A Routing Algorithm for Dynamic Multicast Trees with End-to-End Path Length Control

Hiroshi Fujinoki
Kenneth J. Christensen (*** correspondence author ***)
Department of Computer Science and Engineering
University of South Florida
4202 East Fowler Avenue, ENB 118
Tampa, Florida 33620
Email: {*fujinoki, christen*}*@csee.usf.edu*

Abstract

The new Path Length Control (PLC) algorithm establishes and maintains multicast trees which maximize the bandwidth to be shared by multiple receivers and which satisfy the maximum path length bounds for each receiver. The PLC algorithm can be implemented as a distributed algorithm, can tradeoff end-to-end delay and bandwidth consumption, and can be implemented for polynomial time execution. Analysis and simulation show that (a) the PLC algorithm generates multicast trees which consume less bandwidth than those generated by the SPT algorithm while guaranteeing the same shortest path length and (b) consume less bandwidth than trees generated by the Greedy algorithm with only a moderate increase in path length. The PLC algorithm is more flexible and has a lower cost than a combined SPT and Greedy algorithm.

Key Words

Multicast, dynamic multicast, routing algorithm, path length control

1. Introduction

Different from traditional single-cast applications, the paths for a multicasting application are generally established as a tree, called a multicast tree, which contains a single sending node (the sender) and multiple receiving nodes (the receivers). Applications that utilize video and audio distribution with single senders and multiple receivers are good candidates for multicasting. Many of these applications also require a high Quality of Service (QoS) measured in low delay and low loss. Providing QoS support while simultaneously maximizing network utilization can increase the efficiency of use of scare networking resources. A good multicast routing algorithm must be able to be implemented in a distributed fashion and be efficient in execution (i.e., be able to execute in polynomial time) given rapidly changing populations of receivers (see [1]). In addition, a good multicast routing algorithm must be able to control end-to-end delay and bandwidth consumption in order to effectively support real-time video and audio applications (see, for example, [2], [3], and [4]).

A centralized approach to multicast routing should be avoided from the viewpoint of scalability and reliability. Previous research in the area has focused on creating Steiner Trees using heuristic methods (see [5], [6], [7], [8] and [9]). A Steiner tree, or Steiner Minimal Tree (SMT), is the multicast tree that minimizes total path cost. An algorithm to find an exact SMT is NP-complete, hence heuristic algorithms have been studied. Previous research in heuristic SMT algorithms have tried to achieve a tree with cost very close to the ideal SMT, and to do so with minimum storage and computational overhead. For example, the KMB (Kou, Markowsky, and Berman) algorithm generates multicast trees with an average cost of 5% more than the ideal SMT (see [10]). The KMB algorithm builds a heuristic SMT based on the minimum spanning tree algorithm, which requires a centralized approach. This implies that in the KMB algorithm all the nodes in the network have to be reconsidered each time the Steiner tree is to be updated to reflect the latest receiver connectivity. For large networks with dynamic receiver populations, the Steiner Tree approach becomes problematic. This problem gives birth to a serious need for a distributed multicast routing algorithm, where the number of nodes to be involved in updating a multicast tree is some fraction of the total number of nodes in the network. In this regard, the Shortest Path Tree (SPT) (see [11], [12], [13], and [14]) and the Greedy algorithm (see [15]) are the two major algorithms that use a distributed approach. Crowcroft discusses three different types of multicast trees; Steiner trees, center-based trees, and source-based trees to represent multicast routing algorithms (see [16]). However, center-based trees, and source-based trees mean a categorization by the

place where routing operation is initiated, and those trees are generally represented by a shortest path tree or a tree generated by the Greedy algorithm. Therefore, we choose the SPT and Greedy algorithm as other currently available multicast routing algorithms.

The SPT is a multicast tree that minimizes path length between each receiver and the sender, but at the expense of greater total bandwidth consumption than an SMT. In this paper, it is assumed that the root of an SPT is always the sender of a multicast session. Some routing protocols such as PIM (Protocol Independent Multicast) (see [17]) and CBT (Core Based Tee) (see [18] and [19]) do not have this assumption. That is, the root of an SPT is not necessary the sender. In Cho and Breem (see [20]) it is shown that this fixed root approach can be very inefficient and that there is no systematic way to assign the fixed root in such an SPT. For these reasons, this research does not consider those approaches that assume a fixed (non-sender) root.

An SPT can be created very quickly as a collection of shortest paths (e.g., using any single-cast routing algorithm such as Dijkstra's shortest path algorithm implemented in OSPF (see [21])) from each receiver to the sender. We refer to the SPT as the "naïve" multicast tree. It is suggested that implementing a heuristic SMT algorithm in a dynamic environment is very difficult and that a "naïve" multicast tree is robust and can guarantee an upper bound on cost (see [10]). Providing a multicast routing algorithm in a distributed fashion is the first goal in this research.

An implementable multicast routing algorithm must not be NP-complete in complexity. The Steiner tree algorithm has exponential time complexity. Since finding an SMT is a known NP-complete problem, much effort has been put into finding algorithms for a heuristic Steiner tree rather than creating a pure Steiner tree (see [22] and [23]). Most existing heuristic SMT algorithms have time complexity in the order of $O(n^3)$, where n is the number of nodes in a network. Developing a multicast routing algorithm with less than $O(n^3)$ complexity is the second goal of this research.

Minimizing bandwidth consumption and minimizing path length have been studied separately in previous research. For example, heuristic SMT algorithms and the Greedy algorithm can be categorized as those intended to minimize bandwidth consumption, while the SPT algorithm is an algorithm for minimizing path length. However, as will be descried later in the paper, handling these two issues separately results in a tradeoff relationship. The final and major goal of this research is to introduce a multicast routing algorithm that handles these two factors efficiently

and even allows each receiver to flexibly assign priority to one or the other of the two factors according to a preference from each receiver.

In this paper, a new multicast routing algorithm, called the Path Length Control (PLC) algorithm, is developed. The PLC algorithm is efficient, distributed, and supports path length control. The PLC algorithm supports multicast routing with bounded bandwidth consumption and path lengths that cannot be achieved by the SPT or Greedy algorithms or even by a combination of the two algorithms. The remainder of this paper is organized as follows. In Section 2 the SPT and the Greedy multicasting algorithms are reviewed and a tradeoff relationship between the two algorithms is discussed. In Section 3 the PLC algorithm is described. In Section 4, simulation experiments comparing the three algorithms (the SPT, Greedy and PLC algorithms) are presented. The final section is a summary and is followed by references and one appendix.

2. Existing Distributed Multicast Algorithms

In this paper, only multicast applications with a single sender are considered. Figure 1 is an example of such a multicast tree, where the number adjacent to each edge represents a hop count and each node, except the root of the tree, represents a receiver (labeled as *v1*, *v2*, … ,*v13*). We define the total bandwidth (B_{Total}) as the total amount of bandwidth consumed by a multicast tree for a multicast session at a given time. We also define the total path length (P_{Total}) to be the sum of the hops from all the active receivers to the single sender in the multicast tree. A *joining node* is defined to be a node that is being connected to the sender (i.e., to join a multicast session originating from the sender) while *neighbor nodes* are defined as multicast receiver nodes that have already been connected to the sender.

The SPT algorithm always connects a joining node to the sender using one of the shortest paths measured in hop count. Figure 2(a) shows an example of the resulting multicast tree after the SPT algorithm is applied to the graph of Figure 1. The resulting multicast tree has $B_{Total} = 76$ and $P_{Total} = 89$. The Greedy algorithm always connects a joining node to the nearest neighbor node. If such a neighbor node does not exist, or if the sender is the nearest neighbor node, then the joining node is connected to the sender using a shortest path. Figure 2(b) shows an example

of the resulting multicast tree after the Greedy algorithm is applied to the graph of Figure 1. The resulting multicast tree has $B_{Total} = 62$ and $P_{Total} = 94$.

Intuitively, we can see a tradeoff relationship between B_{Total} and P_{Total} from Figure 2(a) and 2(b). The multicast trees generated by the SPT algorithm generally have larger B_{Total} than those created by the Greedy algorithm. This is because the SPT algorithm does not find paths that necessarily share bandwidth by multiple receivers. The Greedy algorithm solves the most serious disadvantage of the SPT algorithm; the amount of bandwidth to be consumed. One problem with the Greedy algorithm is that we may have cases where the path length for a receiver could become unreasonably long (e.g., as shown in Figure 3), since it performs routing based only on the distance to the nearest neighbor. With the Greedy algorithm when a receiver is connected through several intermediate receivers, bandwidth will be wasted if the intermediate receivers leave the multicast tree. This problem was identified first by Waxman (see [15]). Figure 4 shows an example of this problem. Starting at step 1 in Figure 4, receiver nodes are connected in the order of *v2*, *v3*, *v4*, and *v5* as shown in steps 1 through 4. At step 5, if *v2* and *v3* leave the session, *v4* remains connected using the path through *v2* and *v3* even if *v4* could be connected directly to *v5*. If we assume each link has a distance of one, the resulting multicast tree in Figure 4 wastes bandwidth. Waxman's weighted greedy algorithm (see [16]) has been proposed as a solution to this bandwidth waste problem. The weighted greedy algorithm requires the existence of special fixed nodes called "owner nodes". The efficiency of the routing depends on how these owner nodes are assigned. There is no systematic method to assign these nodes. This presents problems for dynamic multicast trees and thus the weighted greedy algorithm is currently not feasible for actual use.

