the search along successive neighbors in its list until all neighbors are exhausted. If the search through all the neighbors result in a failure, the node changes its state from VISITED to UNVISITED and returns a FAILURE message back to the node from which it received the SEARCH.

A node in the CYCLE state responds to a SEARCH message with a SUCCESS message in which `msg.flagNewNodeAdded` is set to FALSE. This enables the receiving node to not forward the search token to a node that is already on the cycle. As a rule, *a path search token may be forwarded from node i to node j only if node j was added to the path through a path search message from node i to j .* Note that as paths are being searched from the highest node in the ordered list, any node that is on the cycle that receives the search message must be lower in order than the node that initiated the message in the ordered list. Upon receiving a SUCCESS message, a node in the VISITED state changes to the CYCLE state. It adds the node from which it received the SEARCH message as its parent on the blue tree and the node from which it received the SUCCESS as its parent on the red tree. The `flagNewNodeAdded` variable for that neighbor is set to the value indicated in the message. The node then sends a SUCCESS message to the node from which it received the SEARCH message with the `msg.flagNewNodeAdded` set to true.

Forwarding search token. A node that has the path search token attempts to augment a path through each of its neighbor.

The node then forwards the token to those eligible neighbors, traversing the ordered list in the reverse direction (opposite to the order in which the SEARCH messages were initiated), one at a time. An eligible neighbor is one for which the variable `flagNewNodeAdded` is set to true. Such an order reversal for passing the token helps maintain a consistent global ordering in a distributed manner across all the nodes in the network. A node that receives a TOKEN changes its state from CYCLE to TOKEN, starts the path search along each of its neighbors, and forwards the token to its eligible neighbors.

Once the tokens are returned by all neighbors, the node sets its state to FINISH and returns the token to the node from which it first received the token. The token finally reaches the drain, indicating that all nodes in the network are in the FINISH state, at which point the algorithm terminates.

B. Example

The working of the distributed tree construction algorithm is illustrated on an example network shown in Figure 7. The numbers inside the parenthesis denote the shortest path length to the drain, taken as node 1. The neighbors are assumed to be arranged in the neighbor list in the BFS order. The working of the algorithm is as follows:

- 1) Node 1 has the TOKEN. It initiates a path search through node 2. The cycle formed is 1–2–3–6–7–8–1. The links added to the blue tree are shown in Figure 7(a). Node 1 then attempts path search through node 8. However, this node is already added to the trees. As the `flagNewNodeAdded` is false for node 8, the token is forwarded to node 2.
- 2) Node 2 initiates a path search through node 7 (node 3 is skipped because it is the blue parent). The search succeeds, however no new nodes are added to the trees. The token is passed to node 3.
- 3) Node 3 initiates a path search through node 4. The path chosen for augmentation is 3–4–5–6. The links added to the blue tree are shown in Figure 7(b). The path search is not initiated through node 6 because it is the blue parent. Search token is passed to node 4.
- 4) Node 4 does not have any paths to add, as node 5 is the blue parent and node 3 is the red parent. The token is passed to node 5.
- 5) Node 5 augments the path 5–12–11–6. The links added to the blue tree are shown in Figure 7(c). The search token is passed to node 12.
- 6) Node 12 does not have any paths to augment, as both its neighbors are its parents on one of the trees. The token is forwarded to node 11.
- 7) Node 11 augments the path 11–10–7. The links added to the blue tree are shown in Figure 7(d). Node 11 passes the token to node 10.
- 8) Node 10 augments the path 10–9–8. The links added to the blue tree are shown in Figure 7(e). Node 10 passes the token to node 9.
- 9) Node 9 does not have any paths to augment. Node 9 does not pass the token to node 8 as the `newNodeAdded` flag is set to false; hence, it returns the token to node 10.

- 10) The token starts to return from every node and finally reach the drain, indicating the completion of the algorithm.

The blue and red trees thus formed are shown in Figures 7(e) and (f), respectively.

