Independent-Tree Ad hoc MulticAst Routing (ITAMAR) *

S. Sajama
School of Electrical Engineering
Cornell University
Ithaca, NY 14853
E-mail: sajama@ece.cornell.edu

Zygmunt J. Haas
School of Electrical Engineering
Cornell University
Ithaca, NY 14853
E-mail: haas@ece.cornell.edu
http://wnl.ece.cornell.edu

Abstract

Multicasting is an efficient means of one to many communication and is typically implemented by creating a *multicasting tree*. Because of the severe battery power and transmission bandwidth limitations in ad hoc networks, multicast routing can significantly improve the performance of this type of network. However, due to the frequent and hard-to-predict topological changes of ad hoc networks, maintenance of a multicasting tree to ensure its availability could be a difficult task. We borrow from the concept of Alternate Path routing, which has been studied for providing QOS routing, effective congestion control, security, and route failure protection, to propose a scheme in which a set of multicasting trees is continuously maintained. In our scheme, a tree is used until it fails, at which time it is replaced by an alternative tree in the set, so that the time between failure of a tree and resumption of multicast routing is minimal. In this paper, we introduce the basic scheme, termed *ITAMAR*, which is a framework for efficient multicasting in ad hoc networks. We present a number of heuristics that could be used in ITAMAR to compute a set of alternate trees. The heuristics are then compared in terms of transmission cost, improvement in the average time between multicast failures and the probability of usefulness. Simulations show significant gains over a wide range of network operational conditions. In particular, we show that using alternate trees has the potential of improving mean time between interruption by 100-600% in a 50 node network (for most multicast group sizes) with small increase in the tree cost and the route discovery overhead. We show that by renewing the backup tree set, probability of interruptions can be kept at a minimum at all times and that allowing some overlap among trees in the backup set increases the mean time between interruptions.

*This work has been sponsored in part by ONR contract no. N00014-00-1-0564, AFRL contract no. F360602-97-C-0133, and NSF grant no. ANI-9980521.

1 Introduction

An ad hoc network consists of a collection of mobile routers, which are interconnected via wireless links and are free to move about arbitrarily. This technology has its roots in DARPA packet radio networks [1]. Research on multi hop packet-switching radio networks started in the 1970s, with its initial motivation for military applications. Their attractiveness was (and continues to be) based on the ease and speed of deployment in hard-to-access environments [2]. In recent years, interest in ad hoc networks has grown with improvements in laptop computer technology, including greater computational power, longer battery life and decreased weight. Advent of ubiquitous computing and the proliferation of portable computing devices have further increased the importance of efficient routing in mobile networks.

One of the most pressing needs for enhanced communication protocols come from multi point applications, which involve the one-to-many communication model (i.e., multicasting operation). Such applications cover a very wide spectrum, including software distribution, replicated database update, command and control systems, audio/video conferencing, and distributed interactive simulation.

Multicasting is an efficient communication tool for use in multi-point applications. Many of the proposed multicast routing protocols, both for the Internet and for ad hoc networks, construct trees over which information is transmitted. Using trees is evidently more efficient than the brute force approach of sending the same information from the source individually to each of the receivers. Another benefit of using trees is that routing decisions at the intermediate nodes become very simple : a router in a multicast tree that receives a multicast packet over an "in-tree" interface forwards the packet over the rest of its "in-tree" interfaces.

Multicast routing algorithms in the Internet [3] can be classified into three broad categories : **1**) Shortest

Path Tree algorithms [4], **2**) Minimum Cost Tree algorithms [5], [6] and **3**) Constrained Tree algorithms [7], [8]. In general, the two fundamental approaches used in designing multicast routing are: to minimize the distance (or cost) from the sender to each receiver individually (shortest path tree algorithms) and to minimize the overall (total) cost of the multicast tree. Practical considerations lead to a third category of algorithms, which try to optimize both constraints using some metric (minimum cost trees with constrained delays). Majority of multicast routing protocols in the Internet are based on shortest path trees, because of their ease of implementation. Also, they provide minimum delay from sender to receiver, which is desirable for most real-life multicast applications. However, in some more recent protocols, like PIM [9] and CBT [10], an attempt is made to minimize the state stored in the routers.

Multicasting in ad hoc networks is more challenging than in the Internet, because of the need to optimize the use of several resources simultaneously. Firstly, nodes in ad hoc networks are battery-power limited. Furthermore, data travels over the air and wireless resources are scarce. Secondly, there is no centralized access point or existing infrastructure (like in the cellular network) to keep track of the node mobility. Thirdly, the status of communication links between routers is a function of their positions, transmission power levels, etc. The mobility of routers and randomness of other connectivity factors lead to a network with a potentially unpredictable and rapidly changing topology. This means that, by the time reasonable amount of information about the topology of the network is collected and a tree is computed, it may be only useful for a very short duration, if at all.

Work on multicast routing in ad hoc networks gained momentum in the mid 90s. Some early approaches to provide multicast support in ad hoc networks consisted of adapting the existing Internet multicasting protocols; for example, the Shared Tree Wireless Network Multicast [11] protocol. Other protocols have been designed specifically for ad hoc networks; for example ODMRP [13], AMRIS [14], CAMP [15], and others [16], [17], [18], [19], [20], [21], [22], [23], and [24]. ODMRP is a mesh based, on-demand protocol that uses soft state approach for maintenance of the message transmission structure. It exploits robustness of mesh structure to frequent route failure and gains stability at the expense of bandwidth. The Core Assisted Mesh Protocol (CAMP) attempts to remedy this excessive overhead, while still using a mesh by constructing a core for route discovery. AMRIS constructs a shared delivery tree rooted at a node, with ID-numbers increasing as they radiate from the source. Local route recovery is made possible due to this property of ID numbers, hence reducing the route recovery time and confining route recovery traffic to the region of link failures.

One common characteristic of most of these approaches is that they **react** to a link failure; i.e., they act **after** a link has already failed, causing a significant delay in route recovery. In our work, we have explored the possibility of using a set of **precalculated** alternate trees using the information (about network topology) acquired to calculate the first tree. When a link breaks, another tree, which does not include that link, can be immediately utilized. This often leads to significantly reduced delay, whenever a viable backup tree is available at the time of failure of the current tree. In particular, and possibly most importantly, it allows communication of real-time traffic. This approach is inspired by Alternate Path Routing (APR), which has been used in the Internet to alleviate congestion and to improve QOS. Incidentally, performance gain that can be obtained from use of APR in ad hoc networks for unicast routing has been investigated recently [25].

When the network is reasonably stable, like the Internet, the gain in efficiency due to multicasting (when compared to flooding) more than offsets the cost of route discovery and maintenance. However, as the average velocity of nodes increases, so does the cost of route discovery and maintenance. This means that for any mobility pattern, there is an average velocity of nodes beyond which multicasting is no longer efficient when compared to flooding. This velocity is much higher for our scheme, when compared to other tree-based schemes, because our use of the backup trees. We optimize the cost of the multicast tree along with minimizing the mutual correlation of failure times of each pair of trees under the constraints of partial knowledge of the network.

2 Goals and essential ideas

The goal of this work is to improve multicasting performance in ad hoc networks through efficient use of the available knowledge of the network. The basic idea is that if we are able to compute multiple backup multicast trees with minimal overlap, we could use them one after another to reduce the number of service interruptions. This would also improve the mean time between route discovery cycles for a given interruption rate and hence reduce the control overhead and the rate of data loss. At the same time, we want to keep the cost of transmission low (see Section 5 for a definition of cost). The mobility of ad hoc networks requires

that we use very little time for tree computation and hence it is important for the algorithms to be of low complexity.

2.1 Dependence of a set of trees

This method of using one tree after another will be effective if the trees to be used as backup last for a significant amount of time after the previous trees fail. This means that the failure times of the trees should be independent of one another. If we assume that nodes move independently of one another, then having no common nodes (and hence no common edges) would make the trees fail independently of one another.

However, in the case of ad hoc networks, where the average degree of a node is not high, we expect not to find completely independent trees in many cases. Hence the schemes we develop should concentrate on minimizing the dependence between the failure times. The **dependence** of a pair of trees is defined as the correlation of the failure times of the two trees. Given a pair of trees, their dependence relies on the structure of each of the trees, apart from the number of common nodes and edges.

Dependence of a pair of trees is a complicated function of the mobility pattern of the nodes. Hence a practical way to compute independent enough trees would be to discourage common edges and nodes among the trees. Intuition suggests that having a common edge is much worse (causes more dependence in a pair of trees) than having a common node. It is important to understand how much dependence is caused by a common node, when compared to a common edge. This is done here by a probabilistic analysis to find the correlation of the failure times of two edges sharing a common node. For the sake of this analysis, we use the following assumptions about the network in question:

- nodes are distributed uniformly over the area of the network;
- direction of motion of each node is uniformly distributed across all angles, is independent of other nodes, and does not change after initial selection; and
- the velocity is distributed uniformly between 0 and an upper limit (say V) and does not change after initial selection.

The details of the analysis are given in Appendix A. The following are the main conclusions of the analysis:

- If all the nodes move with the same **constant** velocity, as we would expect when most users are walking or driving along roads or highways, having a common node does not cause any dependence between two trees.
- If the nodes' velocity is uniformly distributed over [0,15m/s][1] and the range of transmission is 76.5m [2], we find that the correlation between the failure times of two adjacent links is 0.172. Hence, under this kind of mobility pattern, it is important to minimize common nodes between trees in addition to minimizing common edges, in order to keep the failure times as independent as possible.

2.2 Dependence vs. the lifetime of the trees' set

As stated before, the goal is to compute trees in such a manner as to maximize the time until the last tree fails. The total time for which a system lasts depends on the individual failure times of the trees used and their independence. If a tree has greater number of links, it is likely to fail faster. On the other hand, if trees have to be maximally independent, they might be less efficient and contain more links, as compared with the case in which some overlap is allowed. Hence the trees that we compute should not be so independent, as to make them fail very fast and hence reduce the total system time. This is an important tradeoff and is discussed in Section 6.4.

2.3 Mechanism to replace trees

Once we compute a set of backup trees and start multicasting, we need to replenish the backup tree set in such a way as to maintain some quality of service, i.e., to maintain the probability of interruptions below some threshold. This means that we need to compute new trees by the time the probability of failure of the current set of trees rises above a given threshold. If the sender has an estimate of the time when this will happen, it could initiate the route discovery process at such a time, T, as to allow for the route discovery and tree computation to be completed in time. In what follows, we propose one way to estimate T given the mobility pattern of the nodes.