3. The Path Length Control (PLC) Algorithm for Dynamic Multicast Trees

In the previous section, we saw that there exists a relationship between path length and bandwidth consumption and that existing algorithms do not allow tradeoffs between path length and bandwidth consumption. A mechanism to control path length is introduced in a new algorithm, called the Path Length Control (PLC) algorithm. In the PLC algorithm, the capability to control path length is mainly achieved by introducing two "regions", the *trunk region* and *connection region* which are formally defined below. The variables used in the PLC algorithm are:

- $L_{(j,i)}$ is the total path length from a joining node, *j*, to the sender through a neighbor node *i* (*i* is the node ID of the neighbor node).
- M_j is the distance of the shortest path from a joining node, *j*, to the sender.
- $D_{(j,i)}$ is the deviation between a joining node and a neighbor node *i* defined as $D_{(j,i)} = L_{(j,i)} - M_j$ for a neighbor node *i*).
- $T_{MAX(j)}$ is the maximum deviation for a neighbor node, *i*, to be within the trunk region of the joining node, *j*. A *trunk region* is the range of the relative location of a neighbor node *i* from the joining node *j*, such that $D_{(j,i)} \le T_{MAX(j)}$.
- $C_{MAX(j)}$ is the maximum deviation for a neighbor node, *i*, to be within the connection region of the joining node, *j* and $T_{MAX(j)} \le C_{MAX(j)}$. A *connection region* is defined to be the range of the relative location of a neighbor node *i* from the joining node *j*, such that $T_{MAX(j)} \le D_{(j,i)} \le C_{MAX(j)}$.
- $L_{MAX(j)}$ is the maximum allowed path length as specified by each receiver node (i.e., each receiver node can set this parameter).

A joining node will be connected to the sender as a *trunk node* if (a) the joining node cannot find any neighbor node within either its trunk region or its connection region or (b) the joining node finds another trunk node within its trunk region. A *trunk* is the shortest path (or a path with only minor deviation from the shortest path) to the sender that consists only of the trunk nodes with the same trunk ID. If a joining node is connected as a trunk node when condition (a) above applies, a new trunk is established for the joining node with all the intermediate nodes on the trunk becoming trunk nodes. In this case, only the joining node is active, meaning that the multicast data stream is consumed by an application at the node. If case (b) applies, it becomes a part of the existing trunk (therefor the same trunk ID will be assigned to the joining node). When there is no active node connected to a trunk, the trunk will be deleted. If a joining node finds a trunk node within its connection region, but not within its trunk region, the joining node will be connected to the sender as a *non-trunk node* through the trunk node found in the connection region. Note that the way a joining node is connected to the sender (i.e., as a trunk or a non-trunk node) is determined by referencing only to trunk nodes. This is to make sure that an unreasonably long path (e.g., as shown in Figure 3) will not be created. $T_{MAX(j)}$ and $C_{MAX(j)}$ for any joining node, *j*, are determined based on M_j. $T_{MAX(j)} = \lfloor k_1 \cdot M_j \rfloor$ hops

and $C_{MAX(j)} = \lfloor k_2 \cdot M_j \rfloor$ hops where k_1 and k_2 are constants such that $0 \le k_1 \le k_2$. For example, if the shortest distance between a joining node, *j*, and the sender is 10 (i.e., $M_j = 10$), and $k_1 = 0.2$ and $k_2 = 0.4$, then $T_{MAX(j)} = 2$ and $C_{MAX(j)} = 4$. The constants, k_1 and k_2, determine the strictness of a path from the shortest path. The first constant, k_1, determines the probability of a trunk to survive for long period, while k_2 determines the probability for each joining node to be able to connect to an existing trunk. The values of k_1 and k_2 scale as the diameter of the network measured in hops (i.e., k_1 and k_2 are relative parameters and cannot be described in terms of absolute values). In the extreme case when $k_1 = k_2 = 0$ the resulting multicast tree will be a shortest path tree. When $k_1 = k_2 =$ the diameter of the network, the multicast tree will be the same as that generated by the Greedy algorithm.

3.1 Description of the PLC algorithm

The PLC algorithm establishes a multicast tree with a single sender in a directed graph. It is assumed that the shortest path to the sender is determined for each receiver node. The algorithm requires four parameters:

1. The node ID of a joining node, *j*
2. The node ID of the sender, *d*
3. The maximum distance within which a joining node's connecting point is to be chosen. This value is currently to be set to M_j, even though it can be some value less than M_j. Note that M_j specifies the area within which a connecting point will be chosen, while $L_{MAX(j)}$ specifies the acceptable upper bound in terms of path length for resulting path.
4. A weighted directed graph, $G = (V, E)$, where V is a set of all the nodes that represent potential multicast receivers for a multicast session, and E is a set of all logical links between two potential multicast receivers.

Figure 5 shows the PLC algorithm. The algorithm uses four queues, Q, Q_T, Q_{NT}, and Q_{Temp}. The queues hold all the neighbor nodes found within *M* hops from the joining node, *j*, all trunk nodes within *M* hops from *j*, all non-trunk nodes within *M* hops from *j*, and the set of non-trunk neighbor nodes which belong to the same trunk for a particular neighbor trunk node, respectively. Similar to $T_{MAX(j)}$ and $C_{MAX(j)}$, $L_{MAX(j)}$ is determined based on the shortest

path, M_j. $L_{MAX(j)} = \lfloor (k_3 + 1) \cdot M_j \rfloor$ hops where k_3 is a variable and $0 \le k_3 \le k_2$ (k_2 is defined in the previous section and serves as the threshold for $L_{MAX(j)}$). For example, when M_j = 10, k_2 = 0.4, and k_3 = 0.3, then $L_{MAX(j)}$ = 13 hops. The major motivation behind having $L_{MAX(j)}$ is to provide the flexibility to set the upper bound on path length at each joining node. This is since both $T_{MAX(j)}$ and $C_{MAX(j)}$ are assumed to be the same among all the joining nodes for routing a particular multicast session. Each joining node can set its own k_3, and the upper bound of the resulting path length is determined by $MIN((C_{MAX(j)} + M_j), L_{MAX(j)})$. Thus, $L_{MAX(j)}$ is used to achieve stricter path length in case the path determined by $T_{MAX(j)}$ and $C_{MAX(j)}$ pair does not satisfy a specified end-to-end delay requirement. In such a case, k_3 could be adjusted dynamically taking the delay requirement as a feedback from the paths generated by the $T_{MAX(j)}$ and $C_{MAX(j)}$ pairs.

3.2 Implementation of the neighbor-finding procedure

The first step in the algorithm (line 1 in Figure 5) is responsible for finding potential neighbor nodes for connection. To achieve this step, the joining node can broadcast a polling message while neighbor nodes send reply messages back to the joining node to let their existence and their distance from the sender be known to the joining node. Table 1 shows the fields that can be included in the polling and reply messages. Before the polling message is broadcast, the TTL field is set to M_j at the joining node and is decreased by one at each hop during the broadcast. Similarly, the HPC field is set to zero and is incremented by one at each hop. The polling message will not be broadcast farther when TTL = 0. Each time a joining node tries to join the same multicast session, the PID field will be assigned a different number. The JID and SID fields are self-explanatory. At a neighbor node, the HPC field in the polling message is copied into the DJN field in the reply message when a neighbor node receives the reply message. The DJN field is used to tell the joining node the neighbor node's distance from the joining node.

Using the reply messages returned by neighbors, the PLC algorithm is initiated by a joining node, and it consists of procedures at both a joining node and its neighbor nodes. When a joining node wants to join a multicast session, it broadcasts a polling message and then waits for reply messages from its neighbor nodes. The waiting time is defined as, $T_{wait} = 2 \cdot (T_{pr(M_j)}) + T_{oh(M_j)}$ where $T_{pr(M_j)}$ is the expected propagation delay for a polling message to travel

the distance of M_j. The variable $T_{oh(M_j)}$ is the total overhead delay to handle the polling message at each hop during the broadcast. After the joining node waits for reply messages for the duration of T_{wait}, it starts finding a neighbor node for connection. If no reply message is received during T_{wait}, the joining node will be connected to the sender using the shortest path, which then becomes another trunk. An implementation description of the PLC algorithm is given in the appendix.

