Available online at www.sciencedirect.com
SCIENCE DIRECT®
J. Parallel Distrib. Comput. 64 (2004) 1399–1413
Journal of Parallel and Distributed Computing
www.elsevier.com/locate/jpdc

An efficient algorithm for constructing delay bounded minimum cost multicast trees

Shu Li, Rami Melhem*, Taieb Znati

Department of Computer Science, University of Pittsburgh, Pittsburgh, PA 15260, USA

Received 13 February 2003; received in revised form 11 August 2004

Abstract

Multimedia applications are usually resource intensive, have stringent quality of service requirements, and in many cases involve large groups of participants. Multicasting is poised to play an important role in future deployment of these applications. This paper focuses on developing delay-bounded, minimum-cost multicast trees, linking a source to a set of multicast destination nodes. The approach taken in this paper is efficient, flexible and unique in the sense that it cleverly limits its computation only to paths that originate at multicast nodes, thereby avoiding computing paths that exclusively link non-multicast nodes. The simulation results show that the multicast trees produced by the proposed heuristic are of lower cost than those produced by other well-known heuristics, including those which use an expensive *k*-shortest-paths procedure to minimize the cost of the multicast tree. Furthermore, the results show that, in comparison to other heuristics, the proposed scheme results in a significant reduction in the computation time required to build the multicast tree.

Keywords: Multicast trees; Dijkstra's algorithm; Discrete delays; Bounded delays; Constrained minimization; Low-cost multicasting

1. Introduction

The recent advances in data processing and desktop computing, coupled with the advent of high-speed communication networks, paved the way to the development of large-scale multimedia computing and communications systems. As the Internet evolves into a global communication infrastructure, there is an increasing need to offer different levels of quality of service (QoS) and different types of services to support distributed multimedia applications. One specific service, which is poised to play a prominent role in the deployment of multimedia applications, is multicasting.

Mindful of the need to support multicasting, the Internet community has initially focused on adapting current point-to-point routing algorithms to build multicast trees for wide-area networks [3,8,15,17]. The main objective of this effort is to support multicast traffic with reduced network overhead. This approach, however, may result in inefficient resource utilization and potentially longer paths between the source and the multicast nodes. Furthermore, the multicast routing protocols proposed by the Internet community do not consider the QoS requirements of the underlying applications when building the multicast trees. As a result, their support for the performance requirements of multimedia applications is limited. Recently, several schemes were proposed to support end-to-end delay requirements in Internet settings [24]. Other schemes focused on multicast flow control for heterogeneous receivers and on the impact of traffic concentration on multirate network design [9,14].

The focus of this paper is on the class of dual-metric, multicast tree algorithms. More specifically, we consider multicast algorithms that bound the end-to-end delay from the source to each multicast node, while minimizing the overall cost of the tree. This problem, henceforth referred to as the delay-bound, minimum-cost (DBMC) multicast

* Corresponding author.
E-mail address: melhem@cs.pitt.edu (R. Melhem).

0743-7315/$ - see front matter © 2004 Published by Elsevier Inc.
doi:10.1016/j.jpdc.2004.08.007

problem, is NP-complete.[1] Therefore, efficient heuristics that build low-cost multicast trees with bounded end-to-end delay are needed.

One way to reduce the complexity of building QoS-based multicast trees is to use single-metric multicast algorithms and seek to reduce the graph representing the underlying network. Several reduction techniques have been investigated to achieve this goal [7]. Typically, the size of the uni-metric network graph can be reduced by removing the nodes that are guaranteed not to be in the final multicast tree. As an example, a non-multicast leaf node can be removed from the graph without affecting the final result.

Other research efforts aimed at developing multicast routing algorithms based on building a least-delay, source-to-destinations tree [19]. These approaches do not always lead to efficient network resource utilization, since using the least-delay paths does not necessarily optimize sharing of the network links. Minimizing the overall cost of the multicast tree is a key issue in the deployment of multimedia applications in distributed environments.

In this paper, we describe a heuristic referred to as STAR (*segment*, *trim and reconnect*), to construct multicast trees that minimize the cost, while bounding the end-to-end delay from the source to every multicast nodes.

STAR is a centralized algorithm that is best applied to static and sparse multicast groups, using the categorization in [18]. The main objective of STAR is to simultaneously achieve high efficiency, in terms of computation cost, and near-optimality, in terms of tree cost. To achieve this goal, STAR uses a *Dijkstra*-based approach to select efficient paths linking the source node to the destination nodes, but limits its path search only to those paths that originate at multicast nodes. As a result, STAR does not require computing and storing the shortest-paths between *a*ll pairs of nodes [12] in the network, nor does it require computing an optimal path for every relay path during the optimizing procedures [1,2,16]. This feature enhances considerably STAR's overall performance in comparison to other heuristics, without degrading the "quality of the multicast tree". The simulation results show that, for various network topologies, the multicast trees produced by STAR are more efficient than those produced by other known heuristics and the computation time required by STAR is significantly less than the time required by the other simulated heuristics. For example, in a network of 100 nodes with 10 multicast nodes, the simulation results show that STAR is 50–100 times faster than the competitive algorithms and results in multicast trees whose cost is 2–15% lower. In a network of 1000 nodes where the link cost is randomly proportional to the inverse of link delay, the cost reduction exceeds 200%.

The rest of the paper is organized as follows: In Section 2, we discuss related work to multicast support for multimedia applications. In Section 3, we formally define the DBMC multicast problem and discuss the basic motivations underlying the STAR strategy. In Section 4, we provide the basic steps of the STAR algorithm. In Section 5, we analyze and compare the time complexities of the simulated heuristics. In Section 6, we discuss the simulation experiments and present the performance results of STAR in comparison with the other known algorithms. Finally, in Section 7, we provide concluding remarks.

[1] This can be proved by reducing the Steiner minimum tree (SMT) problem, which is a well known NP-complete problem, to the DBMC multicast problem.

2. Related work

Based on the design objectives, the multicast algorithms proposed in the literature can be viewed as members of one of two possible classes. The first class includes algorithms which are designed to accommodate the Internet environment. The second class includes algorithms which aim at reducing the cost of the multicast tree, while bounding the end-to-end delay.

The Internet community proposed different algorithms to create multicast trees, including distance vector multicast routing protocol (DVMRP) [17], multicast open shortest path first (MOSPF) [15], protocol independent multicasting (PIM) [8], and core based trees (CBT) [3]. These protocols are designed to work specifically with the IP environment and take advantage of the IP routing protocols such as routing information protocol (RIP) and open shortest path first (OSPF) protocols. They focus on issues related to reliability, scalability and reduced communication overhead but do not address the QoS requirements of multimedia applications [5,10].

The second class of multicast algorithms aimed at achieving good approximations of the DBMC multicast problem [22]. These heuristics include least-delay heuristic (LDH), constrained routing algorithm (CRA) [20], Kompella, Pasquale, and Polyzos (KPP) heuristic [12] , constrained KMB (CKMB) [21], and bounded shortest multicast algorithm (BSMA) [16]. They differ in terms of their objectives and the techniques used to achieve them.

LDH builds the shortest-delay path tree, using $Dijkstra's$ shortest path *algorithm*, without any consideration of the cost [4]. LDH finds a solution to the multicast problem, if one exists. The total cost of the multicast tree, however, is not optimal. Similar approaches were also proposed in [11], where the cost was set to be equal to the delay. The authors first construct a minimum spanning tree and then use the shortest path to replace any path in the spanning tree that fails to meet the delay requirements. When cycles occur, redundant edges are removed. CRA is similar to LDH except that it tries to reduce the tree cost by building the minimum-cost tree instead of the least-delay tree [20]. In the worst case, this heuristic requires finding the minimum cost tree as well as the minimum delay tree.

The KPP heuristic [12] finds the delay-bounded minimum-cost paths between any two multicast nodes as-

suming integer delay values. The heuristic first computes delay-bounded minimum-cost (DBMC) paths between any two multicast nodes using a dynamic programming approach and then builds the closure graph, a fully connected graph consisting of the multicast nodes. The cost and delay of an edge in the closure graph are the same as the cost and delay of the DBMC paths that connects the multicast nodes. Using the closure graph, the heuristic computes the minimum spanning tree to produce the final delay-constrained tree.