It is possible to estimate the probability of interruption occurring before given time or after failure of the first n trees, if we have knowledge of the mobility pattern of the nodes. This estimation is illustrated (via

[1] Roughly, the speed of 35 mph

[2] Roughly, the range of a wireless LAN interface

simulations) in Section 6.5. If the estimation for probability is too high, the scheme will resemble a link state multicast protocol, incurring extraneous cost, while if it is too low it, will be too reactive, leading to interruption of the multicasting service.

Let average time for route discovery be T_{RD} and let the average time for computing the set of n trees for a multicast group size m be $T_{m,n}$. Let $F_{n,m}(p)$ be the time, since failure of first n trees, at which probability that all remaining trees will fail increases above p (finding this function is illustrated in Section 6.5). Let P_T be the threshold below which we desire to keep probability of failure at all times.

Initially, set the estimate of $T = F_{0,m}(P_T) - T_{RD} - T_{m,n}$. At the time when n^{th} tree fails, update estimate of T $= F_{n,m}(P_T) - T_{RD} - T_{m,n}$

At time T after the most recent tree failure (or previous route Discovery, if no tree has failed since then), another cycle of route discovery should be started. If at the time of failure of the n^{th} tree, T is estimated to be negative, new route discovery cycle should be started immediately. Thus the probability of interruption to multicast communication is not allowed to rise above P_T, hence maintaining desired level of quality of service.

2.4 Incorporating ITAMAR into an existing routing protocols

ITAMAR is a way of computing a set of trees, such that they are independent. Hence it can be easily used on top of a suitable unicast layer, which provides route discovery. For example, consider the Dynamic Source Routing (DSR) protocol, a unicast protocol for ad hoc networks. In this protocol, in response to a single route discovery as well as through routing information from other packets overheard, a node may learn and cache multiple routes to any destination. This way, when one of the paths fails, the sender uses another cached route. Knowledge of network obtained from route discovery can be increased by using "Diversity Injection" [27]. Also, once the sender discovers paths to all receivers, one of the algorithms we propose can be used to compute and maintain several multicast backup trees. As a matter of fact, the mechanism required to switch between trees in the event of link failure is already available in DSR.

3 Network model

The ad hoc network is represented via a graph (V, E), where V is the set of nodes and E is the set of edges. The network is assumed to be two dimensional and the mobile hosts are represented by nodes of the graph. An edge between any two nodes is present whenever the two nodes are able to communicate directly with one another. Such nodes are sometimes referred to as *neighbors*. The total number of edges in the graph is denoted by L, i.e., $L = \|E\|$ and the sender node is denoted by O (signifying that the node is the origin of the data). $V1$ ($V1 \subseteq V$) is the set of nodes in the multicast receiver group.

We assume that O has some knowledge of the graph topology from route discovery. $E2$ is the set of all edges in the graph that exist according to O's current view of the network, and thus $(V2, E2)$ are the nodes and edges that belong to all these paths. The goal of this study is to find methods of computing a set of trees $T1$, $T2$, $T3$ .. from O to $V1$ in this graph $G = (V2, E2)$, while minimizing the dependence of their failure times. As is usually the case with multicast trees, $T1$ and $T2$ are directed (though the links of the graph are assumed to be bi-directional); i.e., associated with each link i there is a node at which the link begins O_i and a node at which it ends D_i - the origin and the destination of the link.

In the schemes described below, the set of all edges in the graph along with a quantity called cost of each edge is called the *cost function of the graph*. We extensively use Dijkstra SPF algorithm, which takes the cost function and incidence matrix of the graph as input and computes the shortest path tree from a given source to the given set of receivers.

4 Schemes for computing maximally independent trees

As explained before, the goal of this study is to develop schemes to efficiently compute a set of trees, whose failure times are minimally correlated. Under the assumption that mobility of a node is independent of other nodes, this condition translates to the trees having minimum number of common nodes and edges; with common edges being more undesirable than common nodes (from analysis in Appendix A).

There are two ways of using the backup tree set in the event of a link failure : **1**) replace the whole tree being used currently by a backup tree, if available, or

2) determine which of the receivers are disconnected because of this link failure and replace or augment the paths to those nodes by backup paths.

4.1 Computing backup trees

Three ways of finding sets of disjoint multicasting trees from a sender to a group of receivers have been studied in our work. The trees to be found are referred to as $T1$, $T2$, and so on. $T1$ is intended to be used at the start and the others are to be used as backup.

4.1.1 Matroid Intersection Algorithm (MIA)

The *Matroid Intersection Algorithm (MIA)* [28], [29], [30] can be used to find **two** maximally independent spanning trees on any given graph (i.e., spanning trees with minimum possible number of common edges), such that total cost of the two spanning trees is minimized. The two obtained spanning trees are called $J1$ and $J2$. Given a sender, call it the source node, and a set of receivers, two multicasting trees $T1$ and $T2$ are obtained on graphs $J1$ and $J2$, respectively, using the Dijkstra SPF algorithm.

Matroids and Spanning Trees

Let E be a finite set and I be a family of subsets of E, called "Independent" sets. A subset system $\mathbf{M} = (E, I)$ (the finite set E together with the collection I of subsets of E) is called a *matroid* if the following axioms are satisfied:

1. $\emptyset \in I$
2. if $J' \subseteq J \in I$, then $J' \in I$
3. for every $A \subseteq E$, every maximal independent subset of A has the same cardinality.

Example 1: Let E be the set of all edges in a graph G and let I be the family of subsets of E satisfying the condition that none of them contains a circuit of the graph. Hence independent subsets of this graph are all subsets of trees in this graph.

Example 2: Let E be the set of all edges in graphs G and G' (Figure 1), where G' is a copy of G (edge e'_1 is a copy of e_1, and so on). Two matroids, which can be defined on this set, are:

1. $M_1 = (E, I_1)$, where an "independent" set is a union of subsets of trees of G and G'. For example, an independent set in the collection I_1 could be $\{e_1, e_2, e_3, e_4, e'_5, e'_6, e'_7, e'_2\}$ and a set which would not belong to I_2 would be $\{e_1, e_2, e_7, e'_8\}$ since it has a circuit in it.
2. $M_2 = (E, I_2)$, where an "independent" set is one which does not have both copies of any of the edges. An example of an independent set in I_2 would be $\{e_1, e'_3, e'_7\}$ and a set which would not belong to I_2 would be $\{e_1, e'_3, e'_1\}$ since it contains both copies of e_1.

Hence if a subset of E belongs to both matroids defined above, it will have to be a union of 2 trees, one in G and the other in G'. Moreover, the copy of an edge that belongs to the tree in G should not belong to the tree in G'. This observation indicates that when two edge disjoint trees are possible in a graph G, the set belonging to both the collections I_1 and I_2 and having the maximum possible cardinality will be the union of two disjoint spanning trees. Hence the problem of finding two independent forests in the graph can be thought of as finding a maximum cardinality common independent set of the two matroids defined above.

The Matroid Intersection Algorithm (MIA)

In this algorithm, we start off with a set J which belongs to both I_1 and I_2, say the empty set (Refer to Example 2 above for the definition of I_1 and I_2).

Then we repeatedly increase the size of J with the help of an auxiliary directed graph $G = G(M1, M2, J, w^1, w^2)$ constructed using some rules. The variable w^1 and w^2 determine the weight splitting. These are obtained from the weight splitting variables of the previous step in the algorithm, using the rules of the algorithm and initially starting with $w^1 = w$ and $w^2 = 0$.

G has a node set $E \cup \{r, s\}$ and arcs:

- es for every $e \in E \setminus J$ such that $J \cup \{e\} \in I_1$;
- re for every $e \in E \setminus J$ such that $JU\{e\} \in I_2$;
- ef for every $e \in E \setminus J, f \in J$ such that $J \cup \{e\} \notin I_1, (J \cup \{e\}) \setminus \{f\} \in I_1$;
- fe for every $e \in E \setminus J, f \in J$ such that $J \cup \{e\} \notin I_2, (J \cup \{e\}) \setminus \{f\} \in I_2$;

The costs of arcs of G, p_{uv}, are defined by (ci_0 denotes $max\{w^i_e : e \notin J, J \cup \{e\} \in I_i\}$)

- $p_{es} = w^1_0 - w^1_e$ for each $M1$ arc es with $e \notin J$
- $p_{es} = w^2_0 - w^2_e$ for each $M2$ arc re with $e \notin J$
- $p_{es} = -w^1_e + w^1_f$ for each $M1$ arc ef with $e \notin J$ and $f \in J$
- $p_{es} = -w^2_e + w^2_f$ for each $M2$ arc ef with $e \notin J$ and $f \in J$

If there exists an (r, s) dipath in G, then J is not maximum; in fact, if *r, e1, f1, ..., em, fm, em+1, s* is the node sequence of a chordless (r, s)-dipath, then $J \triangle \{e1, f1, ..., em, fm, em+1\}$ in $I_1 \cap I_2$. If there exists no (r, s) dipath in G, then J is maximum (see [29] for a proof). $J \triangle \{e1, f1, ..., em, fm, em+1\}$ is defined as $J \cup \{e_1\} \backslash \{f_1\} ... \backslash \{f_m\} \cup e_{m+1}$.

Weighted Matroid Intersection Algorithm

Set $k = 0$;
Set $J_k = 0$;
Let $w^1 = w, w^2 = 0$;
While J_k is neither $M1$-basis nor an $M2$-basis
{
Construct $G(M1, M2, J_k, w^1, w^2)$;
Find least weight directed path from r to v
in G of cost d_v for each v
For all $v \in E$, let $\sigma_v = min(d_v, d_s)$ and
replace w_v^1 by $w_v^1 - \sigma_v$, w_v^2 by $w_v^2 - \sigma_v$
Construct $G(M1, M2, J_k, w^1, w^2)$;
If there is an (r, s)-dipath in G
{
Find a least weight (r, s) dipath P having
as few arcs as possible;
Augment J_k on P to obtain J_{k+1}
Replace k by $k + 1$
}
else
{ Choose $J = J_p$ and stop }
}

In the above algorithm, Dijkstra's SPF algorithm could be used to find the minimum weight paths.