3.3 Examples of PLC routing

In order to see how the PLC algorithm works, let us see some examples using Figure 1 as the base network. First, let us assume that there is not yet a node connected to the sender in Figure 1, and that *v6* wants to connect to the sender. Since *v6* is the first node in the multicast session, it will be connected to the sender using the shortest path, creating a trunk between *v6* and the sender. Node *v6* and any nodes on the new trunk (in this example, *v5*) will become a trunk node for the trunk. Now, suppose that *v8* wants to join the multicast session after *v6*. Assuming that *v8* receives reply messages from *v5* and *v6* then *v8* knows that *v6* is closer than *v5* and that the length of the path through *v6* is 12 = 3 + 7 + 2, and $D_{(8,6)} = L_{(8,6)} - M_8 = 12 - 9 = 3$. Suppose that $T_{MAX(8)}$ and $C_{MAX(8)}$ are 2 and 4, respectively then since a trunk node *v6* is not within the trunk region of *v8* ($D_{(8,6)} > T_{MAX(8)}$), *v8* can not be connected to *v6* as a trunk node on the same trunk. However, the joining node *v8* can be connected to *v6* as a non-trunk node ($D_{(8,6)} < C_{MAX(8)}$), assuming $L_{(8,6)} \le L_{MAX(8)}$. Note that *v6* should be considered before *v5*, since *v6* is closer to the joining node than *v5* (Figure 6(a) shows this).

For another example, let us assume that *v11* wants to join the multicast session after *v6* and *v8* are connected (i.e., after Figure 6(a)), and that node *v6* and *v8* send a reply message to *v11*. Assuming that $T_{MAX(11)}$ and $C_{MAX(11)}$ are 2 and 4, respectively, *v11* knows that *v6* is not within its connection region ($D_{(11,6)} = L_{(11,6)} - M_{11} = 16 - 11 > C_{MAX(11)}$). Thus, *v11* cannot be connected to *v6* even as a non-trunk node. As for *v8*, it is not within the connection region of *v11* ($D_{(11,8)} = L_{(11,8)} - M_{11} = 16 - 11 > C_{MAX(11)}$). Even if *v8* were within the connection region of *v11*, *v11* cannot be connected to *v8* since *v8* is not a trunk node. As a result, *v11* does not see any trunk node within its trunk or connection region. Node *v11* will be connected to the sender by its shortest path, creating a new trunk. Figure 6(b)

shows the resulting PLC multicast tree for this situation. In case of link of node failure, only the nodes located downstream of the point of the topology change are affected with the PLC algorithm. This is not the case for the distributed Bellman-Ford and Dijkstra algorithms which require changes in the routing table of all the nodes in a network for any topological change. There are four failure cases to be considered: (1) a path becomes unavailable due to node or link failures, (2) the current path length increases due to insertion of node(s), (3) the current path length decreases due to removal of node(s), and (4) the shortest path length has been changed. For the first three cases, the first downstream node after the point of failure or path length change can send notification messages to its all downstream nodes. For link failure, the notification message is sent after the first downstream node finds a new point of connection. Each of the downstream nodes makes a decision regarding whether it should stay on the path notified by the first downstream node. In the case it determines that the new path is not acceptable, it initiates its own PLC algorithm seeking a new path. Case (4) is where another path is created which is shorter than the previous shortest path (but, the new path is not the one currently being used). In this case, there will not be any immediate changes for the currently connected receivers. The new shortest path will take effect for subsequent joining nodes, and a transition takes place as new receivers join the multicast tree.

3.4 Analysis of the PLC algorithm

The motivation for introducing the trunk and connection regions in the PLC algorithm is to introduce flexibility in the routing of multicast connections and to avoid increase in path lengths. Figure 7 illustrates the routing characteristics of the PLC algorithm. Two parameters control path length and bandwidth consumption, whose coverage is shown by the shaded area, which occupy two dimensions in the figure. Figure 7 also illustrates the characteristics of the SPT and Greedy algorithms. Note that both the SPT and Greedy algorithm are represented by a single dot in the figure at the upper left and the lower right corner of the square-shaped graph, respectively. The SPT and Greedy algorithms could be combined with a certain threshold to switch between the two algorithms. Each multicast application (or each joining node) can set a threshold for its longest tolerable path length using the Greedy algorithm so that when the resulting path by the Greedy algorithm exceeds the threshold, the SPT algorithm will be used instead. The routing characteristics of this combined approach is represented by the dashed diagonal line in Figure 7 connecting the SPT and Greedy algorithms. This line occupies a single dimension in the figure and the SPT

and Greedy algorithm cover only single points in the figure. This is less flexible when compared to the two dimensions covered by the PLC algorithm.

The PLC algorithm avoids increases in the path length by using two shortest paths; one between the sender and a trunk node (i.e., paths in a trunk region), and the other between the trunk node and the joining node (i.e., the paths in a connection region). By creating a path using the two shortest paths, some of the bandwidth that would be wasted by the Greedy algorithm can be eliminated, while any increase in path length will also be minimized. Priority between bandwidth consumption and path length can be controlled by the two parameters; the upper bound for the trunk region and for the connection region. This achieves less bandwidth consumption (or the same in its worst case) than the SPT algorithm when the PLC algorithm is configured as "most strict" (or, PLC-MS) to have zero tolerance for the deviation from both trunk and connection regions while it achieves the same shortest paths as the SPT algorithm. We prove that the PLC-MS algorithm avoids some of the bandwidth consumption of the SPT algorithm, while it achieves the same path length as the SPT algorithm.

Theorem: When a joining node has to choose a path to the sender, assume that on average α ($\alpha \geq 1$) paths that are all shortest paths to the sender exist. Let β ($0 \leq \beta \leq \alpha$) be the number of the α shortest paths that already have an active receiver connected to the sender. Then, the PLC-MS algorithm has an average bandwidth consumption less than that of the SPT algorithm by $\left(1-\frac{\beta}{\alpha}\right)\cdot\theta$, where θ ($\theta > 0$) is the average amount of bandwidth to be saved by choosing any one of the β shortest paths.

Proof: The PLC-MS algorithm allows only zero deviation for both trunk and connection regions. Thus, a chosen path will be accepted if and only if it is one of the shortest paths from a joining node to the sender and thus the PLC-MS algorithm is guaranteed to have an average path length as short as that of SPT algorithm (which is the theoretical lower bound for possible shortest path length). For bandwidth consumption, suppose that a joining node has α different shortest paths to the sender on average, and that β of the α different shortest paths are available as a whole or a part for the joining node. The SPT algorithm always connects a joining node to the sender using a fixed shortest path to the sender and it is possible that the fixed shortest path is one of the β shortest paths. Therefore, the probability for the SPT algorithm not to choose one of the β different shortest paths is $\left(1-\frac{\beta}{\alpha}\right)$. Assuming that θ is the average amount of bandwidth that could be otherwise saved by choosing one of the β different shortest paths, the

average amount of bandwidth to be saved is $\left(1-\frac{\beta}{\alpha}\right)\cdot\theta$. The PLC algorithm always chooses one of the β shortest paths. Thus, $\left(1-\frac{\beta}{\alpha}\right)\cdot\theta$ is the amount of bandwidth to be saved by the PLC algorithm. **QED.**

3.5 Complexity of the PLC algorithm

In this subsection, the complexity of the PLC algorithm is analyzed and compared with that of the SPT and Greedy algorithms. The total cost to implement each routing algorithm can be subdivided into the following three categories:

1. **Initialization operations -** the cost required in a whole network only once when a network is first configured to set-up information needed for a routing algorithm to subsequently perform its routing task.
2. **External operations - t**he cost required at a joining node each time it tries to find a neighbor node as its connecting point in order to collect information from outside of the joining node.
3. **Internal operations -** the same definition above applies, with the only exception that the cost is for the operations inside of the joining node.

In the following analysis, $C_{SPT(INITIAL)}$, $C_{Greedy(INITIAL)}$, and $C_{PLC(INITIAL)}$ are the cost for initialization, respectively, while $C_{SPT(NON-INITIAL)}$, $C_{Greedy(NON-INITIAL)}$, and $C_{PLC(NON-INITIAL)}$ are the sum of the external and the internal costs. The last two costs take place each time a joining node joins and are hence combined. Table 2 summarizes the costs for the three algorithms.

For the SPT algorithm, the shortest paths from the sender to each joining node must be determined before routing operations take place. Determining the shortest paths for each joining node should be performed only once when the network is first configured. Assuming that path length is meant to be the hop count from a joining node to the sender, it takes $O(n^2)$ time applying Dijkstra's SPF algorithm (or at least $O(n^2)$ using Bellman-Ford's algorithm). However, once the shortest path has been determined for each joining node, there is no routing operation required as an external operation and only a constant time to retrieve the fixed connecting point is required as its internal operation, since each joining node can be connected to the sender based on the shortest path determined during the

network initialization. As a result, $C_{SPT(INITIAL)}$ and $C_{SPT(NON-INITIAL)}$ for SPT algorithm can be expressed by the following formulas, assuming that the terms in formula (2) represent the external cost, and internal cost, respectively.

$$C_{SPT(INITIAL)} = O(n^2) \quad (1)$$

$$C_{SPT(NON-INITIAL)} = (\text{no cost required}) + O(1) = O(1) \quad (2)$$

which shows that the cost for initialization and non-initialization of SPT algorithm are in $O(n^2)$ and $O(1)$, respectively.