Proof of correctness. The distributed algorithm is based on the path augmentation technique [17]. Hence, the proof of correctness of the algorithm follows from [17] and is not repeated in this paper due to space constraints.

V. PERFORMANCE EVALUATION

The ILP formulation is solved using the CPLEX 8.0 solver [19] with an objective to minimize the average hop length of a path on a tree. The effectiveness of the distributed algorithm is evaluated and compared against the ILP. As the ILP execution time may be prohibitively large, we consider a small network in which we compare the results of the ILP to the heuristic approach. We demonstrate the effectiveness of the BFS ordering by comparing its performance against random ordering of nodes in the neighbor list.

Performance metrics. The performance metrics considered are: (1) ILP execution time, (2) number of messages of each type transmitted, (3) average path length on the red and blue trees, and (4) Average minimum and maximum path lengths across two trees. Among the two paths from each node to the drain, the average of the length of the shortest (longest) path over all nodes is computed as the “average minimum (maximum) path length.” The rationale for these metrics is that when a message is sent along the red and blue paths, the first message received by the drain would have most likely traveled on the shorter path while the second message on the longer path. If the paths are employed for redundancy purposes, then from the standpoint of the receiver, the delay to obtain the message equals that of the shortest path delay (hop-count in this case).

The simulation results for the distributed algorithm were using a C++ program. The solution time for the simulation is simply the running time of the program to obtain a solution.

The performance evaluation reported in this paper has been restricted to obtaining trees satisfying link-disjoint path constraint (by setting the response to SEARCH message at the node initiating the search to SUCCESS). The performance results for node-disjoint path constraints are not reported due to space limitations.

We consider two kinds of network topologies, grid and random. We consider an $M \times M$ network where nodes are placed on a two-dimensional grid at integer coordinates. Two cases are studied where a link between two nodes exists if the distance between them is no more than: (a) 1 unit, and (b) $\sqrt{2}$ units. The node at the origin is assumed to be the drain.

The performance of the algorithm is studied using random arrangement of nodes in the neighbor list. Ten random arrangements of neighbors were employed. The average and standard deviation of various metrics are computed.

A. Results and Discussion

Tables I(a) and (b) show the performance results for a 7×7 grid network. As expected, the ILP solution time for network is significantly higher than the heuristic approach. The heuristic approach completed well under 10 ms for all scenarios. As the neighborhood arrangement technique is only a heuristic approach to minimizing the average path length, it is certainly not optimal. This fact is observed from the tables as well. The average minimum path length is found to be closer to the average minimum path length of the ILP. Thus, under lighter loads, when two messages are transmitted together with one on each tree, the first message is expected to reach the drain sooner, while the second message may arrive much later. Such a property is often preferred in sending redundant data.

The arrangement of neighbors based on the advertised shortest path to the drain helps in significantly reducing the running time. It is observed that the standard deviation of the solution time with random arrangement of nodes is higher than the average, indicating that certain arrangements result in an excessive amount of backtracking during the path searching process. The behavior is also observed based on the number of FAILURE messages, which is reduced significantly by the BFS ordering. The number of TOKEN messages is the same as the number of nodes in the network, indicating that every node receives the token exactly once. The drain that generates this message is also counted in the computation.

The algorithm is also evaluated for a 10×10 grid network. The results are shown in Table II. The optimal solution could not be obtained using ILP for such large networks in a reasonable time. ILP results are, therefore, not reported for this network. The performance results follow similar trends as seen in the 7×7 network. The impact of BFS arrangement over random arrangement is observed to be more significant with increased network sizes.

If two link-disjoint paths were to be obtained independently from every node to the drain in a 7×7 grid network (with 84 links), the average sum of the two path lengths may be obtained as 13.375. Therefore, if destination-and-source-based routing is employed, the number of routing table entries is 642. In contrast, the number of routing table entries needed using the colored tree approach is 96 (two entries per node). The size of the routing table has been reduced by a factor of 6.6875 over the network. Similarly, for the 10×10 grid network, the average hop sum of the path lengths is 19.27 when two link-disjoint paths are computed independently. The colored tree approach, therefore, reduces the routing table information by a factor of 9.635 compared with the source-based routing alternative.