The CKMB heuristic is a derivative of the original STM heuristic [13]. It first finds the DBMC paths between every two multicast nodes using CRA. Using these paths, the heuristic builds a closure graph consisting of the multicast nodes. The cost and delay of an edge in the closure graph is the cost and delay of the DBMC paths between the two end nodes. From the closure graph, the heuristic uses KMB to build the multicast tree [21].

BSMA starts initially with a minimum delay path tree from the source to the destinations and incrementally reduces the tree cost by iteratively replacing relay paths (called super paths in [16]) of higher costs with paths of lower costs [20]. A *relay path* is characterized by the following two properties: (i) all internal nodes on the path are neither multicast nodes nor the source node and are exactly of degree two; (ii) the end nodes of the relay-path are either the source node, a multicast node or a node of degree not equal to two. This iterative process terminates when no relay paths can be replaced.

These low-cost, delay-bounded heuristics described above dealt with the DBMC multicast problem from different perspectives. The simple heuristics (LDH, CRA, CKMB) use the shortest path tree algorithm [6] to build the multicast tree. These heuristics are generally fast, but performs limited optimization. On the other hand, BSMA and KPP are elaborated algorithms that propose the optimization of the relay path and the discretization of the delay bound, respectively. The multicast tree built with BSMA is believed to be near optimal in some delay-cost distributions [19]. The approach of KPP is especially appropriate in situations where paths of low delays may have high costs. BSMA and KPP usually compute satisfactory multicast trees. However, the high-time complexity of KPP and BSMA may make them impractical in large networks.

Recently, an algorithm was proposed using unicast as the underlying protocol [18]. The algorithm computes the unicast routes from the source to multicast nodes only. It then prunes off the redundant paths until the multicast tree is obtained. However, using this scheme, unicast routes are discovered individually, not considering how to share their subroutes optimally with each other. A distributed algorithm is also described in [18] that is based on unicast and the shortest path tree algorithm [6].

Most recently, a class of simple algorithms for low-cost, delay-bounded multicast trees were proposed in [1,2]. The delay-constrained low-cost inexpensive multicasting (SLIM) heuristics builds a low-cost tree incrementally with the minimum-cost paths. The tree is then checked so that the delay bounds for multicast nodes are observed, and minimum delay path is used for individual paths that violate the bounds otherwise. The KSLIM heuristic then improves the cost of the tree constructed with SLIM by applying a simplified version of the strategy used by BSMA to optimize the relay path. The major difference between BSMA and KSLIM is, instead of starting optimization from a minimum-delay tree, KSLIM immediately seeks to build a delay-bounded, low-cost multicast tree. Since the multicast tree is better optimized in the beginning, KSLIM takes fewer optimization steps to converge to a configuration of lower cost than BSMA. Moreover, since they both invoke the k-shortest-paths procedure for each step, KSLIM has a smaller computation overhead than BSMA. SLIM+, a special case of KSLIM, reduces the overhead even further by choosing $k = 1$ and computing beforehand the paths between all pairs of nodes in the network.

In realizing the limitations of centralized algorithms, distributed algorithms are also proposed in [18]. Those algorithms have faster setup time. Moreover, they only require local network status at the proximity of each node. The second merit, however, limits the optimality of the multicast tree. In fact, although an exact and detailed network status is costly to measure and propagate through the nodes, an approximate and structured network status is achievable, and the global network status, even if partial, may increase the quality of multicast. The selection of the distributed or centralized approach is rather a balance between reducing the overhead and increasing the optimality of multicast, which in turn depends on factors such as network size, link status, and multicast duration, etc. The algorithm proposed in this paper, STAR, adopts a centralized approach that assumes the availability of global network status. The knowledge of the network does not have to be complete for each multicast node, but the quality of the multicast tree does depend on the completeness as is expected.

STAR differs from the other centralized heuristics in that it focuses on reducing the computation time without compromising the quality of the tree. A closer look at the process that has been typically used by the heuristics described above reveals that the need to compute the paths between all pairs of nodes leads to a high computation cost. Based on this observation, STAR's strategy is to avoid computing paths between all pairs of nodes in order to reduce computation overhead. To achieve this goal, STAR uses an optimization heuristic that limits the exploration of multicast paths only to those that originate at multicast nodes. Specifically, rather than striving to eliminate relay paths, which at best results in a local optimization of the tree cost, STAR focuses on enhancing the connectivity of the embedded subtrees. Each step of the optimization process consists of removing a subtree and re-attaching its multicast nodes back to the main tree. This process causes the structure of the main tree to change, which in turn creates new anchor points to reattach the removed multicast nodes. The new anchor points

are provided by those nodes that were incorporated in the main tree later than the removed multicast nodes and have potential to enhance the cost of the tree. Consequently, this procedure is fast since it does not require shortest paths to be computed between all pairs of nodes, and produces low-cost trees since the entire structure of the tree is re-arranged using newly discovered anchor points.

3. Problem definition and star strategy

3.1. Formal definition

Consider a network with N nodes and L links. This network can be modeled as a graph $G = (V, E)$ where:

- V, denotes a set of N vertices corresponding to the network nodes, and
- E, denotes a set of L edges corresponding to the network links.

We further define:

- $c : E \to R^+$, denotes a function from E to the set of positive real numbers, R^+, such that $c(e)$ is the cost of link $e \in E$,
- $d : E \to R^+$, denotes a function from E to R^+ such that $d(e)$ is the delay a packet experiences across link $e \in E$,

Given the above notation, the cost of a graph $G(V, E)$ is formally defined as $G.cost = \sum c(e) \forall e \in E$. Similarly, the delay experienced by a packet traversing a path p is defined as $p.delay = \sum d(e)\ \forall e \in E$ on path p.

Assume the multicast group is composed of a source $s \in V$ and a set of multicast destination nodes $M \subseteq V$. Further, assume that every destination multicast node $m \in M$ has a delay bound $\delta(m)$. Given $G(V, E)$, M, δ, c, d and s, the DBMC multicast problem is to construct a tree $T(V', E')$, such that $V' \subseteq V$, $E' \subseteq E$, and (i) $T.cost$ is minimum (ii) $\forall m \in M\ \exists p$ such that p is a path from s to m in T and $p.delay \leqslant \delta(m)$.

Specifically, the DBMC multicast problem can be formulated as:

Find an embedded tree $T(V', E')$ **such that**

$$T.cost = \sum_{e \in E'} c(e) \text{ is minimum}$$

And $\forall m \in M$**,**

$$p_{(s,m)}.delay = \sum_{e \in p_{(s,m)}} d(e) < \delta(m),$$

where $p_{(s,m)}$ is a path in T from s to m and e is an edge on $p_{(s,m)}$.

3.2. Star strategy

The strategy used by STAR to build a multicast tree is based on the observation that the `DBMC paths` from the multicast nodes to all the nodes in the network contain the information necessary to build and optimize the multicast tree. We use `DBMC path` here to denote the path of the shortest cost with a delay smaller than a given bound. Notice that for any given two nodes, there may be multiple `DBMC paths` of different costs corresponding to different given delay bounds.

Consequently, a path between non-multicast nodes need not be considered unless it is contained as a subpath of some `DBMC path` that originates at a multicast node. Since the number of multicast nodes is usually small, relative to the total number of nodes in the network, this translates into considerable savings in computation time. Furthermore, since STAR still uses the relevant path information necessary to minimize the overall cost of the tree, the "quality" of the produced multicast tree is not affected.