In the case of the Greedy algorithm, a joining node needs to find its nearest neighbor as its connecting point. To determine the nearest neighbor node, a shortest path tree algorithm will be performed, and it again takes time $O(n^2)$ using Dijkstra's SPF algorithm. However, the Greedy algorithm does not require any information to be set-up when a network is first configured, since all the only task it requires is to find the nearest neighbor node when a joining node wants to join a multicast session. After all the neighbor nodes located within the joining node's distance to the sender are found, it has to find the neighbor node with the shortest distance from the joining node, which can be done by linear search in $O(n)$. Similar to the previous cost analysis for the SPT algorithm, the two costs for the Greedy algorithm are:

$$C_{Greedy(INITIAL)} = (\text{no cost required}) \quad (3)$$

$$C_{Greedy(NON-INITIAL)} = O(n^2) + O(n) = O(n^2) \quad (4)$$

which shows that the cost for initialization and non-initialization of the Greedy algorithm are zero (no cost) and $O(n^2)$, respectively.

The PLC algorithm requires the same cost as the SPT algorithm for initialization (the cost is in $O(n^2)$), since the PLC algorithm requires that the shortest path for each joining node to be determined before PLC can start routing. The external cost of the PLC algorithm is the same as that of the Greedy algorithm (i.e., in $O(n^2)$) assuming Dijkstra's SPF algorithm, since the PLC algorithm needs to know all the neighbor nodes located within a joining node's distance to the sender. The first step in the PLC algorithm (line 1 in Figure 5) is responsible for this cost. Each of lines 2 and 4 costs $O(n)$ where n is the number of nodes found at line 1, since it requires a linear search on queue Q. Line 3 and 5 performs sorting on queue Q_T and Q_{NT}, each of which takes $O(n \log n)$ time. The while loop

from lines 6 to 20 repeats as many times as the number of trunk nodes found at line 2. Inside of the while loop is another while loop (lines 11 to 18) that repeats as many times as the number of non-trunk nodes for a trunk node. Each statement in the second loop (lines 12 to 17) takes constant time. As a result, the total worst case cost for the first loop is in the order of $O(n^2)$. The time complexity is a constant for the neighbor nodes of the PLC algorithm, since a neighbor node retrieves SID and PID fields from the received polling message, and simply compares them to the ones held at the neighbor node. Processing JID, HPC, and TTL costs some constant time. Similarly, calculating and setting each parameter of NID, DJN, DIS, TNB, and TID costs some constant. As a result, the two costs of the PLC algorithm are:

$$C_{PLC(INITIAL)} = O(n^2) \tag{5}$$

$$C_{PLC(NON-INITIAL)} = (n^2) + O(n^2) = O(n^2) \tag{6}$$

Table 2 summarizes the costs, from the above analysis, for the SPT, Greedy, combined SPT/Greedy, and PLC algorithms. Comparing the PLC and SPT algorithms, the initial cost complexities are the same, but the non-initial cost is lower for the SPT algorithm. Comparing the PLC and Greedy algorithms, the initial cost complexity is lower for the Greedy algorithm, but the non-initial costs are the same. For a combined SPT/Greedy algorithm, all costs are clearly the same as the PLC algorithm.

4. Evaluation of the PLC Algorithm

To compare the performance of the SPT, Greedy, and PLC algorithms, a simulation model was developed. The simulation model was implemented as a time-based simulation where at each discrete clock tick statistics are collected. All continuously distributed random time values are truncated to their corresponding integer values. The major variables in the simulation model are network size and multicast receiver behavior. Two configurations of the PLC algorithm were tested, PLC-MS (PLC-Most Strict) and PLC-S (PLC-Strict). PLC-MS has zero tolerance for the deviation from both trunk and connection region (i.e., $T_{MAX(i)} = 0$ and $C_{MAX(i)} = 0$ for any *i*), while PLC-S has zero tolerance only for the trunk region. The random network model introduced by Batsell was used (see [22]). This network model assumes a grid-patterned square network, where a node representing a multicast receiver site is located only at each intersection of the grid in a network. Thus, the network model is a non-hierarchical, flat

structure. It is predicted that the future Internet will tend towards a non-hierarchical, flat network as the number of domains increases (see [24] for more details). That is, the Internet will grow in "width", but not "height" and thus a good multicast algorithm should be one that can efficiently route in flat networks. A more detailed study on topological structures in internetworking environment such as node degree, and the number of biconnected components can be found in [25].

The number of nodes in a network represents the network size for a random square network. The number of nodes in a network is assumed to be N^2, where N is the number of nodes in each edge of a square network. To assign links between nodes in the random networks, the link assignment model proposed by Waxman (see [15]) is applied, in which the probability of there being an edge between nodes u and v is given by:

$$P(u,v) = \alpha \cdot e^{-d(\beta \cdot L)} \tag{7}$$

where d and L are the Euclidean distance and maximum possible link distance between the nodes u and v. The parameters, α, $\alpha > 0$, and β, $\beta \leq 1$, control the average link distance between nodes u and v.

4.1 Modeling receiver behavior

A receiver's activities are assumed to be either a) joining a multicast tree, or b) disconnecting itself from the multicast tree. The period between the beginning of a connection and the subsequent disconnection is defined to be the session holding time. Poisson arrivals are assumed for session join requests and both exponential and Bounded-Pareto (see [26]) distributed session holding times are assumed. For data sets 1 through 5, exponential distributed session holding times are modeled and for data sets 6 through 10, Bounded-Pareto distributed session holding times are modeled. We choose Bounded-Pareto distribution to simulate heavy-tail effect in session holding time (see [27]). The probability mass function for the Bounded-Pareto, $B(p,k,\alpha)$, is defined as,

$$f(x) = \frac{\alpha \cdot k^{\alpha}}{1 - \left(\frac{k}{p}\right)^{\alpha}} \cdot x^{-\alpha-1} \tag{8}$$

where k is the minimum duration of a session, p the maximum duration, and α a shape parameter. Bounded-Pareto random variables, X, can be generated using the inverse transform method as,

$$X = \left(\frac{-\left(U \cdot p^{\alpha} - U \cdot k^{\alpha} - p^{\alpha}\right)}{p^{\alpha} \cdot k^{\alpha}} \right)^{-\frac{1}{a}} \tag{9}$$

where, U is uniformly distributed random variable from 0 to 1.

The simulation model measures the average bandwidth consumption per minute (A_{BWC}), and the average path length per connection (A_{PL}) for a simulation of T minutes.

$$A_{BWC} = \frac{1}{T} \sum_{s=0}^{T} b_s \tag{10}$$

and

$$A_{PL} = \frac{1}{T} \sum_{s=0}^{T} \left(\frac{1}{n_s} \sum_{i=0}^{n} l_i \right) \tag{11}$$

where b_s is the amount of bandwidth consumed in the network at s clock tick and l_i is the path length of the receiver node i at clock tick s (0 if it is not connected), while n_s is the number of receiver nodes joined at the s clock tick and n is the number of nodes in the network.

4.2 Experimental Design

To evaluate the three routing algorithms (SPT, Greedy, and PLC) a range of network sizes, network connectivities, session holding times, and session initiation arrival rates were modeled. Table 3 summarizes the key parameters, in terms of data sets, for the experiments. Average node utilization is defined to be the ratio of session holding time to total elapsed time for a receiver node. Data sets 1 and 6 simulate long-lived receiver sessions. The parameters for these data sets are taken from statistics collected for the NASA STS-63 space shuttle audio sessions from February 3 to February 11, 1995 on the Mbone (Multicast Backbone), while those for the short-lived session (data set 2 and 7) are based on the statistics collected for Free BSD Lounge also on Mbone (see [28]). Data sets 3 and 8 are artificially created to see the effects from longer inter-session time (session request interarrival time is doubled from data set 1 and 6), while data sets 4 and 9 model shorter session holding time (session holding time halved from data set 1 and 6). Data sets 5 and 10 were created to see the effects of very low node utilization (session request interval time multiplied by six while session holding time halved from data set 1 and 6). The three

parameters for Bounded-Pareto distribution are chosen as follows: first, α, was chosen to be 0.8 to achieve high variance in resulting session holding time (standard deviations range from 95 minutes to 386 minutes in the five data sets). The maximum session holding time, *p*, was fixed to be 2880 minutes (two days), meaning that we assume a receiver will not stay connected more than two days. For each data set, the minimum session holding time, *k*, was adjusted to achieve the same mean as its corresponding set in exponential distribution (see Table 3 for actual values). We chose *k* to control the mean where the mean for a Bounded Pareto random variable *X* is,

$$E[X] = \frac{\alpha \cdot k^{\alpha} \cdot \left(k^{1-\alpha} - p^{1-\alpha}\right)}{(\alpha - 1) \cdot \left(1 - \left(\frac{k}{p}\right)^{\alpha}\right)}. \quad (12)$$

For all the experiments, session requests are assumed to be a Poisson stream. For each of the ten experiments, three different network sizes of N^2 = 100, 400, and 900 were used to model small, medium, and large networks. A simulation time of *T* = 28800 minutes (representing twenty days) was used for all experiments.