For a grid network, a simple scheme for link-disjoint routing can be designed using the geometrical property of grid: If one route first traverses the x-axis and followed by y-axis while the other does vice versa, the two paths will be link-disjoint. However, such routing may be employed only when the entire topology is known, which is not the case here.

The distributed algorithm with random and BFS orderings is also evaluated on randomly generated topologies with 100 and 300 nodes. The random topologies are generated

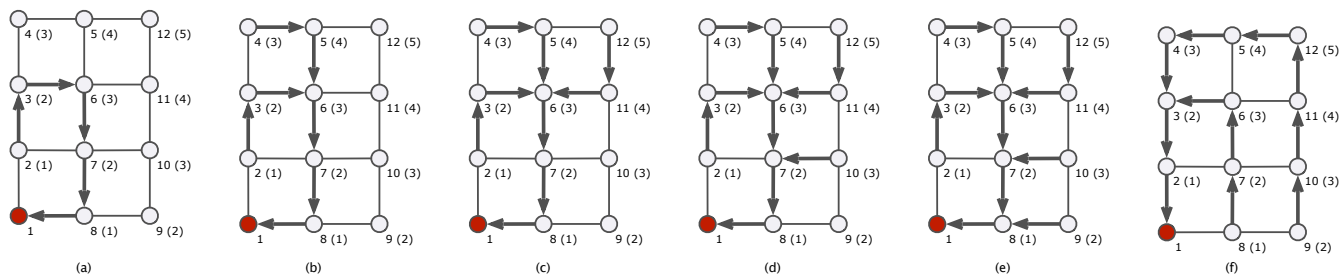


Fig. 7. Example showing the working of the distributed tree construction algorithm with BFS arrangement. The numbers in parenthesis denote the shortest path length from the node to the drain while those outside the parenthesis represent the node numbers.

TABLE I
PERFORMANCE RESULTS FOR 7×7 NETWORK

(a) Network has links between nodes that are not more than 1 unit apart (84 links).

Metrics	ILP	With Neighbor Arrangement	Random Arrangement	
			Mean	Std. Dev.
Solution Time (in seconds)	5133	< 10 ms	< 10 ms	< 10 ms
PATH-SEARCH	-	173	10972	23743
PATH-SEARCH-SUCCESS	-	168	168	0
PATH-SEARCH-FAIL	-	5	10804	23743
PATH-SEARCH-TOKEN	-	49	49	0
PATH-SEARCH-TOKEN-RETURN	-	49	49	0
Average RED Path length	6.3	13.458	16	1.8
Average BLUE Path length	6.3	7	12	2.8
Average Minimum Path length	6.125	6.125	7.8	1.05
Average Maximum Path length	6.475	14.333	20.2	3.67

(b) Network has links between nodes that are not more than $\sqrt{2}$ units apart (156 links).

Metrics	ILP	With Neighbor Arrangement	Random Arrangement	
			Mean	Std. Dev.
Solution Time (in seconds)	85113	< 10 ms	< 10 ms	< 10 ms
PATH-SEARCH	-	312	14930	41663
PATH-SEARCH-SUCCESS	-	312	312	0
PATH-SEARCH-FAIL	-	0	14618	41663
PATH-SEARCH-TOKEN	-	49	49	0
PATH-SEARCH-TOKEN-RETURN	-	49	49	0
Average RED Path length	4.35	9.71	16.5	3.2
Average BLUE Path length	4.27	5.22	19.07	3.4
Average Minimum Path length	4.08	4.95	8.51	1.49
Average Maximum Path length	4.54	10.08	27.06	3.68

TABLE II
PERFORMANCE RESULTS FOR 10×10 NETWORK

(a) Network has links between nodes that are not more than 1 unit apart (360 links).