Maximal vs. Maximum: When it is not possible to have two completely edge-disjoint spanning trees, the above algorithm gives two trees edge disjoint trees with maximal cardinality (These might not be spanning trees, as adding any more edges might require overlap between the two trees). Hence now to complete each tree, we arrange the links in the other tree in ascending order of their costs and keep adding links to the first tree (omitting the ones that would form circuits) until the first tree is complete and vice versa. Note that multicast trees generated in this way may not have minimum possible number of common edges, though the spanning trees do have this property. Also, note that this scheme can be used only to obtain one backup tree, because the problem of finding intersection of 3 matroids is NP-Hard [31].

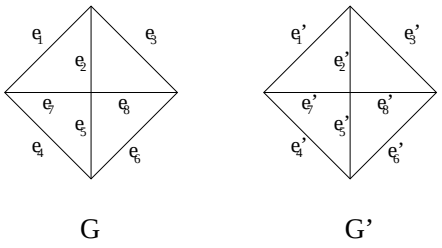

Figure 1: Example of a simple graph

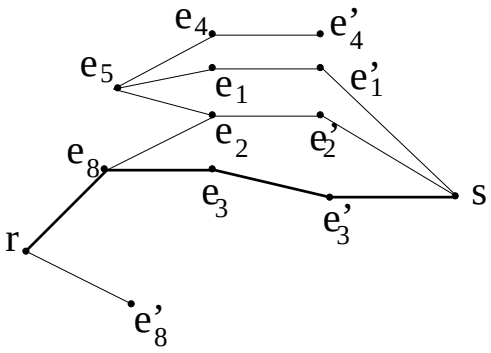

Figure 2: A portion of the sample Auxiliary Digraph

Example

Consider the graph G in Figure 1. The problem is to find two disjoint spanning trees. Firstly we need to verify that this is possible. This is easily done by trying different combinations of 4 edges each; we need four edges to form a tree for a graph with 4 nodes. One example would be the two following trees: $\{e_1, e_3, e_5, e_7\}$ and $\{e_2, e_4, e_6, e_8\}$.

The effectiveness of arriving at a pair of disjoint trees using the Matroid Intersection Algorithm can be seen by going through the process for this simple graph. Suppose we start by building just a tree T_1 first and then removing links of T_1 from the set of

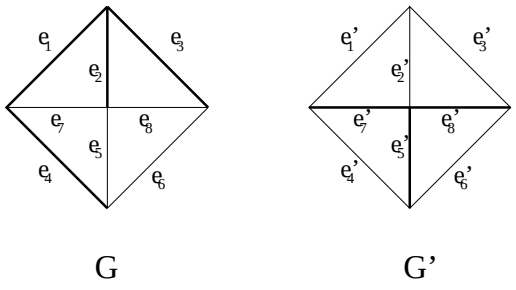

Figure 3: Non-maximal trees obtained by simple enumeration (non-maximal J)

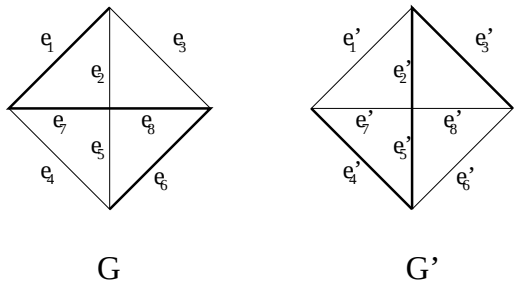

Figure 4: Calculated spanning trees (maximal J)

edges and then trying to build another tree T_2. We would get the following two sets (Figure 3).

$T_1 = \{e_1, e_2, e_3, e_4\}$

$T_2 = \{e_5, e_6, e_7\}$

Adding e'_8 to T_2 will create a circuit and hence is not acceptable. Now we would like to move some edges from T_1 to T_2 and add some new ones to T_1, so that it still remains a tree.

Constructing an auxiliary digraph G (See Figure 2) helps us to find the edges which are to be removed and those which are to be added. In the terminology used above:

$E = \{e_1, ..., e_8, e'_1, ..., e'_8\}$

Current $J = \{e_1, e_2, e_3, e_4, e'_5, e'_6, e'_7\}$

G has the node set $E \cup \{r, s\}$. $e \in S \setminus J$ and $f \in J$.

Its edges are:

- from rule 1, edges of the form es : $e'_1 s, e'_2 s, e'_3 s$ since e'_1, e'_2 and e'_3 combined with T_2 form a tree; i.e., $J \cup \{e\} \in I_1$
- from rule 2, edges of the form re : re'_8, re_8; $J \cup \{e\} \in I_2$
- from rule 3, edges of the form ef : $e_5e_4, e_5e_1, e_5e_2, e_6e_1, e_6e_3, e_6e_4, e_7e_1, e_7e_2, e_8e_2, e_8e_3, e'_4e'_5, e'_4e'_7, e'_8e'_5, e'_8e'_6$; $J \cup \{e\} \notin I_1, (J \cup \{e\}) \setminus \{f\} \in I_1$
- from rule 4, edges of the form fe : $e_1e'_1, e_2e'_2, e_3e'_3, e_4e'_4, e'_5e_5, e'_6e_6, e'_7e_7$; $J \cup \{e\} \notin I_2, (J \cup \{e\}) \setminus \{f\} \in I_2$

Now we need to find an (r, s) dipath in this graph. One of the paths is $P = \{r, e'_8, e'_5, e_5, e_1, e'_1\}$. Doing $J \triangle P$ we obtain the new J as $\{e_2, e_3, e_4, e_5, e'_6, e'_7, e'_1, e'_8\}$, which is a set of two disjoint spanning trees (Figure 4) and hence we are done. (As mentioned before $J \triangle \{e1, f1, ..., em, fm, em + 1\}$ is defined as $J \cup \{e_1\} \backslash \{f_1\} ... \backslash \{f_m\} \cup e_{m+1}$.)

4.1.2 Shortest Path Heuristic (SPTH)

As described in Section 3, the set of all edges in the graph along with a quantity called cost of each edge is called the *cost function of the graph.* The Dijkstra SPF algorithm takes the cost function and incidence matrix of the graph as inputs and computes the shortest path tree from a given source to the given set of receivers.

In the *Shortest Path Heuristic (SPTH)* scheme, the first tree, $T1$, is obtained using the Dijkstra SPF algorithm; i.e., $T1$ is the shortest path tree from the source to the set of receivers. The cost function of the graph is modified after computing the first tree in the following manner: the costs associated with edges which are present in $T1$ are now increased by an amount called the *Link Weight* and the costs associated with edges which share a common node with $T1$ are now increased by an amount called the *Node Weight.* $T2$ is computed using the original incidence matrix of the graph and this new costs. Since Dijkstra SPF algorithm tries to use edges of the lowest cost, this way of modifying the costs discourages use of the edges already used in $T1$ or the edges with a common node with $T1$ (the extent of the discouragement depends on the values of the parameters Link Weight and Node Weight). Computation of subsequent backup trees is carried out in a manner similar to the computation of the second tree, by discouraging the use of links and nodes already used in previous trees by *further* modification of costs. Hence use of nodes present in both trees is discouraged more than nodes used in just one of them.

Shortest Path Heuristic (SPTH) Algorithm

$T1 = Dijkstra\_Algorithm(G, Cost, Source, Receivers)$

Initialize $Cost1$ to be equal to $Cost$ for all edges in G

For each edge i in $T1$

{ $Cost1_i = Cost_i + LinkWeight$ }

For each node in $T1$

{

For each link in G which is incident on this node in $T1$

{ $Cost1_i = Cost_i + NodeWeight$ }

}

$T2 = Dijkstra_Algorithm(G, Cost1, Source, Receivers)$

4.1.3 Low Cost Heuristic (LCH)

The *Low Cost Heuristic (LCH)* algorithm is designed to reduce the total number of transmissions in the multicast trees. The idea is that a single channel wireless network is a broadcast medium, i.e., when a node transmits a packet, all of its neighboring nodes can receive it. Hence to minimize resources used, we should reduce the total number of transmissions required to send data over the multicast tree. To achieve this objective, in Low Cost Heuristic each of the trees are constructed path by path. Computation of a tree given an initial cost function is done in the following way: A path to a node is computed, the cost function is modified, a path to next node is computed and added to the partial tree already constructed, and so on. The modification of cost function in between computing paths to each receiver is done in such a way as to encourage use of minimum number of additional transmissions; i.e., if a link already carries the multicast data , its transmission cost is decreased to a very small value. There will be several links outside the current partial tree with this property, because of the broadcast nature of ad hoc networks.

The cost function taken at the beginning of computation of second tree is a modified version of the original cost function of the tree. This is done in order to discourage use of links and nodes already used in prior trees; for details of modification look at description in Shortest Path Heuristic. Computation of subsequent backup trees is carried out by discouraging use of links and nodes already used in previous trees by modification of the cost function.

Low Cost Heuristic Algorithm

Initialize $Cost'$ to $Cost$
For each receiver j
{
$P_j = Dijkstra_Algorithm(G, Cost', Source, j)$
For each edge in $P1 \oplus \cdots \oplus P_j$
{ $Cost'_i = 0$ }
For each node in $P1 \oplus \cdots \oplus P_j$
{
For each link in G which is incident on this node in $P1 \oplus \cdots \oplus P_j$
{ $Cost'_i = \epsilon$ }
}
}
$T1 = P_1 \oplus P_2 \oplus \cdots \oplus P_N$
Initialize $Cost1$ to $Cost$
For each edge i in $T1$
{ $Cost1_i = Cost_i + LinkWeight$ }
For each node in $T1$
{
For each link in G which is incident on this node in $T1$
{ $c1_i = c_i + NodeWeight$ }
}
Initialize $Cost'$ to $Cost1$
For each receiver j
{
$P'_j = Dijkstra_Algorithm(G, Cost', Source, j)$
For each edge in $P'1 \oplus \cdots \oplus P'_j$
{ $Cost1_i = 0$ }
For each node in $P'1 \oplus \cdots \oplus P'_j$
{
For each link in G which is incident on this node in $P'1 \oplus \cdots \oplus P'_j$
{ $c1_i = \epsilon$ }
}
}
$T2 = P'_1 \oplus P'_2 \oplus \cdots \oplus P'_N$

4.2 Computing backup paths

The *Independent Path Algorithm (IPA)* computes trees such that paths to each receiver in these trees are disjoint, while allowing paths to different receivers to overlap within trees. A lot of work has been done on utilizing path independent trees (trees in which paths to each receiver are independent of one another) in the context of Alternate Path Routing. Using the Independent Path Algorithm we investigate the usefulness of this concept in ad hoc network multicasting.