5. Experimental Results

Table 4 shows the results from experiments 1 through 5 for average bandwidth consumption per minute (A_{BWC}) and average path length per connection (A_{PL}), respectively. The Greedy algorithm results in the longest A_{PL} in the five experiments and the SPT algorithm results the shortest. The Greedy algorithm results in path lengths 6% to 24% longer than those of the SPT algorithm. From the results of experiment 1, it can be seen that the PLC-MS algorithm reduces bandwidth consumption by about 8% when compared to the SPT algorithm, while the PLC-MS algorithm achieves the same theoretical lower bound for A_{PL} as achieved by the SPT algorithm. For the same experiment, PLC-S algorithm results in the lowest bandwidth consumption of all three algorithms; 13% or 14% less consumption when compared to the SPT algorithm, while its tradeoff is a 4% to 7% increase in average path length. The results from experiments 3 and 4 show that the performance of the three algorithms in terms of both A_{BWC} and A_{PL} are similar to that of experiment 1. The only exception is that the Greedy algorithm results in larger bandwidth consumption (2% to 17% larger bandwidth consumption than the SPT algorithm) in contrast to the results from experiments 1 and 2 where the Greedy algorithm results in 3% to 21% lower bandwidth consumption. Note that in both cases, PLC-MS

achieves the same theoretical lower bound achieved by the SPT algorithm for A_{PL} , but with lower bandwidth consumption. Similar to the result from experiment 1, PLC-S results in the lowest bandwidth consumption. Figures 8 and 9 show the results from experiment 4. Figures 10 and 11 show the results from experiment 5. The results from experiments 6 through 10 (the experiments with the heavy-tailed session holding times) are shown in Table 5. This group shows similar results as seen in experiments 1 through 5 (Table 4) with the exception that Greedy algorithm results in larger bandwidth consumption in a large network ($N^2 = 900$) for experiment 9 and longer path length in a large and small ($N^2 = 100$) network for experiments 6 and 7, respectively.

5.1 Observations from the experiments

The following observations are made from the experimental results for both the Exponential and Bounded-Pareto distributed session holding times:

1. For networks with moderate and high node utilization, the SPT algorithm minimizes path length for each connection, but wastes some network bandwidth. The Greedy algorithm achieves less bandwidth consumption than the SPT algorithm, but with longer path length.
2. For networks with low average node utilization the Greedy algorithm is inefficient in terms of both bandwidth consumption and path length.
3. The PLC-MS algorithm always achieves the same average path length, but with less bandwidth consumption, as that of the SPT algorithm except for a network with very low node utilization (experiment 5). When average node utilization is very low, bandwidth consumption by the PLC-MS algorithm is about the same as that of the SPT algorithm.
4. For networks with moderate average node utilization (experiments 3 and 4), the PLC-MS algorithm demonstrates the best performance. The PLC-MS algorithm consumes less bandwidth than both the SPT and Greedy algorithms, while it achieves the shortest possible path length (i.e., as short as that of the SPT algorithm).
5. The PLC-S algorithm demonstrates the least bandwidth consumption of the three algorithms tested, except for a network with very high average node utilization (experiment 2). When average node utilization is

very high, bandwidth consumption by the PLC-S algorithm is about the same as that of the Greedy algorithm. For resulting path lengths, the PLC-S algorithm is always around the mid-point of the SPT and Greedy algorithms.

6. The performance of each routing algorithm depends more on node utilization than on the average inter-arrival and session holding times.
7. The results from experiments 6 to 10 show that the Greedy algorithm performs poorly given heavy-tailed session holding times.

6. Summary

In this paper, a new dynamic multicast routing algorithm called the Path Length Control (PLC) algorithm has been developed and evaluated. The PLC algorithm is a distributed, polynomial time algorithm that allows a tradeoff between path length and bandwidth consumption. By defining trunk and connection regions, end-to-end path length, and hence delay, can be controlled. From simulation experiments we have seen a tradeoff relationship between the SPT and Greedy algorithms. The SPT algorithm guarantees the shortest path for each receiver, while it generally results in larger bandwidth consumption than the Greedy algorithm. On the other hand, the Greedy algorithm generally results in larger average path length, but usually with less bandwidth consumption than the SPT algorithm. For networks with low average node utilization, the SPT algorithms works well in terms of bandwidth consumption and path length (which is always the shortest), while the Greedy algorithm is inefficient in both bandwidth consumption and path length. The complexity of the PLC algorithm is $O(n^2)$ for both the initial and non-initial costs, while the SPT algorithm has $O(n^2)$ and $O(1)$, and the Greedy algorithm has no initial cost and $O(n^2)$ cost for non-initial operations. The overall cost for the PLC algorithm is the same order as that of combined cost of the SPT and Greedy algorithms. In addition, (a) for a delay sensitive multicast application, PLC-MS can save bandwidth which would be otherwise wasted by the SPT algorithm, while it achieves the same shortest path length as that of the SPT algorithm and (b) for multicast applications in which delay is not the highest priority, the PLC-S algorithm can achieve the lowest bandwidth consumption with only a moderate increase in path length.

Acknowledgments

We thank the anonymous referees for their comments that have improved the quality of this paper.

References

[1] L. Wei and D. Estrin, "The Tradeoffs of Multicast Trees and Algorithms," *Proceedings of the Fourth International Conference on Computer Communications and Networks*, pp. 150 - 157, September 1995.

[2] V. Kompella, J. Pasquale, and G. Polyzos, " Multicasting for Multimedia Applications," *Proceedings of IEEE INFOCOM*, pp. 2078 - 2085, 1992.

[3] Q. Zhu, M. Parsa, and J. Garcia-Luna-Aceves, "A Source-Based Algorithm for Delay-Constrained Minimum-Cost Multicasting," *Proceedings of IEEE INFOCOM*, pp. 377 - 385, 1995.

[4] M. Hofmann, "A Generic Concept for Large-Scale Multicast," *International Zurich Seminar on Digital Communication*, Broadband Communications: Lecture Notes in Computer Science, No. 1044, Editor: B. Plattner, Springer Verlag, February 1996.

[5] E. Gilbert, and H. Pollak, "Steiner Minimum Trees," *SIAM Journal on Applied Mathematics*, Vol. 16, No. 1, pp. 1 - 29, January 1968.

[6] R. Karp, Reducibility among Combinatorial Problems. Plennum Press, New York, 1972.

[7] P. Winter, "Steiner Problem in Networks: A Survey," *Networks*, Vol. 17, No. 2, pp. 129 - 167, 1987.

[8] K. Bharath-Kumar, and J. Jaffe, "Routing to Multiple Destinations in Computer Networks," *IEEE Transactions on Communications*, Vol. COM-31, No. 3, pp. 343 - 351, March 1983.

[9] G. Rousas, and I. Baldine, "Multicast Routing with End-to-End Delay and Delay Variation Constraints," *Proceedings of IEEE INFOCOM*, pp. 353 - 360, 1996.

[10] M. Doar and I. Leslie, "How Bad is Naïve Multicast Routing?," *Proceedings of IEEE INFOCOM*, pp. 82 - 89, 1993.

[11] S. Deering, and D. Cheriton, " Multicast Routing in Datagram Internetworks and Extended LANs," *ACM Transactions on Computer Systems*, Vol. 8, No. 2, pp. 85 - 111, May 1990.

[12] S. Deering, "Host Extensions for IP Multicasting," RFC-1112, August 1989.

[13] C. Noronha, Jr., and F. Tobagi, "Optimum Routing of Multicast Streams," *Proceedings of IEEE INFOCOM*, pp. 865 - 873, 1994.

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

[15] B. Waxman, "Routing of Multipoint Connections," *IEEE Journal of Selected Areas in Communications*, Vol. 6, No. 9, pp. 1617 - 1622, December 1988.

[16] C. Diot, W. Dabbous, and J. Crowcroft, "Multipoint Communication: A Survey of Protocols, Functions, and Mechanisms," *IEEE Journal on Selected Areas in Communications*, Vol. 15, No. 3, pp. 277 - 290, April 1997.

[17] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C. Liu, and L. Wei, "Protocol Independent Multicast (PIM): Protocol Specification," *Internet Draft RFC*, January 1995.