Metrics	With Neighbor Arrangement	Random Arrangement	
		Mean	Std. Dev.
PATH-SEARCH	368	434.3	125.6
PATH-SEARCH-SUCCESS	360	360	0
PATH-SEARCH-FAIL	8	74.3	125.6
PATH-SEARCH-TOKEN	100	100	0
PATH-SEARCH-TOKEN-RETURN	100	100	0
Average RED Path length	21.79	16.38	2.47
Average BLUE Path length	10	27.56	4.92
Average Minimum Path length	9.09	11.26	0.83
Average Maximum Path length	22.7	32.68	6.4

(b) Network has links between nodes that are not more than $\sqrt{2}$ units apart (522 links).

Metrics	With Neighbor Arrangement	Random Arrangement	
		Mean	Std. Dev.
PATH-SEARCH	684	8522.5	22882.9
PATH-SEARCH-SUCCESS	684	684	0
PATH-SEARCH-FAIL	0	7838.5	22882.9
PATH-SEARCH-TOKEN	100	100	0
PATH-SEARCH-TOKEN-RETURN	100	100	0
Average RED Path length	16.09	38.8	6.50
Average BLUE Path length	7.22	41.9	4.34
Average Minimum Path length	7.02	19.9	3.09
Average Maximum Path length	16.29	60.8	7.78

using Waxman's model [20]. Tables III and IV show the performance results. The average node degree for the 100-node and 300-node networks were observed to be 12.25 and 17.34, respectively. BFS ordering performs better than the random arrangement consistently. Comparing the results of the 10×10 grid network (with 156 links) and 100-node random networks, it may be observed that increasing the connectivity of the network first deteriorates the performance as evidenced from Tables II(a) and (b). The increase in connectivity is observed to improve the performance for random arrangement, beyond a threshold as evidenced from the results of Tables II(b) and III.

VI. CONCLUSIONS

This paper develops an effective disjoint multipath forwarding approach based on colored trees. A distributed algorithm for constructing two trees rooted at a drain such that the paths from every node to the drain are link/node-disjoint is developed. The distributed algorithm is guaranteed to obtain a solution whenever one exists. In addition, a neighbor arrangement technique that orders the neighbors based on the advertised least cost path to the drain is shown to reduce the average path length on the trees significantly. The algorithm is evaluated on six different networks and compared with ILP results to demonstrate feasibility of distributed implementation and its effectiveness. The technique of arranging neighbors

TABLE III

PERFORMANCE RESULTS FOR RANDOM NETWORK TOPOLOGIES WITH 100 NODES

Metrics	BFS Arrangement	Random Arrangement	
		Mean	Std. Dev.
PATH-SEARCH	1119.3	1189.3	64.1
PATH-SEARCH-SUCCESS	1118	1118	--
PATH-SEARCH-FAIL	1.3	71.3	40.09
PATH-SEARCH-TOKEN	100	100	0
PATH-SEARCH-TOKEN-RETURN	100	100	0
Average RED Path length	6.7	23.35	9.25
Average BLUE Path length	3.9	34.34	7.69
Average Minimum Path length	3.2	12.56	4.27
Average Maximum Path length	7.4	45.13	12.57

TABLE IV

PERFORMANCE RESULTS FOR RANDOM NETWORK TOPOLOGIES WITH 300 NODES

Metrics	BFS Arrangement	Random Arrangement	
		Mean	Std. Dev.
PATH-SEARCH	4890.18	5224.1	365.73
PATH-SEARCH-SUCCESS	4890.09	4890.09	0
PATH-SEARCH-FAIL	0.09	356.64	192.56
PATH-SEARCH-TOKEN	300	300	0
PATH-SEARCH-TOKEN-RETURN	300	300	0
Average RED Path length	15.90	79.20	33.94
Average BLUE Path length	3.73	107.90	26.51
Average Minimum Path length	3.53	41.30	16.21
Average Maximum Path length	16.1	145.80	44.08

based on the BFS ordering is shown to improve the performance, however is not guaranteed to provide optimal results.