Based on the above observation, the `DBMC tree` can be constructed by computing $\wp(m, f, w)$, $\forall m \in M, \forall w \in V$, and $\forall f \mid 0 < f < \delta(m)$, where $\wp(m, f, w)$ is a path in the network from node w to multicast node m whose cost is the lowest among all paths from w to m with delay smaller than f. It is clear, however, that computing exactly $\wp(m, f, w)$, $\forall f \mid 0 < f < \delta(m)$, is computationally prohibitive. To address this issue, STAR approximates $\wp$ by considering only discrete values of f. This approach is similar to the one used in KPP [12]. The difference, however, is that KPP computes the shortest paths between all pairs of nodes in the network, whereas STAR does not compute "unnecessary" paths which are not relevant to build the "optimal" multicast tree.

The `DBMC paths` are computed for the multiple delay bounds and the results are stored in an array which is consulted during the tree building process. Furthermore, similar to other schemes, STAR uses an optimization procedure to further reduce the cost of the tree [1,2,16]. STAR's approach, however, is to only use relevant paths and avoid exploring paths that are not likely to impact the final tree. Based on this approach, the embedded subtrees are trimmed and the disconnected multicast nodes are reconnected to the main trunk. The optimization terminates when all the subtrees have been considered and no cost improvement has been achieved. This approach differs from the approaches used in BSMA [16] or KSLIM [1,2] in two ways: first, the approach used by BSMA and KSLIM to optimize the tree is primarily focused on eliminating relay-paths and does not consider reconnecting the whole subtree as in STAR; second, the approach used by BSMA and KSLIM requires that all paths between all pairs of nodes be explored, whereas STAR focuses on relevant paths only.

Fig. 1 illustrates graphically the differences between STAR and the two other schemes, namely KSLIM and BSMA. In this figure, the path from x to y in *Box A* is a relay path. In an attempt to further reduce the cost of the tree, KSLIM and BSMA delete the relay path resulting in two separate trees as shown in *Box B*. We will refer to the subtree containing the source node as the `source tree` and the other subtree as the `relay tree`. Both schemes

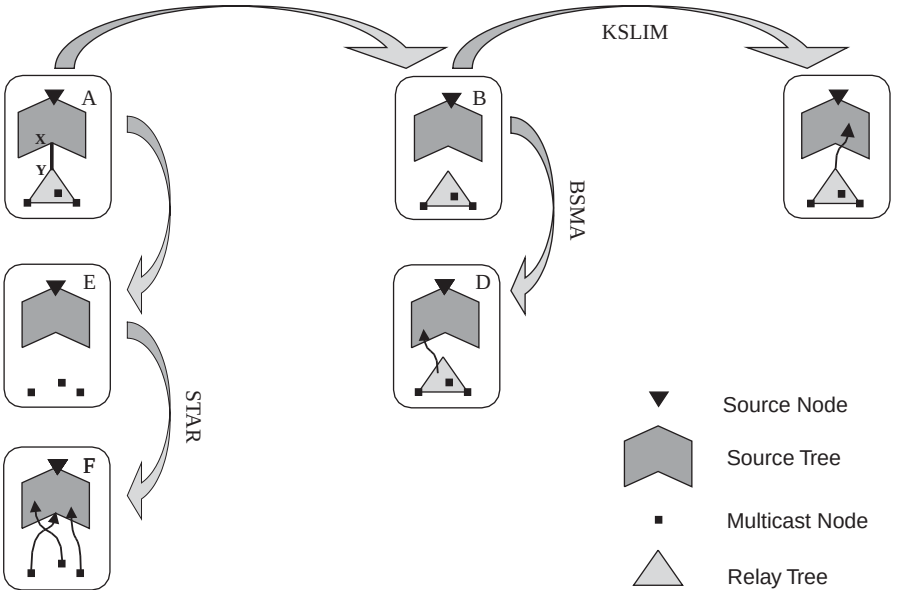

Fig. 1. Comparison of algorithms to optimize a relay path.

then select the best possible paths to reconnect the `source tree`, using any of the nodes in the `relay tree` in the case of BSMA, and using the root node of the `relay tree` in the case of KSLIM. As stated previously, STAR takes a different approach and views a relay path simply as a common subpath of the multicast routes from the source to all the multicast nodes in the `relay tree`. In this example, STAR trims the subtree that contains the relay path (*Box A–E*) and reconnects all the multicast nodes back to the `source tree` (*Box E–F*). Notice that a subtree reconstruction may only result in replacing the relay path with a better substitution, but most likely, as shown in $BoxF$, the entire `relay tree` is destroyed and the embedded multicast nodes are re-distributed to the different branches of the `source tree`. Hence STAR's approach is more comprehensive than just repairing relay paths in an isolated manner. Furthermore, since the multicast nodes are allowed to connect to different nodes in the `source tree`, STAR provides a greater flexibility on how to reconstruct the multicast tree.

With respect to meeting the end-to-end delay requirements of the multicast nodes, STAR uses a different approach than the one typically used by other schemes. Schemes, such as BSMA and KSLIM, first select the minimum cost path and use it to connect the source to the multicast node. If the path violates the delay bound, the next minimum cost path is selected. The process continues until either a path which meets the delay bound requirement is discovered or a total of k paths have been examined. The difficulty of using a k-shortest-path based approach is that the optimum value of k cannot be known a priori. Moreover, paths that have no or little impact on the final tree may still end up being considered during the tree construction process.

To meet the delay bound requirements, STAR computes the `DBMC paths` and stores them in an array Π that is indexed by the delay values associated with the paths. Let $p_{(s,w)}$ be a path from the multicast source node to an intermediate node w in the tree. To connect a new multicast node m to the tree at node w, such that the delay bound $\delta(m)$ is not exceeded, a path from m to w is retrieved from Π using the index value of $\delta(m) - p_{(s,w)}.delay$. The retrieved path is an approximation of the `DBMC path` and is guaranteed to meet the delay requirement. The desired level of approximation accuracy can be achieved by a careful selection of the granularity of partitioning the delay bound.

Fig. 2 illustrates the basic steps STAR uses to build the multicast tree for the network depicted in (a). In this network, S represents the source of the multicast group and M1, M2 and M3 represent the destination multicast nodes. Each link, represented as a thin line, is associated with a cost. It is further assumed that the link delays are all 1, and the delay bound is 3 for all multicast nodes. Paths linking the source to the multicast nodes are represented as thick lines. The steps used to build the multicast tree are depicted in subfigures (b)–(d), respectively. Nodes M1, M2 and M3 are included in the `source tree` using the shortest path to the tree recursively. In an attempt to optimize the relay path from S to $N1$, depicted in subfigure (d), KSLIM and BSMA fail to find a better substitution. In fact, the multicast tree depicted in subfigure (d) is the final tree since there exist no better substitution for the relay path. In contrast, STAR proceeds with further optimization of the tree, as shown in subfigures (d)–(g). The cost of the final multicast tree produced by STAR, which is depicted in subfigure (g), is 40% lower than the cost of the final multicast tree produced by KSLIM and BSMA.

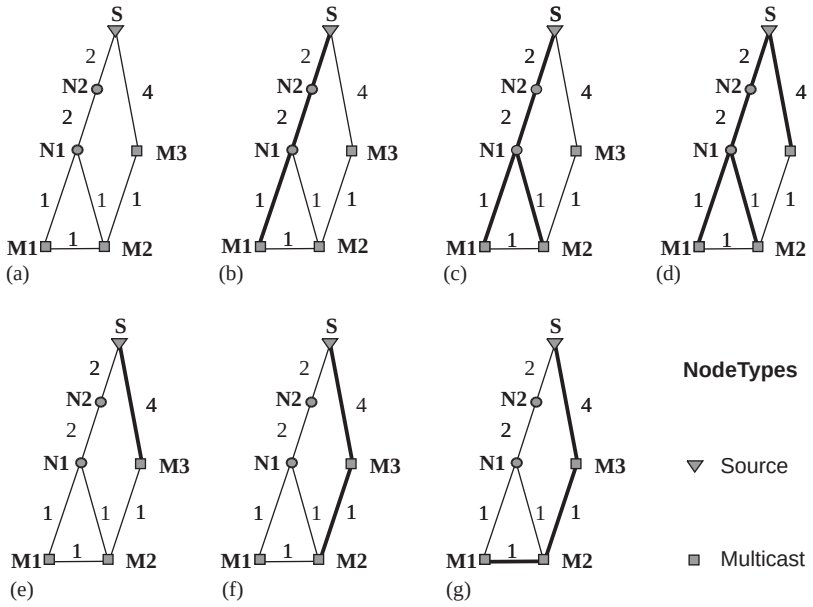

Fig. 2. Demonstration of the flexibility of STAR.