The problem with using trees as backup is that even if just one link in the tree fails, we need to replace the whole tree by another, when most of the first tree may, in fact, be still intact. Instead, in the Independent Path Algorithm, we start off with a tree and then for each receiver, have a set of backup **paths**, which are maximally disjoint from one another and from the path to the receiver in the first tree.

The first tree can be computed using either Dijkstra SPF algorithm or using the Low Cost Heuristic (if cost is critical). For each receiver, a path independent of the original path to the node in the first tree is computed by modifying the cost function (as is done in the Shortest Path Heuristic) in order to discourage use of already used nodes and edges.

This method differs from the backup tree methods not only in that it replaces only the damaged part of the tree (local repair), but also in that the backup path to any given receiver can overlap with the rest of the first tree (apart from what is being used to transmit data to that receiver). It is more likely to find paths independent from a given path rather than one independent from a given tree. As in the previous two methods, computation of subsequent backup trees is

carried out (similar to what is done in the case of the second tree) by discouraging the use of links and nodes already used in previous trees through modifying the cost function.

Independent Path Algorithm

$T1 = Dijkstra_Algorithm(G, Cost, Source, Receivers)$
Initialize $Cost1$ to equal $Cost$
For receiver node k
{
For link i in path (in $T1$) node k
{ $Cost1_i = Cost_i + LinkWeight$ }
For each node in path (in $T1$) node k
{
For each link in G which is incident on this node in $T1$
{ $c1_i = c_i + NodeWeight$ }
}
Backup Path to k is $Dijkstra_Algorithm(G, Cost1, Source, k)$
}

5 Performance comparison criteria

5.1 Cost

A number, *Cost* c_i, is associated with each link i in the graph. As in traditional networks, it could be chosen to be inversely proportional to the link capacity, proportional to the current load on the link, the delay of the link, etc, or some combination of these parameters. Hence it changes with the changes in the network, such as congestion. For example, the Cost of a failed link is infinite. The choice depends on what one would like to minimize while communicating information in a given multicast group.

The Cost of a tree is defined as the sum of Costs of all the links in the tree. The Cost of a set of trees is defined as the sum of Costs of all the trees in the set.

For a given multicast group size, the average Cost of a scheme is the weighted average of the Cost of all the trees being computed, weighted by the average amount of time each of the trees is being used; i.e., it is

$Average\ Cost = \frac{\Sigma_{i=1}^{n} Cost_{treei} * T_{treei}}{\Sigma_{i=1}^{n} T_{treei}}$.

5.2 Dcost

The idea behind defining *Dcost* is that in a single channel wireless network, the MAC layer is naturally of broadcast type. In other words, when a node transmits, all its neighbors are able to listen to it. Hence the cost of transmission of information to all neighbors from one node is the same as the cost of transmission to the most "expensive" neighbor.

To find Dcost, we divide up the tree graph T into many trees T^j with the following property: if an edge i belongs to T^j for some j, all edges in T with the same origin node as i (denoted by O_i) also belong to T^j and all other edges that belong to T^j have the origin node O_i. With each T^j, we associate a number dc_j which is $\{max\ c_i\ :\ i \in T^j\}$. The Dcost of the tree T is then defined as $\sum_j dc_j$. The Dcost of a set of trees is defined as the sum of all Dcosts of the trees in the set.

Just as in the case of the Cost, the average Dcost of a scheme is the weighted average of the Dcost of all the trees being computed, weighted by the average amount of time each of the trees is being used; i.e., it is

$Average\ Dcost = \frac{\Sigma_{i=1}^{n} Dcost_{treei} * T_{treei}}{\Sigma_{i=1}^{n} T_{treei}}$.

5.3 Time of failure or mean time between interruptions

The *time of failure of a tree* is the minimum time by which at least one of the links of the tree fails and the *time of failure of the system* is the minimum time at which all paths to at least one of the multicast receivers fail in the first and in all the backup trees.

We use the terms *system time* and the *mean time between interruptions*, Interchangeably, since an interruption occurs whenever there is a failure of all the trees triggering re-computation of trees.

5.4 Probability of usefulness

The probability that any of the trees will be used is defined as the *probability of usefulness*. It is that fraction of the total number of trials for which failure time of the system is greater than failure time of the first tree.

5.5 Increase in mean time between interruptions

The *increase in mean time between interruptions* due to backup is $T_{system} - T_{tree1}$. Here T_{system} is the time

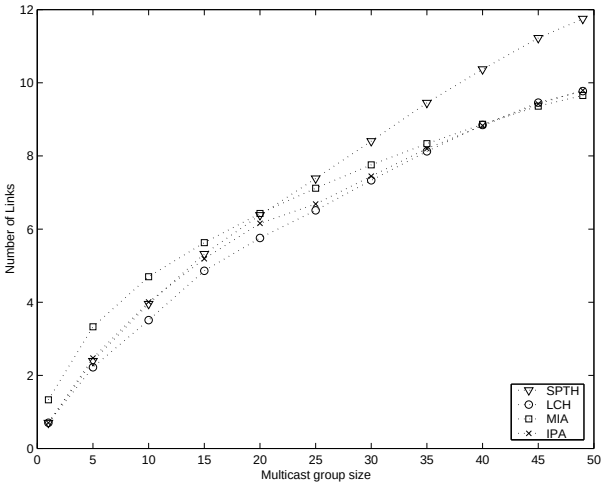

Figure 5: Average Cost (One backup tree)

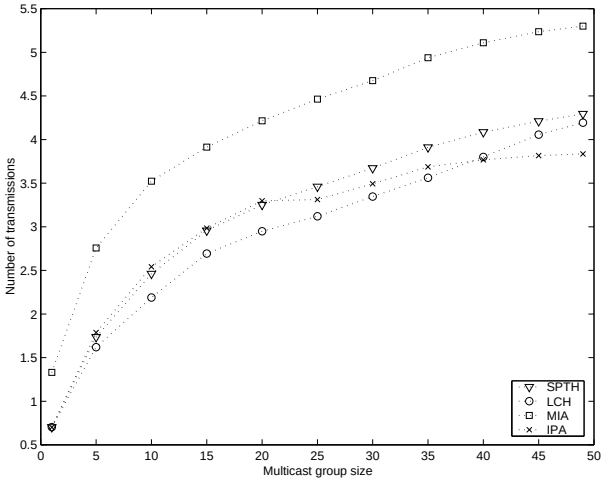

Figure 6: Average Dcost (One backup tree)

of failure of the system and T_{tree1} is the time of failure of the first tree.

6 Simulation results and discussion

6.1 The simulation environment

N nodes are uniformly distributed over a square area of size L meters by L meters. Each node can exchange

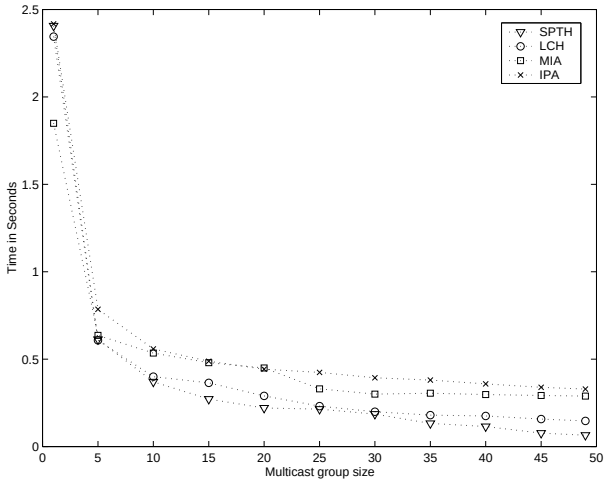

Figure 7: Time of failure of the system (One backup tree)

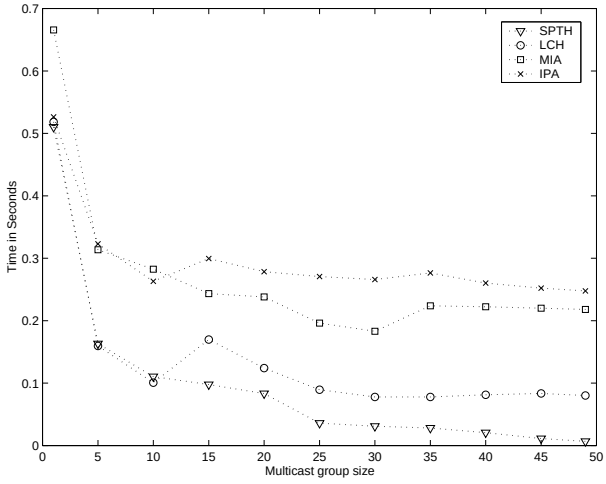

Figure 8: Increase in mean time between interruptions (One backup tree)

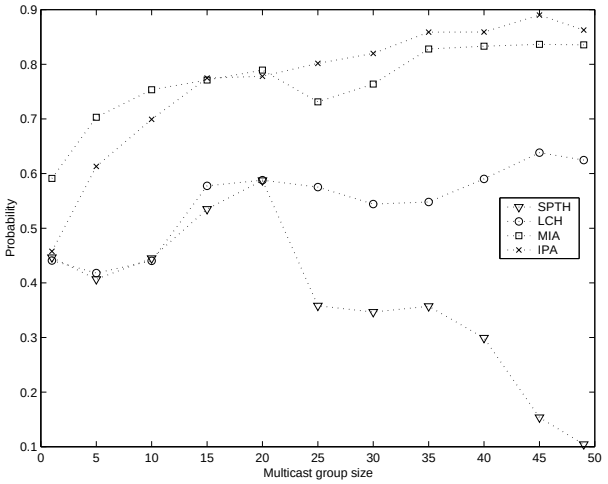

Figure 9: Probability of usefulness (One backup tree)

information with any other node within R meters of itself. At time 0, with probability 0.5 they pick a destination point (which is also uniformly distributed in the area) and start moving in that direction with velocity V [m/s] and with probability 0.5 they wait in their positions for a random amount of time (uniformly distributed over [0,5 sec]) before choosing a destination. After reaching their destination point, they stop with probability 0.5 in their positions for a random amount of time (uniformly distributed over [0,5 sec]), choose another destination point and start moving in the new direction with probability 0.5.