REFERENCES

- [1] Z. Ye, S.V. Krishnamurthy, and S.K. Tripathi, "A framework for reliable routing in mobile adhoc networks," in *Proceedings of IEEE INFOCOM'03*, April 2003, pp. 270–280.
- [2] P. P. Pham and S. Perreau, "Performance analysis of reactive shortest path and multipath routing mechanism with load balance," in *Proceedings of IEEE INFOCOM*, March-April 2003, vol. 1, pp. 251–259.
- [3] S. Murthy and J. J. Garcia-Luna-Aceves, "Congestion-oriented shortest multipath routing," in *Proceedings of IEEE INFOCOM*, March 1996, vol. 3, pp. 1028–1036.
- [4] D. Ganesan, R. Govindan, S. Shenker, and D. Estrin, "Highly resilient energy-efficient multipath routing in wireless sensor networks," *ACK SIGMOBILE Mobile Computing and Communications Review*, vol. 4, no. 5, pp. 11–25, 2001.
- [5] R. Bhandari, *Survivable Networks: Algorithms for Diverse Routing*, Kluwer Academic Publishers, 1999.
- [6] W. D. Grover, *Mesh-based Survivable Networks: Options and Strategies for Optical, MPLS, SONET and ATM Networking*, Prentice Hall Publishers, New Jersey, USA, 2003.
- [7] A. C. Begen, Y. Altunbasak, and O. Ergun, "Multi-path selection for multiple description encoded video streaming," in *IEEE International Conference on Communications*, May 2003, vol. 3, pp. 1583–1589.
- [8] S. Lee and M. Gerla, "Split multipath routing with maximally disjoint paths in ad hoc networks," in *Proceedings of IEEE ICC*, 2001, pp. 3201–3205.
- [9] A. Nasipuri and S. R. Das, "On-demand multipath routing for mobile ad hoc networks," in *Proceedings of IEEE International Conference on Computer Communications and Networks*, October 1999, pp. 64–70.

- [10] J. Wu, "An extended dynamic source routing scheme in ad hoc wireless networks," in *Proceedings of 35th Annual Hawaii International Conference on System Sciences*, January 2002, pp. 3832–3838.
- [11] M. K. Marina and S. R. Das, "On-demand multipath distance vector routing in ad hoc networks," in *Proceedings of IEEE ICNP*, November 2001, pp. 14–23.
- [12] V. D. Park and M. S. Corson, "A highly adaptive distributed routing algorithm for mobile wireless networks," in *Proceedings of IEEE INFOCOM*, April 1997, pp. 1405–1413.
- [13] J. Raju and J. J. Garcia-Luna-Aceves, "A new approach to on-demand loop-free multipath routing," in *Proceedings of IEEE International Conference on Computer Communications and Networks (ICCCN)*, October 1999, pp. 522–527.
- [14] A. Valera, W. K. G. Seah, and S. V. Rao, "Cooperative packet caching and shortest multipath in mobile adhoc networks," in *Proceedings of IEEE INFOCOM*, March-April 2003, pp. 260–269.
- [15] S. Lee and M. Gerla, "Aodv-br: Backup routing in ad hoc network," in *Proceedings of IEEE WCNC*, September 2000, pp. 1311–1316.
- [16] A. Itai and M. Rodeh, "The multi-tree approach to reliability in distributed networks," in *Foundations of Computer Science'84*, October 1984.
- [17] M. Medard, R.A. Barry, S.G. Finn, and R.G. Gallager, "Redundant trees for preplanned recovery in arbitrary vertex- redundant or edge redundant graphs," *IEEE/ACM Transactions on Networking*, vol. 7, no. 5, pp. 641–652, October 1999.
- [18] G. Xue, L. Chen, and K. Thulasiraman, "Quality-of-service and quality-of-protection issues in preplanned recovery schemes using redundant-trees," *IEEE Journal on Selected Areas in Communication*, vol. 21, no. 8, pp. 1332–1345, October 2003.
- [19] CPLEX Solver, <http://www.cplex.com>.
- [20] B. M. Waxman, "Routing of multipoint connections," *IEEE Journal of Selected Areas in Communications*, vol. 6, no. 9, pp. 1617–1622, December 1988.