Fig. 2 also illustrates the reason why STAR requires less computation time than BSMA and KSLIM. Consider the relay path from S to $N1$. Since it cannot be known a priori that $N1$ terminates a relay path, either a substituting path has to be computed for every relay path or the shortest paths for all pairs of nodes have to be computed beforehand. In either case, the process requires a substantial computation overhead. In contrast, since STAR only needs to compute the shortest path starting at multicast nodes, it does not compute the shortest path from $N1$ to all the other nodes.

4. Algorithm description

In the following, we give a description of the three phases of computing the multicast tree. In the first phase, the DBMC paths are computed. In the second phase, the delay feasible multicast tree is built. In the last phase, the cost of the tree is further reduced.

4.1. Computing the DBMC paths

In order to simplify the discussion, the cost and delay are assumed to be symmetric, i.e., the values are equal for both directions of a link. When links are asymmetric, using the original graph as the template, the algorithm will make an image graph first where an in-link is created for each out-link and vice versa. The algorithm will then find the path using the image graph. In the simulation presented in Section 6, however, we assume that the cost and delay are asymmetric.

The algorithm *SegmentedDelayPaths* presented in this section computes the array Π. Each entry $\Pi(u, i, w)$ is an approximation for the corresponding path $\wp(u, f, w)$ where f is mapped to i by a user defined function *DtoI*. *DtoI* can be any monotonically increasing function. Furthermore, just as the delay bounds may be different for different multicast nodes, *DtoI* may take different forms for different multicast nodes. To simplify the issue, in the simulation and Example 1, we assume that $\delta(m)$ has a constant value δ, $\forall m \in M$, and *DtoI* takes the simple form of $\lfloor (f/\delta)\lambda \rfloor$, where f is the delay value and λ is the number of intended partitions of δ.

As shown in Algorithm 1, procedure *SegmentedDelayPath* computes all the DBMC paths that originate at a multicast node. Before procedure *SegmentedDelayPaths* is called, however, the costs and delays for all the paths in Π should be initialized to infinity, except the paths $\Pi(m, 0, m) \forall m \in M$, for which the costs and delays are initialized to 0.

The same principle used in $Dijkstra's\ Algorithm$ is followed in *SegmentedDelayPath*. As shown in lines 3–4, for each iteration step, the shortest path of the "candidate set", Θ, is selected and named as θ. In lines 6–8, an attempt is made to extend θ by one more edge, and the extended path is denoted as ξ. Lines 9–19 are selection rules. The path just discovered, ξ, will be stored in Π and added to Θ (lines 20–23) if and only if the boolean variable β is true. Should β always be true, *SegmentedDelayPath* would simply probe the connectivity of the network and store all the connected paths. The selection rule in lines 10–12 is added to exclude paths that violate the delay bound $\delta(m)$. The selection rule in lines 14–16 is added to exclude a path if another path has been discovered that has smaller delay and cost. Until now, no DBMC path has been excluded for all the delay constraints smaller than $\delta(m)$. In other word, all the excluded paths are not DBMC paths. The proof follows.

Algorithm 1. SegmentedDelayPath

1: **for all** $m \in M$ **do**
2: $\Theta \leftarrow \{\Pi(m, 0, m)\}$
3: **while** Θ is not empty **do**
4: $\theta \leftarrow$ the path in Θ with the smallest delay
5: remove θ from Θ
6: $v \leftarrow$ the ending node of θ
7: **for all** edge $(v, w) \in E$ **do**
8: $\xi \leftarrow$ the path resulting from extending θ by (v, w)
9: $\beta \leftarrow true$
10: **if** $\xi.delay > \delta(m)$ **then**
11: $\beta \leftarrow false$
12: **end if**
13: $\Xi \leftarrow \{\Pi(m, i, w) \mid i$ is a valid index of $\Pi\}$
14: **if** $\exists \xi' \in \Xi \mid \xi'.cost < \xi.cost \wedge \xi'.delay \leqslant \xi.delay$ **then**
15: $\beta \leftarrow false$
16: **end if**
17: **if** $\exists \xi' \in \Xi \mid \xi'.cost < \xi.cost \wedge DtoI(\xi'.delay) = DtoI(\xi.delay)$ **then**
18: $\beta \leftarrow false$
19: **end if**
20: **if** $\beta = true$ **then**
21: $\Pi(m, DtoI(\xi.delay), w) \leftarrow \xi$
22: add ξ to Θ
23: **end if**
24: **end for**
25: **end while**
26: **for all** valid index i **do**
27: **if** $\exists \Pi(m, j, w) \mid j < i \wedge \Pi(m, j, w).cost < \Pi(m, i, w).cost$ **then**
28: $\Pi(m, i, w) \leftarrow \Pi(m, j, w)$
29: **end if**
30: **end for**
31: **end for**
32: return Π

Theorem 1. *In Algorithm* 1, *let U be a storage of unlimited space that can be indexed by a tuple* (*starting node, delay bound, ending node*). *If Π is replaced by U, without the last selection rule* (*lines* 17–19) *the* DBMC paths *originating from m ∈ M can be found in U for any delay bound smaller than* $\delta(m)$.

Proof. Proof by contradiction. Given a multicast node *m*, a destination *w*, and a delay bound $f < \delta(m)$, let *v* be the path of the minimum cost that starts at *m*, ends at *w* and whose delay is smaller than *f*. Suppose *v* will not be found in *U*. There are only three possibilities: (1) *v* is never assigned to ξ in the process of execution; (2) *v* is assigned to ξ but not stored in Π; (3) *v* is stored in Π but overwritten by some other path. The last possibility is clearly a contradiction. For a new path to overwrite *v*, it should have the same delay as *v* since *U* is indexed by (starting node, delay, ending node). Moreover, it should have a lower cost than *v* according to line 14. For the same reason, the second possibility is a contradiction. Since *v* has a delay smaller than $\delta(m)$ by definition, if *v* is not stored, then the boolean condition on line 14 must be false. But then that means there exits a path of the same delay but lower cost, which is a contradiction. For the first possibility, without losing generality, we assume that *v* ends with edge (v, w) and θ is the subpath of *v* that excludes only edge (v, w). Thus θ is a DBMC path that starts at *m*, ends at *v*, and for a delay bound smaller than *f* minus the delay of edge (v, w). Otherwise if there is a path σ better than θ, extending σ by edge (v, w) will result in a path better than *v*. Thus θ cannot be found in *U* by the assumption. By induction, any subpath of *v* starting at *m* is not in *U*, including the path that starts and ends at *m*, with cost and delay being zero. This is contradictory to line 2 of the algorithm. □

The last selection rule (lines 17–19) is needed, because otherwise the number of DBMC paths in *U* can be enormous. According to the rule, only the path with the minimum cost will be stored for all the DBMC paths whose delays are within the same partition. The partition is defined naturally by the above-mentioned function *DtoI*. The following gives an example of selecting paths based on delay and cost.

Example 1. Fig. 4 demonstrates the partial execution steps for the network illustrated in Fig. 3. In both figures, the delay and cost of a link or a path are noted as the tuple (delay, cost). Fig. 4 shows the content of the array of DBMC paths for multicast node m_0. Multicast node m_1 will have a similar array. Suppose the delay bound is 4. *DtoI* maps the path to one element of the array when the delay is smaller than 2 and to another element otherwise. The first two rows show paths directly discovered by the links to node *a*, *c* and *v*. The third row shows the new paths discovered through links *ab* and *cv*. Since the new paths have the delay of 2, they will have to be stored in the first array element according to *DtoI*. The path to *v* is problematic as there is already a path stored. Thus the algorithm break the tie by choosing the path with smaller cost, irrespective of the delay. The result is row 4. Row 5 shows the discovery of another path to *v* through link *bv*. Since this path has delay of 3, it is stored in a different array element with no conflict. The arrows in the figure trace the paths back to the multicast node m_0.