[18] A. Ballardie, P. Francis, and J. Crowcroft, "Core Based Tree (CBT)," *Proceedings of the ACM SIGCOMM*, pp. 85 - 95, September 1993.

[19] C. Shields, and J. Garcia-Luna-Aceves, "The Ordered Core Based Tree Protocol," *Proceedings of IEEE INFOCOM*, pp. 884 - 891, 1997.

[20] J. Cho, and J. Breen, "Analysis of the Performance of Dynamic Multicast Routing Algorithms," submitted to *ICCCN*, June 1998. URL: *http://cs-tr.cs.cornell.edu:80/Dienst/UI/1.0/Display/xxx.cs.NI/9809102*.

[21] J. Moy, "Multicast Extensions to OSPF," *RFC-1584*, March 1994.

[22] S. Batsell, "Performance and Resource Cost Comparisons for CBT and PIM Multicast Routing Protocols," *IEEE Journal on Selected Areas in Communication*, Vol. 15, No. 3, pp. 304 - 315, April 1997.

[23] J. Behrens and J. Garcia-Luna-Aceves, "Distributed, Scalable Routing Based on Link-State Vectors," *Computer Communications Review*, Vol. 24, No. 4, pp. 136 - 147, October 1994.

[24] R. Govindan, and A. Reddy, "An Analysis of Internet Inter-Domain Topology and Route Stability," *Proceedings of IEEE INFOCOM*, pp. 850 - 857, 1997.

[25] K. Calvert, E. Zegura, and S Bhattacharjee, "How to Model an Internetwork," *Proceedings of IEEE INFOCOM*, pp. 594 - 602, 1996.

[26] M. Crovella, M. Harchol-Balter, and C. Murta, "Task Assignment in a Distributed System: Improving Performance by Unbalancing Load," *Technical Report BUCS-TR-1997-018*, October 1997.

[27] V. Paxson, and S. Floyd, "Wide-Area Traffic: The Failure of Poisson Modeling," *IEEE/ACM Transactions on Networking*, Vol.3, No.3, pp. 226 - 244, June 1995.

[28] K. Almeroth, and M. Ammar, "Multicast Group Behavior in the Internet's Multicast Backbone (Mbone)," *IEEE Communications Magazine*, Vol. 35, No. 6, pp. 124 - 129, June 1997.

Appendix - Implementation of the PLC Algorithm

This appendix outlines a step-by-step implementation of the PLC algorithm. Definitions for each of the symbols used in this appendix are given in Table 1. The implementation for a joining node is:

Step1: Set *timer* = 0 and broadcast the polling message with all five fields (JID, HPC, TTL, SID, and PID) set. See Table 1 for their initial values.

Step 2: Receive reply messages from neighbor nodes until the $timer = T_{wait}$.

Step 3: The joining node stops receiving reply messages. Separate the reply messages of trunk nodes from those of non-trunk nodes. If a trunk node is not found, the joining node terminates and is connected to the sender using the shortest path.

Step 4: Sort the reply messages from trunk nodes in the ascending order of their distance from the joining node using the DJN field in the received reply messages.

Step 5: Repeat step 4 for the reply messages from the non-trunk nodes.

Step 6: For the reply messages from the trunk node with the shortest distance from the joining node, find all the non-trunk nodes that belong to the same trunk as does the trunk node.

Step 7: From the non-trunk nodes found at step 6, find the one with the shortest distance from the joining node as the *candidate* for the connection point from the joining node. If a non-trunk node cannot be found in step 6, or if the trunk has the shortest distance from the joining node, choose the trunk node found at step 6 as the candidate of the connection point from the joining node.

Step 8: Check the length of the resulting path through the candidate found at step 7. If the path length is less than the upper bound specified by an application running at the joining node (i.e., $L_{(j,canditate)} \le L_{MAX(j)}$, where *j* is the node ID of the joining node, and *candidate* the node id of the candidate node). If the length exceeds the upper limit (i.e., $L_{(j,canditate)} > L_{MAX(j)}$), choose the trunk node with the next shortest distance to the joining node at step 4 and repeat the steps until the one that satisfies the conditions is found. If the search for trunk nodes is exhausted, then the joining node will be connected to the sender using the shortest path.

The implementation for the neighbor nodes is:

Step1: Receive a polling message from a joining node.

Step 2: Copy the HPC field in the polling message into the DJN field of the reply message.

Step 3: For the received polling message, decrement the received TTL value and increment the HPC value. Replace the original TTL and HPC field with the new values.

Step 4: For the SID field in the received polling message, check all the senders to which this neighbor node is connected. If one matches with the SID field in the received polling message, send back the reply message to the joining node. If no matching sender ID is found, the reply message will not be returned to the joining node.

Step 5: If TTL > 0, broadcast the polling message.

List of Figures and Tables

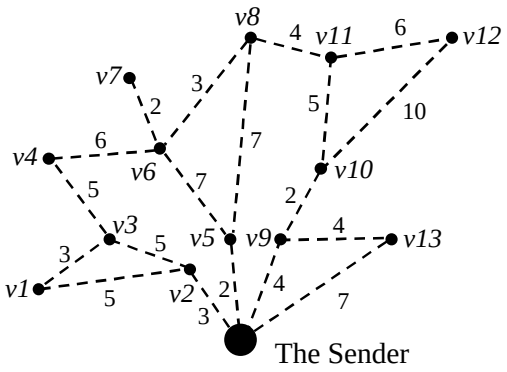

Figure 1 - A graph with multiple receiving vertices and one sending vertex

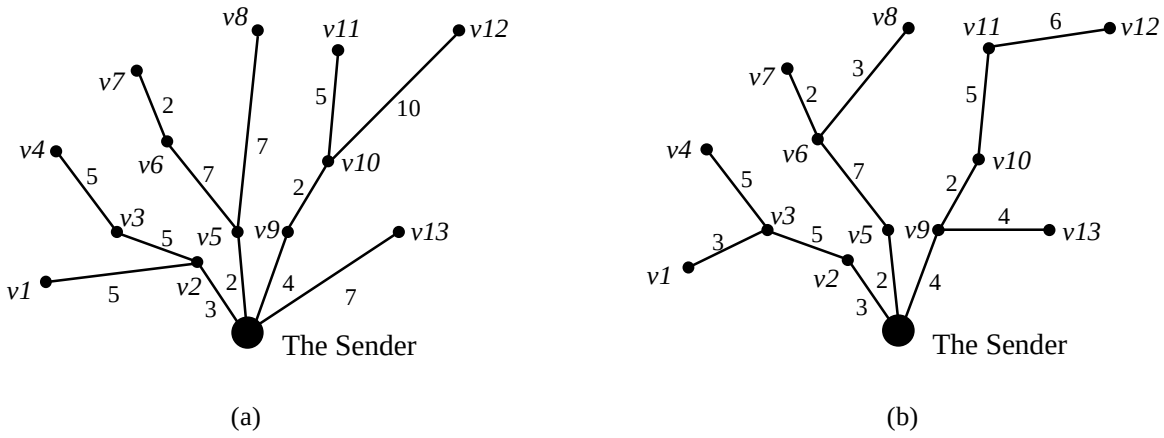

Figure 2 - The resulting graphs after (a) the SPT and (b) Greedy algorithms are applied