The nodes were allowed to move according to the above mobility model until the multicast tree and all backups had failed and the failure times of various schemes were recorded. The data presented here are averaged over 500 different trees (each under 25 realizations of the mobility pattern) for each multicast group size. In all the graphs in this section, SPTH refers to the Shortest Path Tree Heuristic, LCH refers to the Low Cost Heuristic, IPA refers to the Independent Path Algorithm, and MIA refers to the Matroid Intersection Algorithm.

6.2 Results for one backup tree

Simulation results in this section are for a 50 node network in a square area of size 700m by 700m with nodes moving at 40[m/s] and with the transmission range of 140m.

Figure 5 shows that the *average Cost* of trees used is not very different for the various schemes. However, from Figure 6 we see that the average number of transmissions required per packet, the *Dcost*, is significantly higher for the Matroid Intersection Algorithm, while the Dcost curves for other three schemes are relatively bunched together. This is because of the fact that the Matroid Intersection Algorithm, in the process of making the two spanning trees edge disjoint, causes links incident on any given node to be distributed among the two spanning trees. Hence, there is a smaller number of links incident on any given node in each of these spanning trees, when compared to the whole network. Because of this, each multicast tree (which is computed on these spanning trees as described in section 4.1.1) has smaller number of outgoing links to choose from at each node and hence has greater number of transmissions.

If we were using just one tree, we would expect that the *mean time between interruptions* be reduced with an increase in multicast group size. This is so, since increase in multicast group size increases the size of the tree and hence increases the probability that at least one of the links fails by any given time. However, while using backups, the total time for which the system lasts may increase with an increase in multicast group size due to the increase in probability of usefulness. This is because even though the first tree fails faster, the backup trees are available more often, hence increasing the total time, on an average, for which the the set of trees lasts. The effect of these two factors can be seen in Figures 8 and 7. From these figures, we can rank the schemes based on the increase in the mean time between interruptions, because of the backup trees, in the following order IPA, MIA, LCH and SPTH, with the IPA scheme performing best. The two trees in LCH are expected to have greater independence than the SPTH, because by encouraging several links from one node to be included in the first tree, we make the tree occupy a smaller "area", hence leaving greater space for the other tree to be formed without having to overlap with the first one.

MIA ensures that the trees are almost edge disjoint, by computing the two trees simultaneously, while the SPTH and LCH compute the first tree before the second one and hence losing out on the possibility of combined optimization. IPA lasts much longer than other schemes, especially for larger multicast groups, because of the fact that it includes local repair. Firstly, since independence means that the two paths to each receiver are independent of one another, the average dependence does not increase much with the size of the multicast group like the other schemes. Secondly, since we do local repair, if one part of first tree and one part of second tree have failed, the system can still be working by some combination of the two trees, hence increasing the system time.

Number of backup trees	SPTH	LCH	IPA
1	0.100s	0.142s	0.297s
2	0.058s	0.081s	0.105s
3	0.056s	0.079s	0.132s

Table 1: Increase in mean time between interruptions due to n^{th} backup tree

Number of backup trees	SPTH	LCH	IPA
1	0.37	0.54	0.77
2	0.07	0.12	0.12
3	0.03	0.05	0.03

Table 2: Increase in probability of usefulness due to n^{th} backup tree

Probability of usefulness decreases with an increase in dependence between the two trees. For the tree based algorithms, dependence between the two trees increases with increase in multicast group size, because each tree occupies more "area." On the other hand, as the size of a tree increases, its failure time decreases. For this reason, given that first tree fails, it is very likely that the rest of the network is still intact and hence the second tree is intact with higher probability. These opposing factors can be seen at play in Figure 9 especially for the SPTH curve. Despite the above argument, the LCH curve does not change much, because the two factors balance each other out. In the case of MIA, the two trees are almost edge disjoint, irrespective of the size of the group and hence only the second factor dominates. Same is the case with IPA, because, as described in the previous paragraph, its dependence does not increase much with group size.

Results for two and three backup trees follow the same trends for various parameters as in the one tree backup case, but with greater improvements in terms of probability and mean time between interruptions, in terms of probability of usefulness, and in terms of higher cost and Dcost of the trees.

6.3 Improvements as a function of number of backups

This section presents the performance of the algorithms as a function of The number of backup trees computed. The Table 1 contains the increase in mean time between interruptions (averaged over multicast group sizes) as a function of the number of backups for various schemes. We see that the IPA results in greater increase in mean time between interruptions than the other two schemes for any amount of backup used. Surprisingly, we also observe that, for IPA, the improvement in time due to third backup tree is greater than the improvement due to second backup tree. This is also true for LCH and SPTH for low multicast group sizes, where dependence between two trees is still low.

To understand how this might be possible, consider 4 independent, identically distributed random variables in time: Tb_1, Tb_2, Tb_3, Tb_4. They could for examples of the time of failure of 4 paths from a sender to a receiver, say P_1, P_2, P_3 and P_4. Let the cumulative distribution function (cdf) of Tb_i be $F(t)$. Then, the cdf of time by which two paths fail is $F^2(t)$ ($P\{max(Tb_1, Tb_2) \leq t\} = P\{Tb_1 \leq t, Tb_2 \leq t\} = P\{Tb_1 \leq t\} * P\{Tb_2 \leq t\} = F(t) * F(t)$) and so on. Hence mean time of failure with i backup trees is:

$$\int_t t \frac{d(F(t))}{dt} \frac{d(F^{i+1}(t))}{dt} dt.$$

Hence, if I_i denotes the improvement due to i^{th} backup, the improvement due to the third backup minus the improvement due to second backup is:

$$I_3 - I_2 = \int_t t \frac{d(F(t))}{dt}(2F(t) + 4F^3(t) - 6F^2(t))dt$$

Of the terms in the integral t, dt, and $\frac{d(F(t)}{dt}$ are all positive, while $2F(t) + 4F^3(t) - 6F^2(t)$ is positive for $F(t) \in [0, 0.5]$ and negative for $F(t) \in [0.5, 1]$. Hence it is possible for the improvement due to third backup to be better than improvement due to second backup. In fact, $I_3 - I_2$ is positive for the cdf for failure time of a single link calculated theoretically in Appendix A.

From Table 2, we observe that the increase in probability of usefulness decreases with an increase in number of backup trees. This is an expected result, because with an increase in n, the probability that at least one first $n-1$ backup trees is available along with the n^{th} backup tree at the time of failure of first tree increases.

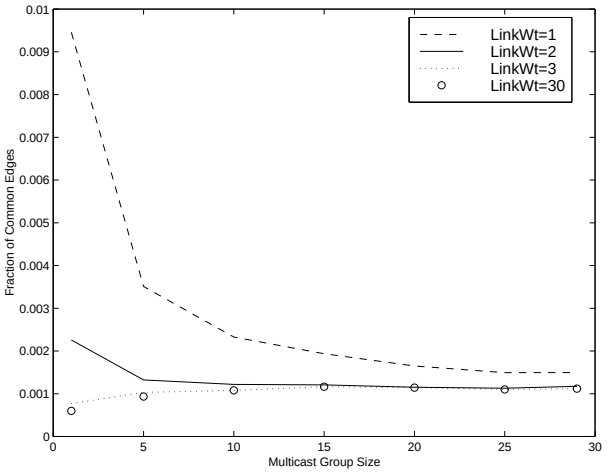

Figure 10: Ratio of number of common edges to average of the number of edges in the two trees

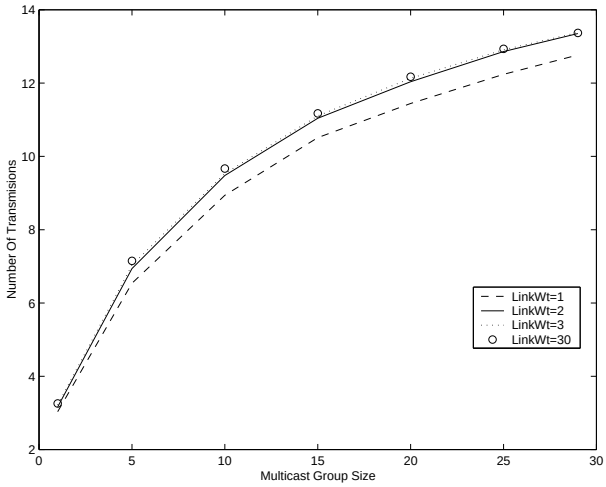

Figure 11: Average Dcost of Trees as a function of Link Weight

6.4 Tradeoff between dependence of a pair of trees and mean time between interruptions

This section illustrates that maximum independence among the trees does not maximize mean time between interruptions (explanation has been provided in section 2.2). Results in this section are based on the Independent Path Algorithm and a 30-node network with node degree of 6 and one backup tree. The Link Weight and Node Weight parameters (defined in Section 4), which regulate the amount of dependence, were set equal to each other and varied over the range [1,30]. The effect on the number of common edges, Dcost of trees, and the mean time between interruptions is shown in Figures 10, 11 and 12. We see that an increase in Link Weight/Node Weight increases the average Dcost of trees monotonically and decreases the number of common edges (and hence dependence) monotonically. However, we see that the mean time between interruptions increases first, until Link Weight/Node Weight value of 2, and then decreases slightly to reach a saturation level shown by the circles. This illustrates the tradeoff between dependence of a pair of trees and the total time for which at least one of them lasts (Section 2.2). Hence the ITAMAR framework performs best when some dependence is allowed among the set of trees, by choosing a relatively moderate value for the Link Weight and the Node Weight parameters in tree computation algorithms.