Finally, in order to approximate $\wp(u, f, w)$, we need to define $\Pi(u, i, w)$ as the approximation of the minimum cost path with delay bound *f* such that $DtoI(f) \leqslant i$, instead of $DtoI(f) = i$ as computed in the previous steps. Thus path $\Pi(u, j, w)$ is replaced with a path with smaller cost and delay if such a path can be found in Π (lines 26–30).

Notice that, because of the discretization of the delay bound, the paths found by Algorithm 1 for a given delay bound may not be the minimum cost path for that delay bound. Moreover, for the same reason, a path may not be

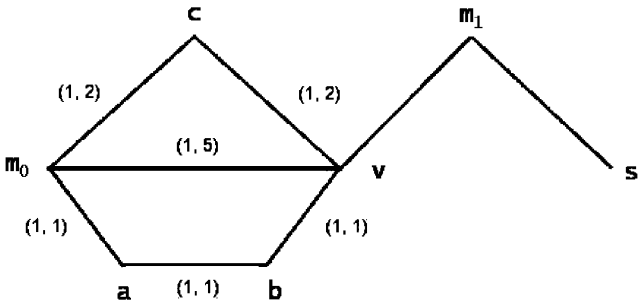

Fig. 3. A simple network to illustrate Algorithm 1.

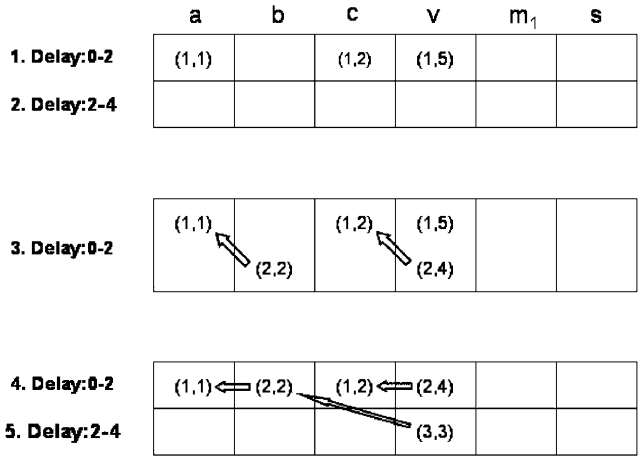

Fig. 4. Partial execution steps of Algorithm 1.

found for a given delay bound even if such a path exists. In order to overcome this, in the simulation presented in Section 6, the minimum-delay path is computed if needed to guarantee that a multicast tree will always be built if one exists.

Theorem 2. *The time complexity of Algorithm 1 is* $O(\lambda|M||V|^2)$.[2]

Proof. In Algorithm 1, *DtoI* partitions the delay bound into λ partitions. A path of delay f will be stored in partition $DtoI(f)$. In line 4, Algorithm 1 selects the path of the minimum delay. In the selection however, only the current delay partition needs to be considered, since any unmarked path in the other partitions will have a larger delay than any of the path in this partition. There are at most $|V|$ paths in one partition and one for each node in the network as a destination. Thus each selection takes $O(|V|)$ steps. Since all the paths in the partition have to be marked, there are $O(|V|)$ selections to take. Finally, since there are λ partitions for each of the multicast nodes, the whole algorithm has time complexity of $O(\lambda|M||V|^2)$. □

Algorithm 1 can be further improved from the following observations. For any specified source, u, and destination, w, a path $\wp(u, x, w)$ may have a lower cost than the path $\wp(u, y, w)$ if $x > y$, i.e, if $\wp(u, x, w)$ has a larger delay bound than $\wp(u, y, w)$. However, this is not true if the specified delay bounds are smaller than d_{mindelay}, the delay of the minimum delay path, or larger than d_{mincost}, the delay of the minimum cost path. This is simply because, on the one hand, the path of the minimum cost is the path of the minimum cost for all the delay bounds; on the other hand, no paths will be found for delay bounds smaller than d_{mindelay}. Hence partitioning the delay bound in the range of $(d_{\text{mindelay}}, \min(\delta, d_{\text{mincost}}))$ instead of $(0, \delta)$ results in a finer granularity with no extra computations, where d_{mincost} and d_{mindelay} can be computed separately by *Dijkstra's algorithm.*

4.2. Constructing the multicast tree

The next step is to construct the multicast tree using array Π. The multicast tree T is built with the shortest path tree algorithm [6] as was done in [18].

Definition 1. For any multicast node m not yet in T, let $P(m, T, w)$ be a $\wp(m, f, w)$, where $f = \delta(m) - \xi_w.delay$, and ξ_w is the path from the multicast source to a node w in T. In other words, it is the minimum cost path that links the multicast node m to the node w in T without violating the delay bound, $\delta(m)$.

Example 2. In Fig. 5, multicast tree T is shown as thick lines. The link delay for some links are annotated. Multicast node m, whose delay bound is 9, is to be connected to w. Path $[s, u, v, w]$ is ξ_w in the above definition. Since $\xi_w.delay$ is 5, a total budget of 4 is left for m to connect to w. If path $[w, x, m]$ is $\wp(m, 4, w)$, then $P(m, T, w)$ is path $[w, x, m]$.

Fig. 5 also shows that, in addition to the linkages to its parent and child nodes, every node w in T built so far will store the delay $\xi_w.delay$ and the cost $\xi_w.cost$, where ξ_w is the path in a multicast tree from the multicast source to a node w. Using the precomputed array Π, the following steps demonstrate how to compute $P(m, T, w)$ defined above.

(1) Compute the maximum allowed delay for new paths anchored at w as $f = \delta(m) - \xi_w.delay$, where $\xi_w.delay$ can be read from node w.
(2) Retrieve from Π the path of the minimum cost from w to m with delay bound f.

Path P will be used as the building block for the construction and optimization of the multicast tree. With T initialized to s, the multicast tree can be constructed with the

[2] A faster algorithm can be developed to compute Π. We choose an implementation consistent with those in [11–14] for fairness of comparison.

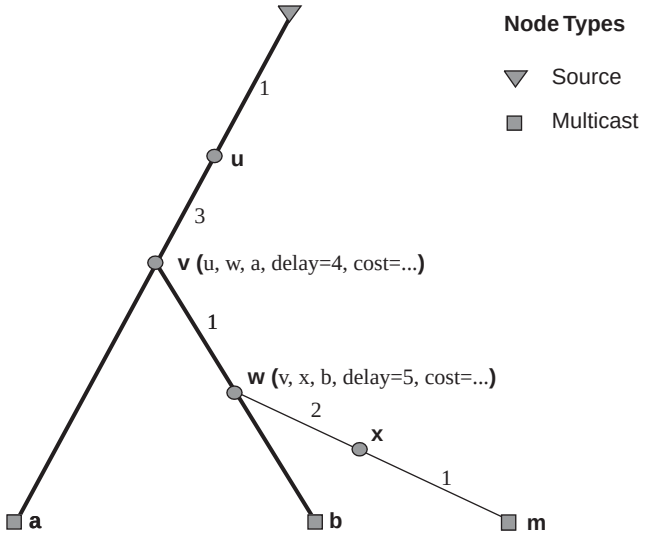

Fig. 5. An example of path $P(m, T, w)$.

following procedure.

while $\exists m \in M \mid m \notin T$ **do**
  $\xi \leftarrow P(m, T, w)$ of the minimum cost, $\forall m \in M \mid m \notin T, \forall w \in T$
  add ξ to T
end while

In the above procedure, a path P with the minimum cost is included in the multicast tree iteratively, until all the multicast nodes are connected. Notice that P is selected for all the nodes in T and for all the multicast nodes not yet connected.