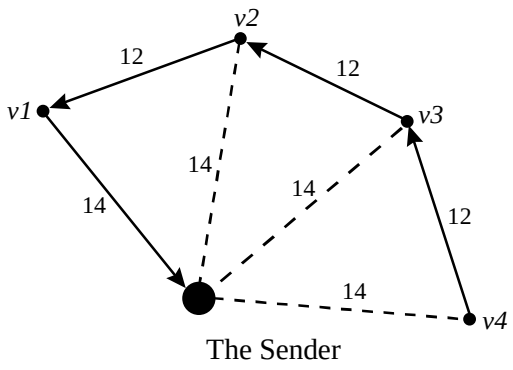

Figure 3 - An example of a worst case path resulting from the Greedy algorithm

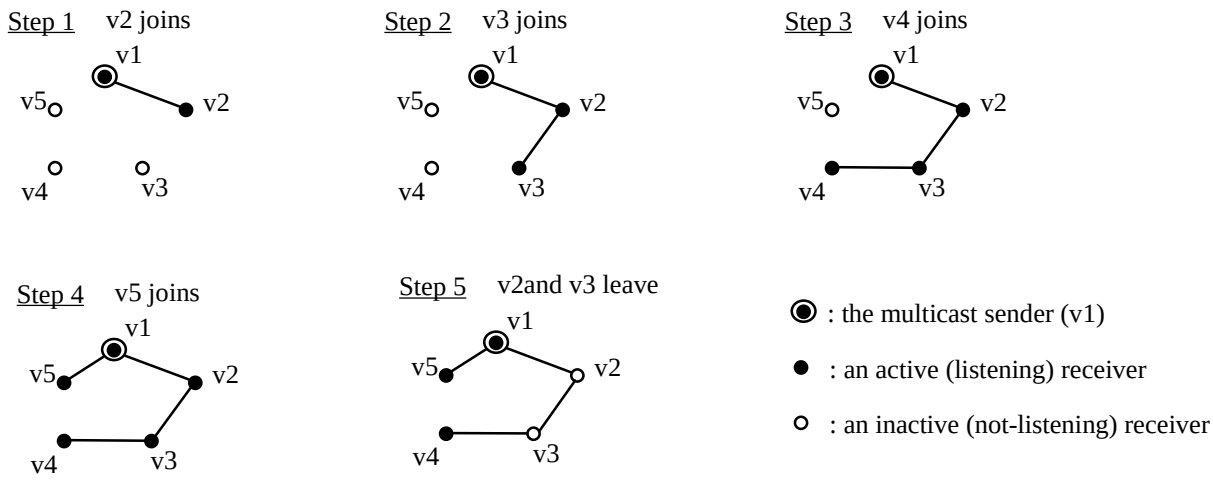

Figure 4 - An example of bandwidth waste when using the Greedy algorithm

```
PLC algorithm(j,d,M,G)
1.  Q ← (all V ∈ G such that V is within M hops from the joining node j)
2.  Q_T ← (all n ∈ Q such that n is a trunk node)
3.  sort Q_T in the ascending order of the distance from j
4.  Q_NT ← (all m ∈ Q such that m is a non-trunk node)
5.  sort Q_NT in the ascending order of the distance from j
6.  while(Q_T ≠ ∅)do
7.    u ← the first element in Q_T
8.    remove u from Q_T
9.    if (u is located at least within the connection region of j) then
10.     Q_Temp ← (all v ∈ Q_NT such that v.TID = u.TID)
11.     while (Q_Temp ≠ ∅)do
12.       w ← the first element in Q_temp
13.       remove w from Q_Temp
14.       if(L_MAX(j) ≥ w.DJN + w.DIS)then
15.         connect j to v as a trunk node and return
16.       else
17.         connect j to v as a non-trunk node and return
18.     end-while
19.   end-if
20. end-while
21. connect j to the sender using the shortest path
22. return
```