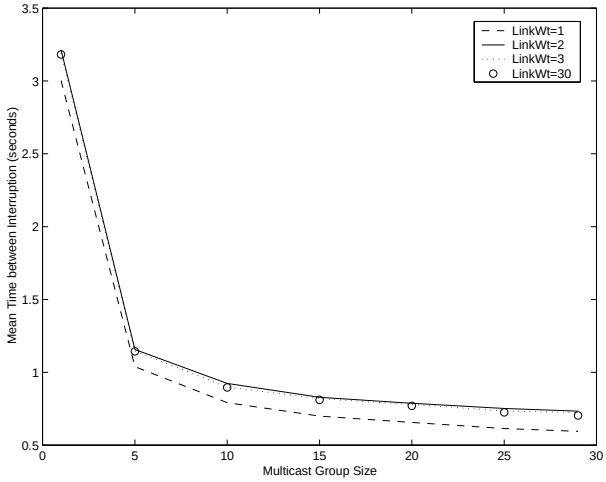

Figure 12: Mean time between interruption as a function of Link Weight

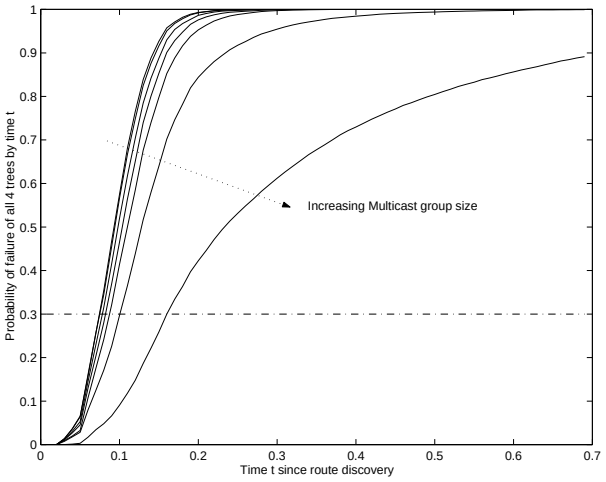

Figure 13: Probability that the multicast tree set has not failed by time t since route discovery

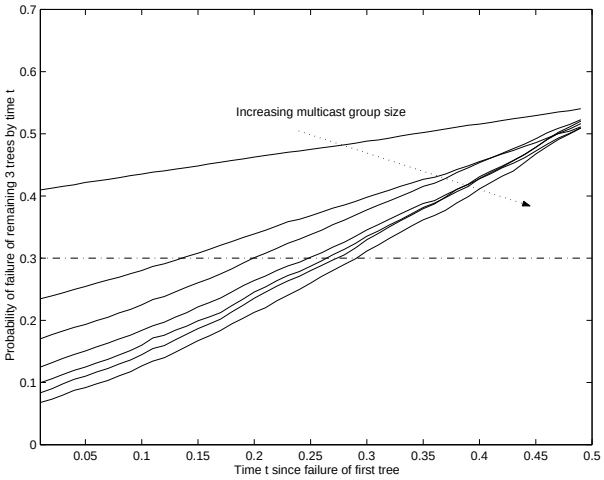

Figure 14: Probability that the multicast tree set has not failed by time t since failure of first tree

6.5 Replacing trees

Results in this section are for a 30-node network with node degree of 6. The curves shown are for multicast group size of 1,5,10,15,20,25, and 29. To implement the scheme, the time by which route discovery has to be redone in order to maintain a low probability of interruptions has to be determined. Figure 13 shows the probability of interruption occurring after any given time t measured from most recent route discovery cycle ($F_{0,m}$ referred to in Section 2.3). For example, if we want the probability of interruption to be below 30% at all times, initial estimate of the time by which we have to redo the route discovery is 0.07 seconds for group size 1 and 0.17 seconds for group size 29 (Figure 13). Figures 14 and 15 show the probability of interruption occurring after any given time, measured from the failure time of Tree 1 and Tree 2, respectively. For the threshold of 30% for probability of interruption, updated estimate of the time by which to redo route discovery at the time of failure of first tree might be greater than 0 for many multicast group sizes, i.e., we might be able to wait some more time before starting route discovery (since several curves are below 30% near t=0). However, at the time of failure of the second tree, we will have to start route discovery immediately, if we have not already done so, since probability of interruption is above 30% for t=0, for all multicast group sizes (Figure 15).

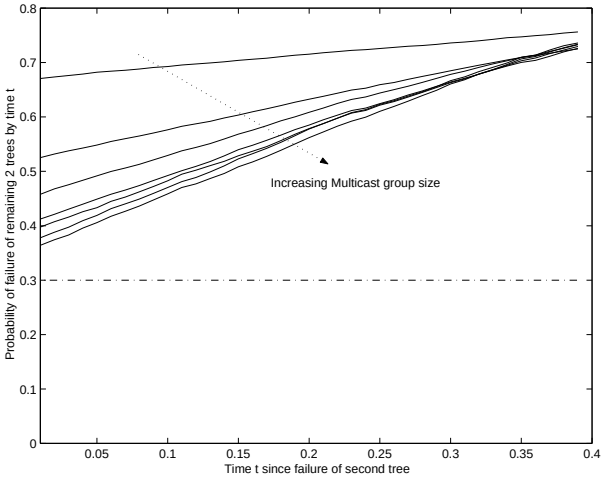

Figure 15: Probability that the multicast tree set has not failed by time t since failure of second tree

6.6 Sensitivity analysis

6.6.1 The effect of speed of nodes

Simulations for this set of results were done on a 30-node network with a node range of 76.5m and node degree 6. To study the effect of the speed, the speed of mobiles was set at values 1.8 m/s (walking speed), 18m/s (speed of a car), and 40 m/s (high speed). The results showed a small decrease in probability of usefulness with increase in speed. However, the percentage increase in the mean time between interruptions was almost the same for all three speeds (Figure 16).

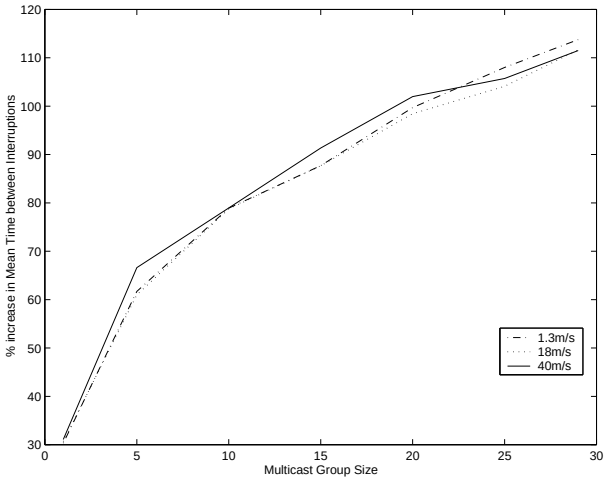

Figure 16: Percentage increase in mean time between interruptions as a function of speed

6.6.2 The effect of node degree

Simulations for this set of results were done on a 30-node network with a node range of 76.5m and speed of 18m/s. Average node degree of the network was set at 4, 6, and 8. We saw a decrease in the average Cost of trees, which is expected, since an increase in node degree means that there are more outgoing links from each node and hence Dcost will be lower. Also, since there are more links in the network but the same number of nodes, greater degree of independence is possible between nodes and hence we saw an increase in % Increase in mean time between interruptions (Figure 17).

6.6.3 The effect of the network size

The number of nodes in the network was set to 30, 40, and 50 nodes. The node degree was maintained at 6, the range of nodes at 140m, and the speed at 40m/s. We found that the average Cost of trees increases with the number of nodes (since average number of links between any two nodes increases), while the average Dcost decreases. However the probability of usefulness does not change much, while the percentage of the increase in mean time between interruptions increases with the number of nodes (Figure 18).

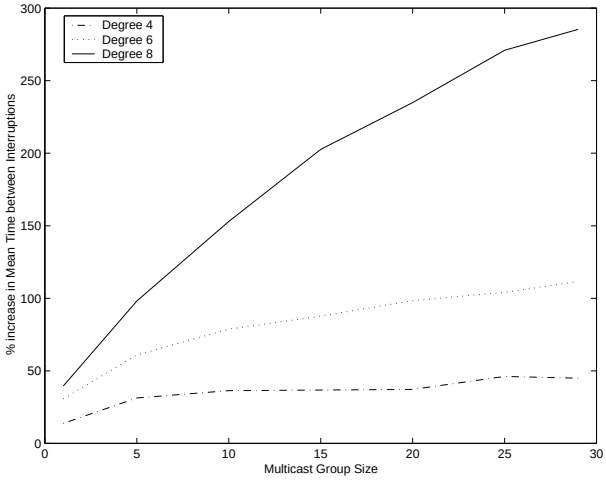

Figure 17: Percentage increase in mean time between interruptions as a function of Node Degree

7 Conclusions

Several heuristic schemes for constructing multiple "independent" trees were developed, simulated, and their performance figures were compared in various network conditions. We found that the Independent Path Algorithm gives much better performance than the other schemes, with a very small increase in transmission cost of the multicast trees. We have shown through simulations that in a typical ad hoc network it is possible to

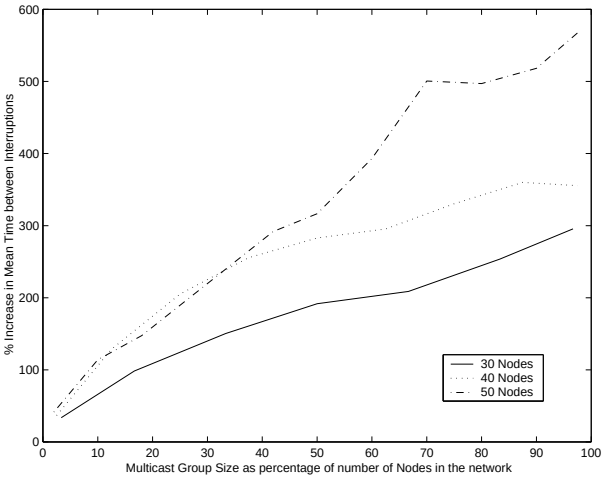

Figure 18: Percentage increase in mean time between interruptions as a function of Total number of Nodes in the network

have working backup infrastructure with high probability, without much extra expense in terms of the cost of the trees, the computation complexity, or data collection overhead. Existence of an optimal level of independence that allows for maximum mean time between interruptions has been illustrated through simulation. The probability of backup being useful is 0.9 for just 2 backup trees computed with no extra control overhead and mean time between interruptions is increased by 100%-600% (for most multicast group sizes) through the use of 3 backup trees in a 50-node network. The simulation results also indicate that, contrary to intuition, the improvement obtained due to additional trees does not always decrease with an increase in number of backup trees. Sensitivity analysis for the Independent Path Algorithm indicates performance gains over a wide range of network conditions, with performance gains increasing with an increase in size of the network and in node degree, while remaining constant with an increase in speed. Timely update of the backup tree set can keep the probability of interruption below a desired value. One way of estimating the time to update the backup tree set has been proposed and illustrated for a 30-node network.