Theorem 3. *In a multicast tree constructed as described above, no relay path can be replaced by another path of lower cost.*

Proof. Proof by contradiction. Consider a relay path v that starts with node x, ends with node y, and with a delay of f. Since a relay path is a simple path and has no multicast nodes anywhere except at the end nodes, v had to be introduced to the tree when a multicast node was being connected to the tree. Let this multicast node be m and the path that connects m to the tree be ξ. Then v is a subpath of ξ. Now suppose v' is a replacement path for v, then there will be a path ξ' that can be derived from ξ with v replaced by v'. Clearly ξ' is a path of lower cost than ξ. Moreover, when m is linked to the tree, $\delta(m)$ will be observed. However, by Definition 1 , ξ is the `DBMC path` that does not violate $\delta(m)$, which is a contradiction. □

Adding a path to the tree may generate Loops. STAR removes loops on the fly by deleting the redundant path of larger delay and modifying the delay/cost values of the affected subtrees.

4.3. Optimizing the multicast tree

The tree is optimized by calling *TrimAndReconnect*. The algorithm cuts the subtree rooted at `trim nodes` and then reconnects the multicast nodes back to the `source tree` with delay bounded paths of minimum cost.

Definition 2. A `trim node` is a node in the multicast tree that is a child of node w, where w is the multicast source or a node of degree greater than 2.

Algorithm 2. `TrimAndReconnect`

```
1:  T'.cost ← ∞
2:  T ← the multicast tree
3:  Label-KeepTheNewTree:
4:  T' ← T /*backup the current tree*/
5:  Ψ ← an empty stack
6:  do a breadth-first search of T and
    push trim nodes to Ψ
7:  while Ψ is not empty do
8:    v ← pop from Ψ
9:    u ← the parent node of v in T
10:   R ← the subtree rooted at v of T
11:   prune R from T
12:   while ∃m ∈ M | m ∉ T do
13:     ξ ← P(m, T, w) of the minimum cost,
        ∀m ∈ M | m ∉ T, ∀w ∈ T | w ≠ u
14:     add ξ to T
15:   end while
16:   if T.cost < T'.cost(1 − ε) then
17:     goto Label-KeepTheNewTree
18:   else
19:     T ← T' /*restore the backup*/
20:   end if
21: end while
22: return T'
```

In Algorithm 2, the optimization starts by discovering the `trim nodes` in a tree. For example, in the tree shown in Fig. 6, the `trim nodes` are a, b, v, c, m_3, d in the order of discovery. The `trim nodes` are stored in a stack, Ψ (lines 5–6). A stack is used because we want to process them in the inverse order of the breadth-first search. Starting optimization from a small subtree favors progressively modifying the structure of a multicast tree. The alternative is to optimize a large tree before its subtrees. The extreme in this case is to build up different and completely new trees from the very beginning and retain the one that has the lowest cost. From our preliminary studies, this strategy is not a good strategy for most cases.

The `relay tree` rooted at a `trim node` is then pruned (line 11). The multicast nodes not yet connected are reconnected back to the `source tree` in the following while loop. The loop is similar to the one that we used to build the multicast tree in Section 4.2, except for the added condition

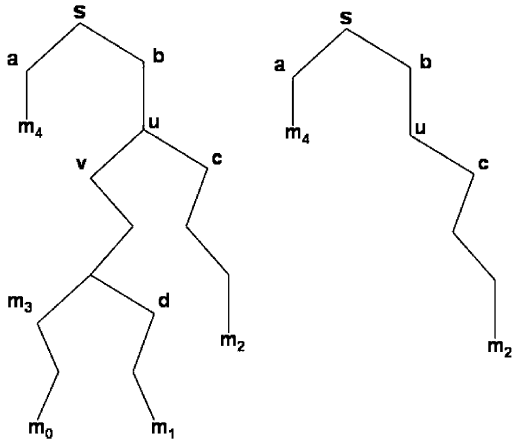

Fig. 6. Left: an intermediate tree; Right: after trimming at node *v*.

($w \neq u$). The added condition shields the original anchor u and forces the tree to reorganize as much as possible. The reconstructed tree will have a topology different from the original one.

The remaining steps (lines 16–19) evaluate the cost of the new tree. If the new tree improves the cost by a fraction of at least ε, the new tree is kept and the optimization continues (line 17). However, because the topology of the tree has changed, we rescan the tree for the new `trim nodes`. If, however, the new tree does not improve the cost by at least ε, the new tree is discarded and the process is repeated at the next `trim node` in Ψ (line 19). The process stops and outputs T' as the optimized tree when all the `trim nodes` in Ψ have been considered and none of them results in a lower-cost tree.

According to the above descriptions, we make the following observations:

- Since $P(m, T, w)$ excludes from its budget the delay that is consumed by the anchor node, $\delta(m)$ will be observed for every path in the multicast tree from the multicast source to the multicast node m.
- Since STAR handles the redundant path, the construction and optimization is loop free.
- Since each step of optimization ensures the improvement of cost by a fraction of ε, the algorithm halts after a finite number of steps.
- Since STAR selects the best path among the paths from $|M|$ multicast nodes to $|V|$ anchor points until all the multicast nodes are connected, STAR takes $O(|M|^2|V|)$ for each step of optimization(line 12–15).

In summary, STAR terminates after a finite number of steps and produces a delay bounded multicast tree, if one exists.

5. Time complexity analysis

Let the number of vertices be $|V|$, and the number of multicast nodes be $|M|$. The time required by KPP to compute a path is on the order of $O(\lambda|V|^3)$, where λ is the number of discrete values of the delay bound. KSLIM and BSMA require $O(k|V|^2)$ for each step of optimization, where k is the number of shortest paths to be considered during the search. As a special version of KSLIM ($k = 1$) [1,2], SLIM+ requires $O(|V|^3)$ to compute and store the shortest paths for all pairs of nodes and takes $O(|V|)$ for each step of optimization. STAR requires a time complexity of $O(\lambda|M||V|^2)$ to compute the `DBMC paths` and takes $O(|M|^2|V|)$ for each step of optimization, where λ has the same meaning as in KPP. Table 1 summarizes the complexity of these algorithms.

Table 1
Time complexity and bounds of the DBMC heuristics

Heuristics	Path computation	Tree construction	N-step iterative cost reduction
KPP	$O(\lambda\|V\|^3)$	$O(\|M\|\|V\|)$	—
BSMA	—	$O(\|V\|^2)$	$O(k\|V\|^2N)$
KSLIM	—	$O(k\|M\|\|V\|^2)$	$O(k\|V\|^2N)$
SLIM+	$O(\|V\|^3)$	$O(\|M\|\|V\|^2)$	$O(\|V\|N)$
STAR	$O(\lambda\|M\|\|V\|^2)$	$O(\|M\|^2\|V\|)$	$O(\|M\|^2\|V\|N)$

The overall time complexity of the algorithms discussed above, with the exception of KPP, depends on the number of steps to be carried out during the optimization process, which is denoted as N in Table 1. The derivation of an upper bound on the number of these steps is difficult. It has been shown that under specific assumptions this bound is on the order of $O(|V|\log|V|)$ [16]. To derive this bound, however, it is assumed that the number of nodes in the multicast tree is on the order of $O(|V|)$. It can be argued that the number of nodes in a multicast tree is usually much smaller than the number of network nodes. Furthermore, it is assumed that the transition from a multicast tree of higher cost to any of the multicast trees of lower costs is equally likely. In this case also, the transition may be faster if the cost of the tree reduction is larger for every optimization step. Consequently, in certain cases, $|V|\log|V|$ is likely to represent a loose bound of the number of optimization steps. In fact, the expected number of optimization steps has been shown to be on the order of $O(|V|)$, for a degree-bounded network [16].