Figure 5 - The PLC algorithm

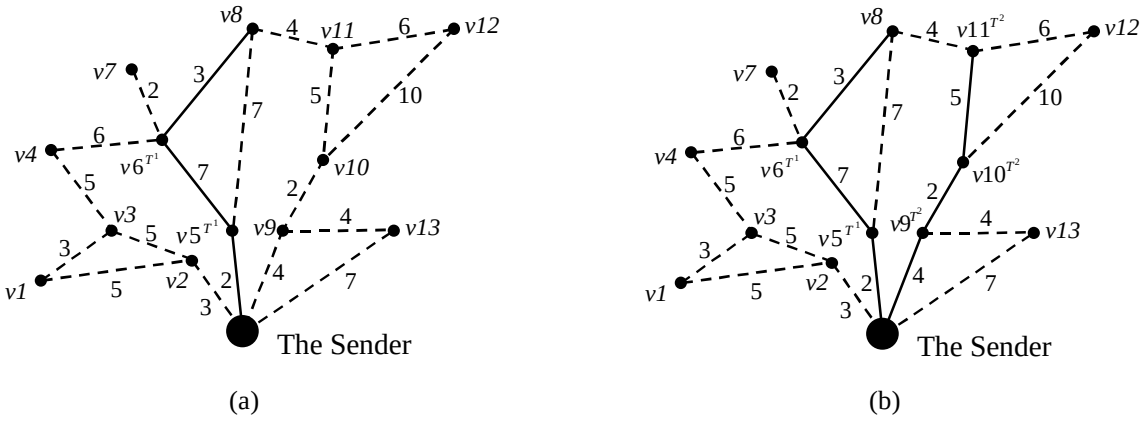

Figure 6 - The networks after (a) *v8* and (b) *v11* are connected

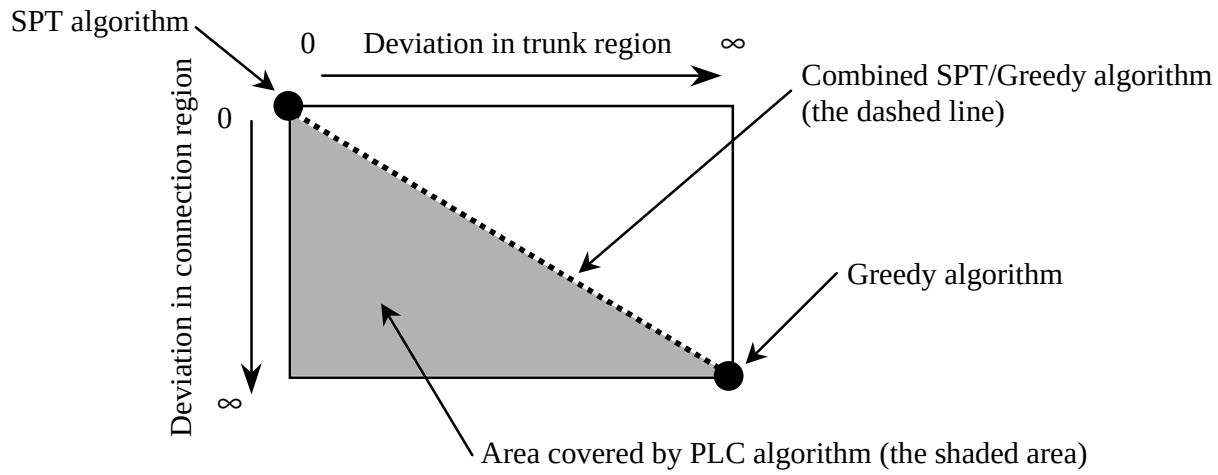

Figure 7 - Routing characteristics of the SPT, Greedy and PLC algorithms

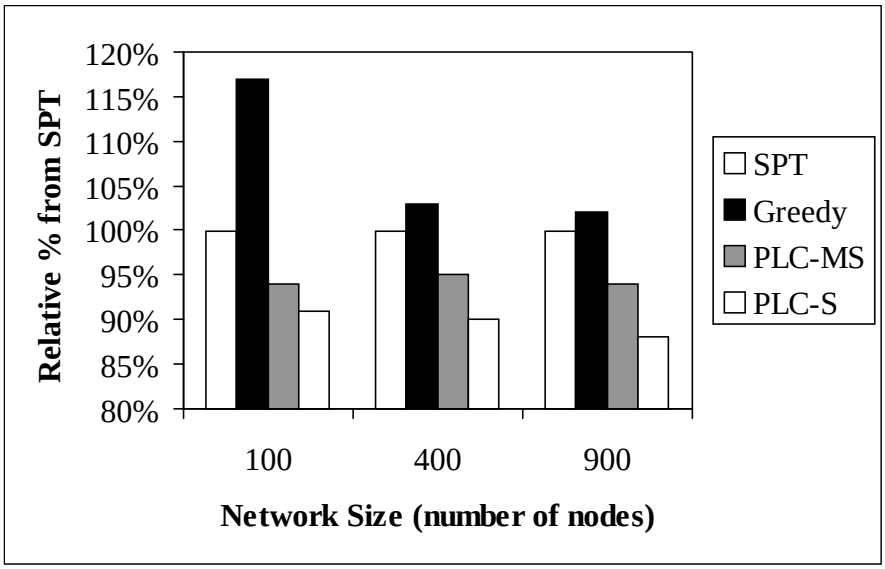

Figure 8 - The relative difference in A_{BWC} from experiment 4

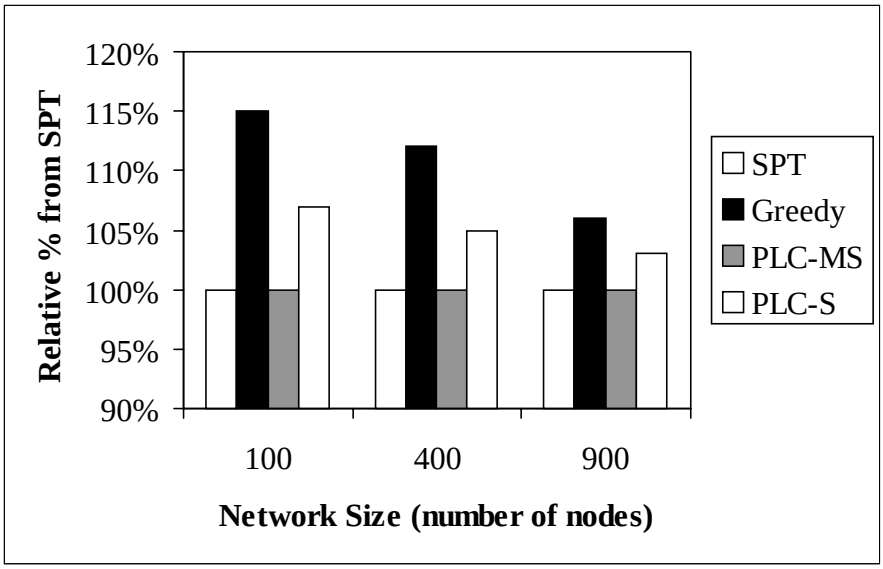

Figure 9 - The relative difference in A_{PL} from experiment 4

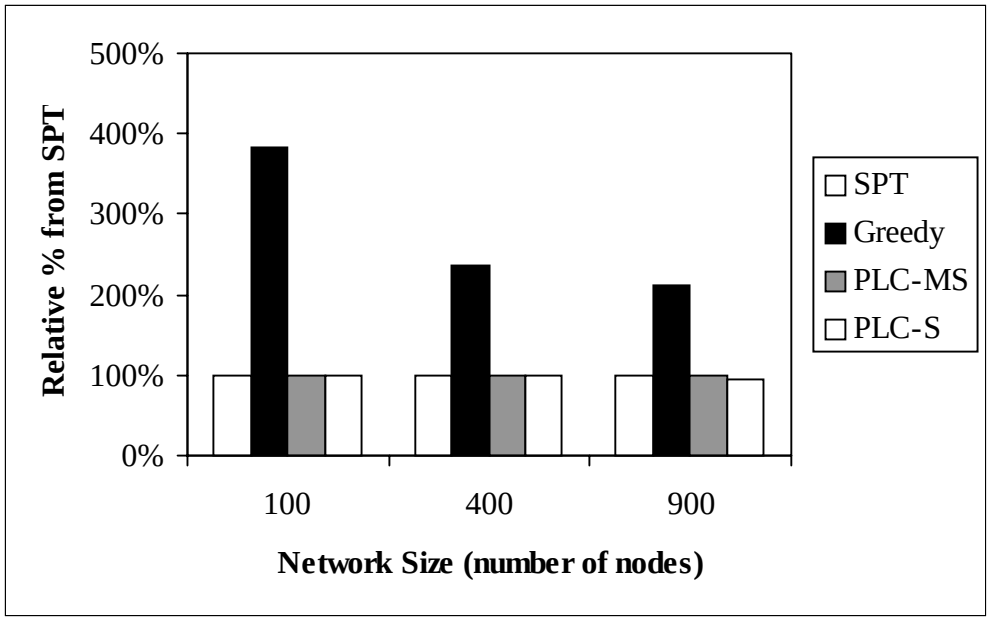

Figure 10 - The relative difference in A_{BWC} from experiment 5

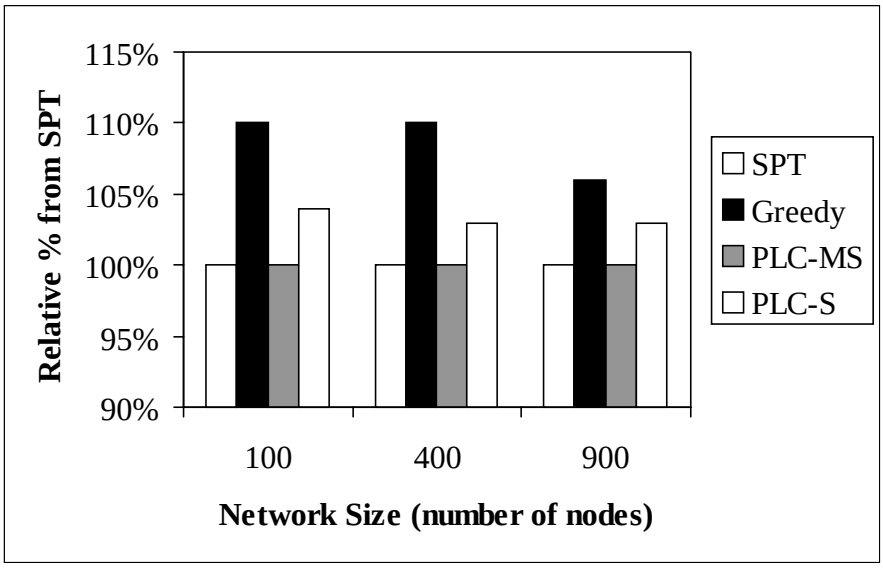

Figure 11 - The relative difference in A_{PL} from experiment 5

Acronym	Description
Joining Node	
JID	The node ID of the joining node
HPC	The hop count (initial value = 0)
TTL	Time To Live (in hops, initial value = M_j)
SID	The node ID of the sender
PID	A unique ID for each polling message
Neighbor Nodes	
NID	The node ID of the neighbor node
DJN	The distance between the joining node and the neighbor node
DIS	The current total distance from the neighbor node to the sender
TNB	The Boolean trunk node indicator (TRUE = a trunk node)
TID	The trunk ID to which the neighbor node belongs

Table 1 - The contents of polling and reply messages with definitions

	SPT	Greedy	Combined SPT/Greedy	PLC
Initial cost	$O(n^2)$	no cost	$O(n^2)$	$O(n^2)$
Non-initial cost	$O(1)$	$O(n^2)$	$O(n^2)$	$O(n^2)$
- External cost	no cost	$O(n^2)$	$O(n^2)$	$O(n^2)$
- Internal cost	$O(1)$	$O(n)$	$O(n)$	$O(n)$

Table 2 - A summary of algorithm complexities for the SPT, Greedy, and PLC algorithms

Data set:	Data set 1	Data set 2	Data set 3	Data set 4	Data set 5
Mean session holding time (exponential) in min	35.0	258.0	35.0	17.5	17.5
Mean inter-session time (Poisson) in min	67.5	7.5	135.1	67.6	405.4
Mean node utilization	43.8%	97.8%	25.3%	25.8%	4.2%
Data set:	**Data set 6**	**Data set 7**	**Data set 8**	**Data set 9**	**Data set 10**
Mean session holding time (Bounded-Pareto) in min	34.7	258.0	34.7	17.2	17.2
Mean inter-session time (Poisson) in min	67.6	7.5	135.1	67.6	405.4
Mean node utilization	45.9%	97.8%	24.2%	23.9%	3.8%
Bounded-Pareto shape parameter (α)	0.8	0.8	0.8	0.8	0.8
Minimum session holding time (*k*) in min	2.9	48.2	2.9	1.2	1.2
Maximum session hold time (p) in min	2880.0	2880.0	2880.0	2880.0	2880.0

Table 3 - Description of the data sets used for the experiments

		Average Bandwidth Consumption			Average Path Length		
	Network Size	**100**	**400**	**900**	**100**	**400**	**900**
Data set 1	**SPT**	100%	100%	100%	100%	100%	100%
	Greedy	97	91	92	117	113	109
	PLC-MS	91	92	92	100	100	100
	PLC-S	86	87	86	107	105	104
Data set 2	**SPT**	100	100	100	100	100	100
	Greedy	79	83	83	124	113	109
	PLC-MS	83	89	88	100	100	100
	PLC-S	81	82	81	110	104	103
Data set 3	**SPT**	100	100	100	100	100	100
	Greedy	117	106	102	114	113	107
	PLC-MS	95	95	94	100	100	100
	PLC-S	92	91	89	107	105	103
Data set 4	**SPT**	100	100	100	100	100	100
	Greedy	117	103	102	115	112	106
	PLC-MS	94	95	94	100	100	100
	PLC-S	91	90	88	107	105	103
Data set 5	**SPT**	100	100	100	100	100	100
	Greedy	382	237	212	110	110	106
	PLC-MS	100	100	99	100	100	100
	PLC-S	99	99	95	104	103	103

Table 4 - The results from experiments 1 through 5

		Average Bandwidth Consumption			Average Path Length		
	Network Size	**100**	**400**	**900**	**100**	**400**	**900**
Data set 6	**SPT**	100%	100%	100%	100%	100%	100%
	Greedy	101	96	98	110	112	139
	PLC-MS	92	93	93	100	100	100
	PLC-S	89	87	87	107	105	104
Data set 7	**SPT**	100	100	100	100	100	100
	Greedy	77	84	81	121	115	109
	PLC-MS	82	89	87	100	100	100
	PLC-S	75	81	79	107	104	103
Data set 8	**SPT**	100	100	100	100	100	100
	Greedy	136	114	108	112	113	108
	PLC-MS	97	96	95	100	100	100
	PLC-S	94	91	89	106	105	103
Data set 9	**SPT**	100	100	100	100	100	100
	Greedy	129	113	111	119	109	106
	PLC-MS	96	95	94	100	100	100
	PLC-S	93	91	88	106	105	104
Data set 10	**SPT**	100	100	100	100	100	100
	Greedy	360	231	237	115	108	104
	PLC-MS	100	100	99	100	100	100
	PLC-S	99	99	97	103	103	102

Table 5 - The results from experiments 6 through 10