References

[1] J. Jubin and J. Tornow, *The DARPA Packet Radio Network Protocols*, Proceedings of the IEEE, 1987, pp 21-32.

[2] I. Gitman, R. M. Van Slyke and H. Frank, *Routing in Packet Switching Broadcast Radio Networks*, IEEE Transactions on communications, Aug. 1976, pp 926-930.

[3] S. Paul, *Multicasting on the Internet and its applications*, Kluwer Academic Publishers, 1998.

[4] D. Bertsekas and R. Gallagher. *Data Networks*, Prentice Hall, 1992.

[5] C.-H. Chow, "On multicast path finding algorithms," *Proceedings of the IEEE INFOCOM '91,* pp. 1274-1283, 1991.

[6] B. M. Waxman, "Performance Evaluation of Multipoint Routing Algorithms," *Proceedings of IEEE INFOCOM '93*, pp. 980-986, 1993.

[7] B. Kadaba and J. M. Jaffe. "Routing to multiple Destinations in Computer Networks," *IEEE Transactions on Communications*, vol. com-31, no. 3, pp. 343-351, March 1983.

[8] V. P. Kompella, J. C. Pasquale, and G. C. Polyzos, Multicast routing for multimedia communication, IEEE/ACM Transactions on Networking, Vol. 1, No. 3, pp. 286-292, June 1993.

[9] S. E. Deering, D. Estrin, D. Farinacci, V. Jacobson, C-G Liu and L. Wei, "An Architecture for Wide-Area Multicast Routing," *IEEE/ACM Transactions on Networking*, Vol. 4, No. 2, Pages 153-162, April 1996.

[10] A. Ballardie, "Core Based Trees (CBT Version 2) Multicast Routing - Protocol Specification," *RFC-2189,* September 1997.

[11] C. Chiang, M. Gerla, and L. Zhang, "Shared tree wireless network multicast", *IEEE International Conference on Computer Communications and Networks (ICCCN'97)*, September 1997.

[12] J. Meggers and G. Filios, *"Multicast Communication in ad hoc Networks,"* Vehicular Technology Conference, 1998, pp. 372-376.

[13] S.-J. Lee, M. Gerla and C.-C. Chiang, "On-Demand Multicast Routing Protocol", *Proc. IEEE WCNC'99*, New Orleans, LA, Sept 1999, pp. 1298-1304.

[14] C. W. Wu and Y. C. Tay, *"AMRIS: A Multicast Protocol for Ad hoc Wireless Networks,"* Proceedings of IEEE MILCOM '99, Atlantic City, NJ, Nov. 1999.

[15] J.J. Garcia-Luna-Aceves, and E.L.Madruga, "The Core-assisted mesh protocol," *IEEE Journal on Selected Areas in Communications*, Special Issue on Ad-Hoc Networks, Vol. 17, No. 8, Aug. 1998.

[16] Elizebeth Royer, and Charles E. Perkins "Multicast Operation of ad-hoc, on-demand distance vector routing protocol," *MobiCom '99*, Aug. 1999.

[17] Bommaiah, McAuley, Talpade, and Liu. *AMRoute: Ad hoc multicast routing protocol*, Internet-Draft, IETF, August 1998.

[18] S. Lee and C. Kim, "Neighbor Supporting Ad Hoc Multicast Routing Protocol", *2000 First Annual Workshop on Mobile and Ad Hoc Networking and Computing*, 2000. pp 37-44.

[19] L. Briesemeister and G. Hommel, "Role-Based Multicast in Highly Mobile but sparsely Connected Ad Hoc Networks", *2000 First Annual Workshop on Mobile and Ad Hoc Networking and Computing*, 2000. pp 45-50.

[20] H. Zhou and S. Singh, "Content Based Multicast in Ad Hoc Networks", *2000 First Annual Workshop on Mobile and Ad Hoc Networking and Computing*, 2000. pp 51-60.

[21] G. D. Kondylis, S. V. Krishnamurthy, S. K. Dao and G. J. Pottie, "Multicasting Sustained CBR and VBR Traffic in Wireless Ad Hoc Networks", *International Conference on Communications*, 2000. pp 543-549.

[22] C. Sankaran and A. Ephremides, "Multicasting with Multiuser detection in Ad-Hoc Wireless Networks", *Proceedings of the Conference on Information Sciences and Systems(CISS)*, 1998. pp 47-54.

[23] P. Sinha, R. Sivakumar and V. Bharghavan, "MCEDAR: Multicast Core-Extraction Distributed Ad hoc Routing", *In Proc. of the Wireless Communications and Networking Conference*, 1999. pp. 1313-1318.

[24] J. E. Wieselthier, G. D. Nguyen, and A. Ephremides, "Algorithms for Energy-Efficient Multicasting in Ad Hoc Wireless Networks", *Proceedings of the IEEE Military Communications Conference (MILCOM)*, 1999, pp. 1414-1418.

[25] M. Pearlman and Z. Haas, *On the impact of Alternate Path Routing for Load Balancing in Mobile Ad Hoc Networks*, MobiHOC'2000, Boston, MA, Sept. 1999, pp 3-10.

[26] Katia Obraczka, Gene Tsudik, "Multicast Routing Issues in Ad Hoc Networks," IEEE International Conference on Universal Personal Communication (ICUPC'98), Oct. 1998.

[27] M. Pearlman and Z. Haas, *Improving the Performance of Query-Based Routing Protocols Through 'Diversity Injection'*, WCNC'99, New Orleans, LA, Sept. 1999.

[28] Lawler, E. L., "Matroid Intersection Algorithms," *Math. Prog.*, September 1975, pp 31-56.

[29] W.J. Cook, W.H. Cunningham, W.R. Pulleyblank, and A. Schrijver, *Combinatorial Optimization*, John Wiley and Sons, 1998.

[30] Edmonds, J., "Minimum Partition of a Matroid into Independent Subsets," *J. Res. NBS*, 69B, 1965, pp 147-153.

[31] R. G. Parker and R. L. Rardin, *Discrete Optimization*, Academic Press, Inc., 1988.

A Dependence of adjacent links

In this Appendix, the dependence between two edges of a graph with a common node is analyzed probabilistically under the following (usual) assumptions about spatial distribution of nodes and their mobility patterns:

- nodes are distributed uniformly over the area of the network;
- direction of motion of each node is uniformly distributed in all directions, is independent of other nodes and does not change after initial selection; and
- the nodal velocity is distributed uniformly between 0 and an upper limit (say V) and does not change after initial selection.

Notation and conventions used in the following analysis are as follows: r is the range of transmission of each node; n_0, n_1, and n_2 are the three nodes in question; r_1 and r_2 are distances between n_0 and n_1, and between n_0 and n_2, respectively at time $t = 0$. All angles are measured in the counterclockwise direction from the x-axis. The x and y axes are defined along the line joining n_0 with n_1 and the line joining n_0 with n_2, respectively. F_1 is a coordinate frame of reference fixed to the earth, while F_2 is a coordinate frame of reference fixed to n_0 and hence moves with respect to

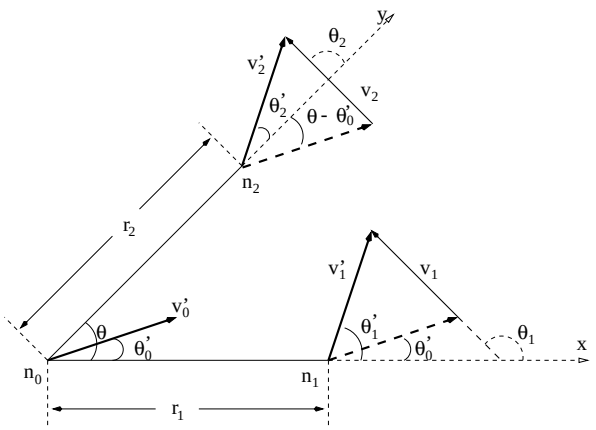

Figure 19: Two edges with a common node

F_1 with the same velocity as n_0. The V_i' is the speed of n_i in F_1, θ is the angle of y-axis, θ_0', θ_1', and $\theta_2' + \theta$ are angles of velocity of n_0, n_1, and n_2, respectively in F_1. The V_i is the speed of n_i in F_2, θ_0, θ_1, and $\theta_2 + \theta$ are angles of velocity of n_0, n_1, and n_2, respectively in F_2. Finally, Tb_1 and Tb_2 are times at which transmission between n_0 and n_1, and transmission between $n_0$and n_2, respectively, fails.

The basic random variables and their distributions (derived from the assumptions stated above) are:

1. V_0', V_1', V_2' - assumed to be uniformly distributed over the interval [0,V]
2. θ_0', θ_1', θ_2' and θ - assumed to be uniformly distributed over the interval [0,2π)
3. r_1 and r_2 with values in the interval [0,r] with $P(r_1 <= x) = \frac{x^2}{r^2}$. (This is because we want the probability that a node is in a region to be proportional to the area of that region.)