To obtain a close estimate of the actual bound, Fig. 7 plots the number of optimization steps in STAR, where each optimization step includes a trimming and a reconnection procedure as illustrated in Fig. 1. In this case, the value of λ is set to 1, the number of nodes in the network is 100, the number of multicast nodes is shown on the figure.[3] As shown in the figure, the number of optimization steps is far less than $O(|V|\log|V|)$ (~ 1000) when the fraction of multicast nodes is less than 20% of the total number of network nodes. Fig. 7 also plots the ratio of computation time required to compute the `DBMC paths` relative to the time required to optimize the multicast tree. Even though the `DBMC paths`

[3] The network parameters are set to their default values as described in Section 6.

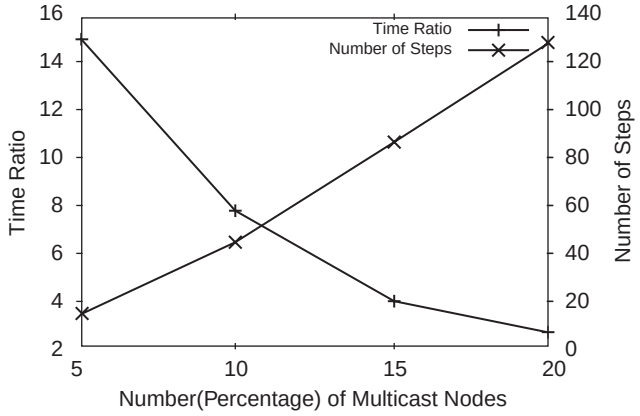

Fig. 7. Time ratio and number of optimization steps.

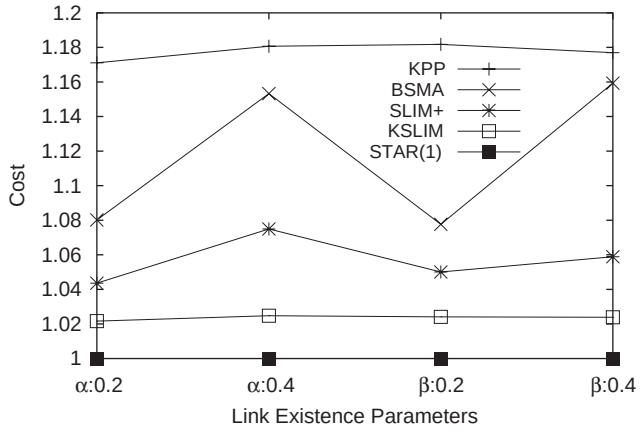

Fig. 8. Effect of α and β of link existence probability on cost.

are computed with $\lambda = 1$, the path computation takes far more time than the optimization procedures. This confirms the point made previously that the path computation rather than the optimization cost is the dominant factor of the computation time for these algorithms. This further explains why STAR is faster than SLIM+ even though SLIM+ takes less computation for every optimization steps.

6. Simulation

This section shows the simulation results. The simulation is done by choosing a default set of parameters that we thought are most probable in practice, and change one or two parameters at a time for comparisons. Unless stated otherwise, the default parameters should be assumed.

We use the random graph generator[23] as used in some other papers[1,2,12,16]. Random graphs are generated on a $2000 * 5000$ area. A random link has a probability of existence of $\beta e^{-g/h\alpha}$, where g is the physical distance of the link, and h is the diagonal distance of the simulated area. We choose the following set of default values: α and β being 0.3, the number of nodes being 100, the number of multicast nodes being 10, the delay bounds being 7000, the cost-delay distribution being random cost and proportional delay, abbreviated as RcPd. By RcPd, the link cost function is a random number from 1 to 1000, while the link delay function is simply the geometric distance on the random graph.

A random graph is then generated with these parameters and the minimum delay paths are computed. If the delay bound is smaller than any of the delays to the multicast nodes, the random graph is discarded and a new graph is generated until the multicast tree can be built. On the other hand, a multicast tree will always be built if it exists. This is guaranteed by using the minimum delay path if no alternatives are found. With the same graph, multicast trees are constructed using the various algorithms. For a specific algorithm, both the computation time and the cost of the multicast tree are normalized with respect to those of STAR. In the following discussion and figures, they are abbreviated as time and cost respectively. The time and costs are taken from experiments and averaged over 500 runs.

Graphs of 500 and 1000 nodes are also used in the simulation. The default conditions are the same as before except that α and β are 0.1 and that the average is taken for 200 experiments. With the default parameters, the average node degrees for the graphs of size 100, 500, and 1000 turn out to be 11, 9, and 17, respectively.

The optimization step of STAR will not store the new configuration of a multicast tree if its cost is over $1-\varepsilon$ times that of the old configuration. STAR uses 0.05% as its ε for all the experiments. For graphs of 100 nodes in our experiments, the enforcement of ε has no effect on the experiment results compared with the case when no ε is enforced. For graphs of 500 and 1000 nodes, enforcing ε does make the program run a little faster in some cases.

The number of segments used by STAR to compute the paths is denoted by λ and the corresponding algorithm is denoted by STAR(λ). We will first present the results of STAR(1), i.e., STAR with no segmentation in computing paths. The results of $\lambda > 1$ is shown later.

Fig. 8–11 compare the cost of the multicast tree and show that our solution is about 15–20% better than KPP, about 8–15% better than BSMA, about 5–10% better than SLIM+, and 2% better than KSLIM. Fig. 11 shows that STAR performs well in large networks. Fig. 11 only compares the result with SLIM+, since the other algorithms require prohibitive computation times when the network size is large.

Fig. 12 is a typical running time comparison. Our algorithm is 50–80 times faster than KSLIM, KPP and BSMA, and 10 times faster than SLIM+ in default conditions.

Fig. 13 shows that the running time for the algorithms increases with the decrease in the number of multicast nodes. Since the time is normalized to that of STAR, this increase is because STAR's running time decreases with the decrease in the number of multicast nodes. This coincides with the observation that *SegmentedDelayPath* is more expensive than

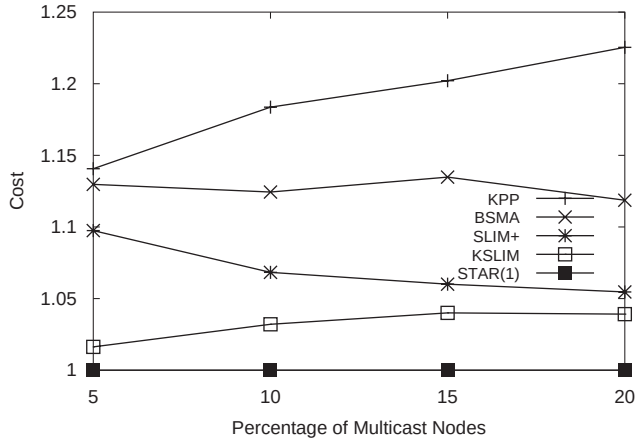

Fig. 9. Effect of number of multicast nodes on cost.

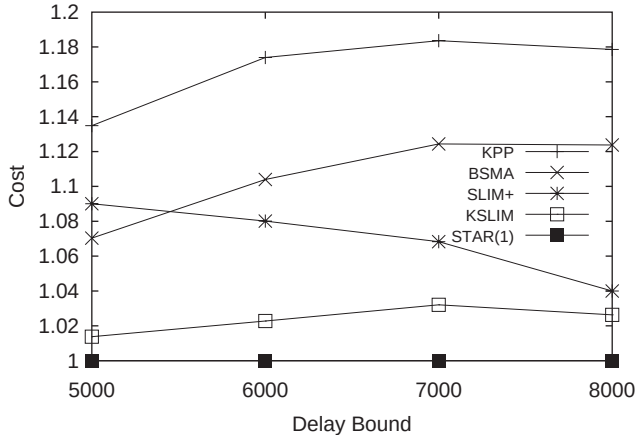

Fig. 10. Effect of delay bound on cost.