Writing horizontal and vertical components of V_1 and of V_2 (Figure 19), we obtain the expressions for V_1, V_2, θ_1, and θ_2 in terms of the basic random variables:

$$V_{1x} = V_1' \cos\theta_1' - V_0' \cos\theta_0'$$

$$V_{1y} = V_1' \sin\theta_1' - V_0' \sin\theta_0'$$

$$V_1 = \sqrt{V_{1x}^2 + V_{1y}^2} \quad (1)$$

$$= \sqrt{V_1'^2 + V_0'^2 - 2V_0'V_1' \cos(\theta_1' - \theta_0')} \quad (2)$$

$$\theta_1 = \tan^{-1}(\frac{V_{1y}}{V_{1x}}) \quad (3)$$

$$= \tan^{-1}(\frac{V_1' \sin\theta_1' - V_0' \sin\theta_0'}{V_1' \cos\theta_1' - V_0' \cos\theta_0'}) \quad (4)$$

$$(5)$$

$$V_2 = \sqrt{V_{2x}^2 + V_{2y}^2} \quad (6)$$

$$= \sqrt{V_2'^2 + V_0'^2 - 2V_0'V_2' \cos(\theta_2' - \theta_0' + \theta)} \quad (7)$$

$$\theta_2 = \tan^{-1}(\frac{V_{2y}}{V_{2x}}) \quad (8)$$

$$= \tan^{-1}(\frac{V_2' \sin\theta_2' - V_0' \sin(\theta_0' - \theta)}{V_2' \cos\theta_2' - V_0' \cos(\theta_0' - \theta)}) \quad (9)$$

It can be seen from the expression for V_2 and θ_2 above that θ_0' always appears along with θ and that irrespective of the value of θ_0', $(\theta_0' - \theta)$ is uniform over $[0, 2\pi)$. Hence we conclude that V_2 and θ_2 are independent of the random variable θ_0'. The only common random variable between V_1, θ_1 and V_2, θ_2 is V_0'. **From this we deduce that if all the nodes are moving at the same constant velocity, as we would expect when most users are walking or driving along a defined trail, having a common node does not cause any dependence between the two trees**. Also, we can rewrite the expression for V_2 and θ_2 in the following way without affecting our calculations in any way (where θ is a random variable uniformly distributed over $[0, 2\pi)$):

$$V_2 = \sqrt{V_2'^2 + V_0'^2 - 2V_0'V_2' \cos(\theta_2' - \theta)} \quad (10)$$

$$\theta_2 = \tan^{-1}(\frac{V_2' \sin\theta_2' - V_0' \sin\theta}{V_2' \cos\theta_2' - V_0' \cos\theta}). \quad (11)$$

Finding expressions for the random variables Tb_1 and Tb_2 in terms of the basic random variables is done by calculating the time taken for the distance between two nodes to increase beyond the transmission range r:

$$Tb_1 = \frac{1}{V_1}(\sqrt{r^2 - r_1^2 \sin^2\theta_1} - r_1 \cos\theta_1) \quad (12)$$

$$Tb_2 = \frac{1}{V_2}(\sqrt{r^2 - r_2^2 \sin^2\theta_2} - r_2 \cos\theta_2). \quad (13)$$

We assume that the basic random variables listed above are an independent set. Since V_0' is the only random variable common to the expressions between V_1, θ_1 and V_2, θ_2 we conclude that it is the random variable causing dependence between Tb_1 and Tb_2. Now we have all the necessary expressions to compute $Pr(Tb_1 <= t/V_0')$, for a given value of V_0'. If we hold V_1' and $\theta_1' - \theta_0'$ constant, V_1 remains constant and only

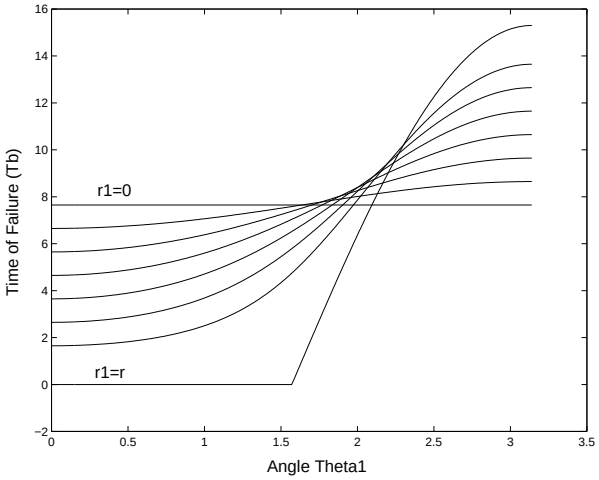

Figure 20: Tb as a function of θ_1 for different values of r_1

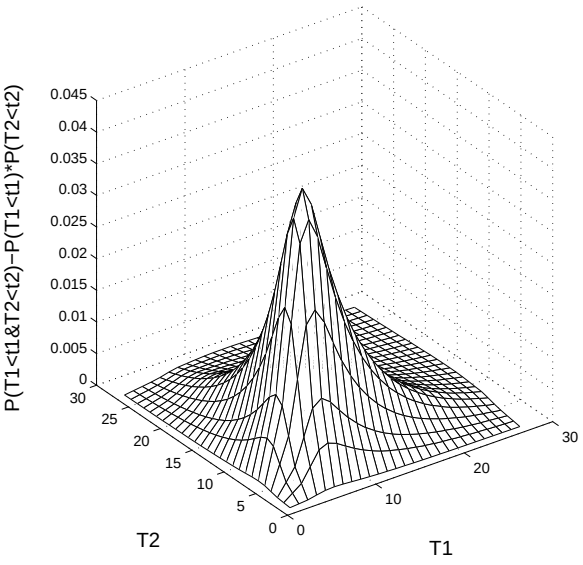

Figure 21: Difference between the actual joint CDF of failure times of two adjacent links and the joint CDF expected if they were independent

θ_1 changes. Observe that value of θ_0' completely determines that of θ_1' and that θ_1 starts at some value (depending on V_1' and $\theta_1' - \theta_0'$), call it $\theta_{1-initial}$, for $\theta_0' = 0$ and increases linearly with θ_0'. Hence the range and distribution of θ_1 ($(0, 2\pi)$ and uniform) is independent of V_1' and $\theta_1' - \theta_0'$ and hence independent of V_1.

Figure 20 shows variation of Tb_1 with θ_1 and r_1 for fixed V_1. It can be seen from the graph that given r_1, the probability that $Tb_1 \leq t$ is the angle θ_e at which the curve corresponding to r_1 crosses the line $Tb_1 = t$. A simple calculation dictates that:

$$\theta_e = \cos^{-1}(1 - \frac{(r_1 + V_1 t)^2 - r^2}{2V_1 t r_1}) \tag{14}$$

and the probability that $Tb_1 \leq t$ is found using the following expression

$$Pr(Tb_1 \leq t/V_1) = \int_x \frac{\theta_e}{\pi} P(r_1 = x)dx \tag{15}$$

Considering all the different cases: **a)** the line $Tb_1 = t$ cuts the curve corresponding to r_1, **b)** the line $Tb_1 = t$ passes above the curve corresponding to r_1 and **c)** the line $Tb_1 = t$ passes below the curve corresponding to r_1 we obtain the following expression:

$$Pr(Tb_1 \leq t/V_1) = \begin{cases} INT, & V_1 < \frac{r}{t}; \\ \frac{(V_1 t - r)^2}{r^2} + INT, & \frac{r}{t} < V_1 < \frac{2r}{t}; \\ 1, & V_1 > \frac{2r}{t}. \end{cases} \tag{16}$$

where,

$$INT = \frac{2}{\pi r^2} \int_{|r - V_1 t|}^{r} r' \cos^{-1}(1 - \frac{(r' + V_1 t)^2 - r^2}{2V_1 t r'})dr' \tag{17}$$

Now to find CDF of V_1 for a fixed V_0'. Defining α to be V_1'/V_0' and ϕ to be $\theta_1' - \theta_0'$, we can rewrite

$$V_1 = V_0' \sqrt{\alpha^2 + 1 - 2\alpha \cos\phi} \tag{18}$$

In the above equation, ϕ is uniformly distributed over $(0, 2\pi)$ and α is uniformly distributed over (V/V_0'). Defining α_1 to be V_1/V_0', we see that for given α, the value of α_1 lies between $|\alpha - 1|$ and $\alpha + 1$ (Figure 22). Also, α_1 increases monotonically with ϕ for a given α. We can see that given $\alpha_1 = x$, the ϕ at which $\sqrt{\alpha^2 + 1 - 2\alpha \cos\phi}$ equals x is

$$\phi_{max} = \cos^{-1}(\frac{1 + \alpha^2 - x^2}{2\alpha}). \tag{19}$$

Hence the expression for $Pr(\alpha_1 \leq x)$ is found to be

$$Pr(\alpha_1 \leq x) = \frac{H(x-1)}{1 + V/V_0'} + \int_{|\alpha - 1|}^{G} \frac{1}{\pi} \phi_{max}, \tag{20}$$

where H(x) is the heavy side step function, $f_Z(x)$ is the PDF function of random variable Z, and $G = min(1 + \alpha, V/V_0')$.

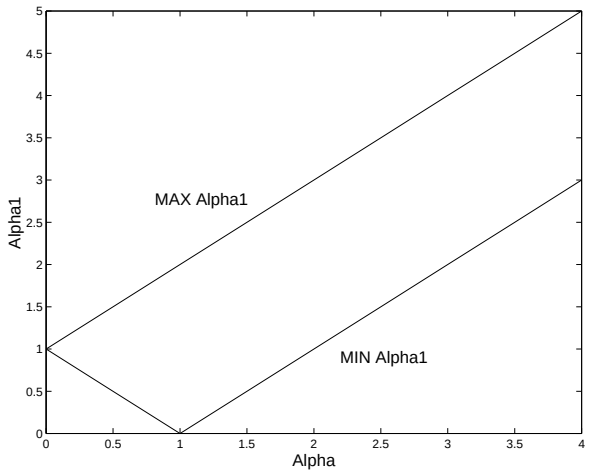

Figure 22: Minimum and maximum values of α_1 as a function of α

$$Pr(Tb_1 \leq t/V_0') = \int_x Pr(Tb_1 \leq t/V_1 = x) f_{V_1}(x) dx \quad (21)$$

$$Pr(Tb_1 \leq t) = \int_x Pr(Tb_1 <= t/V_0') f_{V_0}(x) dx \quad (22)$$

$$Pr(Tb_1 \leq t_1 \cap Tb_1 \leq t_2) =$$
$$\int_x Pr(Tb_1 \leq t_1/V_0') * P(Tb_2 \leq t_2/V_0') \ * \ f_{V_0}(x) dx$$

Figure 21 contains the plot of difference between $Pr(A \leq t_1 \cap B \leq t_1)$ and $Pr(A \leq t_1) * Pr(B \leq t_1)$ as a function of t_1 and t_2 and assuming $r = 76.5m$ and $V = 15m/s$. We see that there is non-zero dependence between the two links sharing a node. For the mobility pattern assumed, we find that the correlation between the two random variable Tb_1 and Tb_2 is 0.172. **From this analysis we conclude that when velocity of nodes is a random variable, common nodes induce dependence between the two trees**.