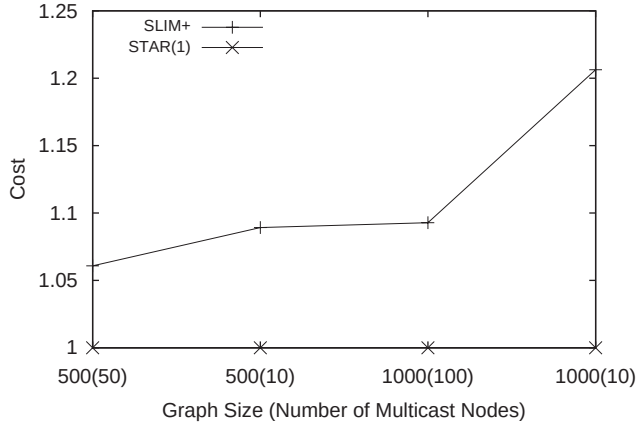

Fig. 11. Effect of network size on cost.

TrimAndReconnect for STAR as shown in Fig. 7, and decreasing the load of computations by *SegmentedDelayPath* increases STAR's speed. Note that the running time of KPP seems to behave differently from the rest. That is because KPP's complexity is determined by finding the all-pair shortest paths, which is independ of the number of multicast nodes. Given the fact that the running times for all the other

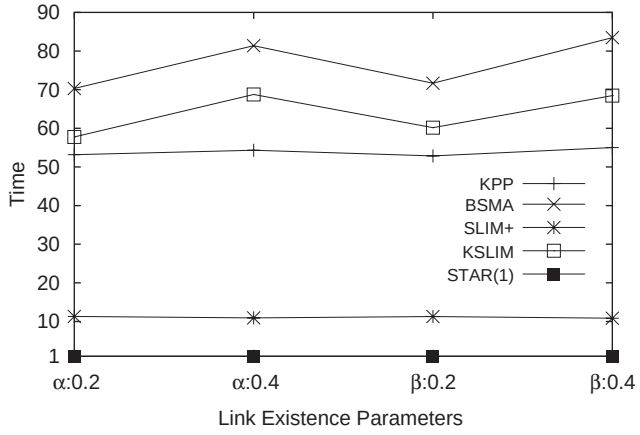

Fig. 12. Effect of α and β of link existent probability on time.

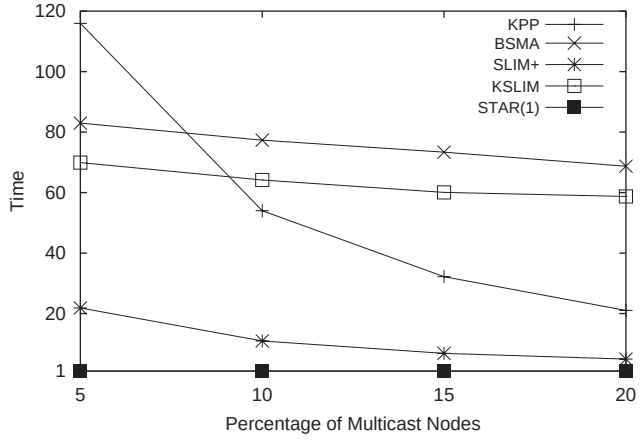

Fig. 13. Effect of number of multicast nodes on time.

algorithms is proportional to the number of multicast nodes, the running time of KPP relative to STAR(1) decreases when the number of multicast nodes increases.

Fig. 14 shows the running time comparison for large networks. Only SLIM+ can be simulated since it is the fastest among the other algorithms. The speed advantage of STAR over SLIM+ is obvious.

Fig. 15 shows the result of some other delay-cost distributions. RcPd is what we use as the default. In [1,2] a model, which we call FcPd, is used. In that model, the link delay is the geometric distance, and the link cost is a random fraction of the link delay. In [16], yet another different model is used, which we call PcRd, in which the link cost is the geometric distance, and the link delay is random. For completeness, we add a fourth model, IcPd, in which the link delay is the geometric distance, and the link cost is randomly proportional to the inverse of link delay. Fig. 15 shows that STAR performs well in all the delay-cost distribution conditions.

As shown in Fig. 15, STAR has large advantages over the other algorithms when the IcPd distribution is assumed. Fig. 16 shows that the advantage is even larger when the network

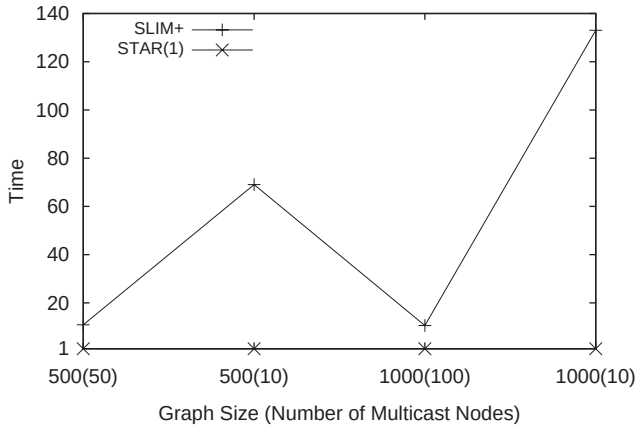

Fig. 14. Effect of network size on time.

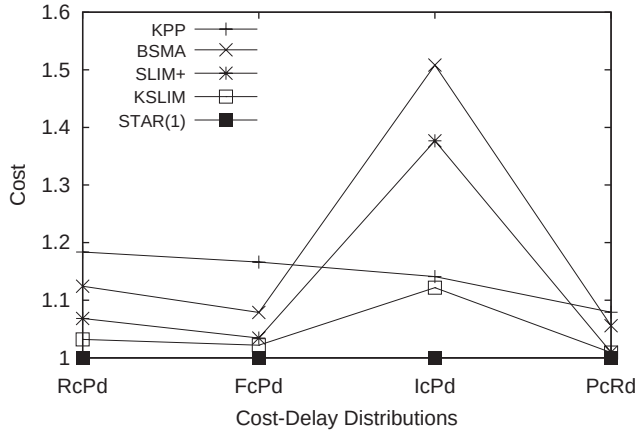

Fig. 15. Effect of delay-cost distributions on cost.

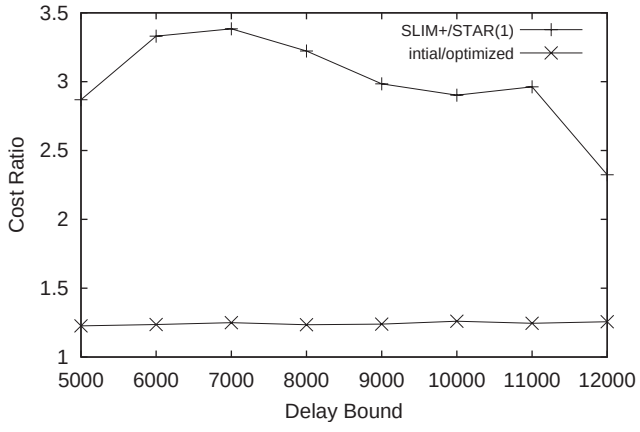

Fig. 16. Comparison in network of 1000 nodes and IcPd distribution.

is larger. The cost ratio between SLIM+ and STAR can be as large as 3. We also plot the cost ratio of STAR before/after *TrimAndReconnect* is called. It can be seen that STAR's paths have lower cost than SLIM+'s, since optimization only reduces the original cost by less than 30%.

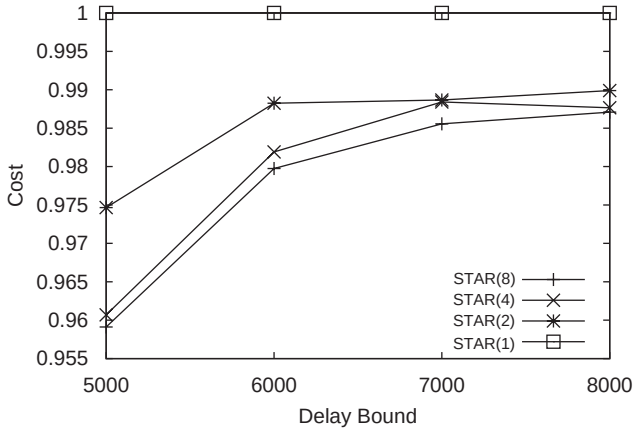

Fig. 17. Effect of delay bound on cost.

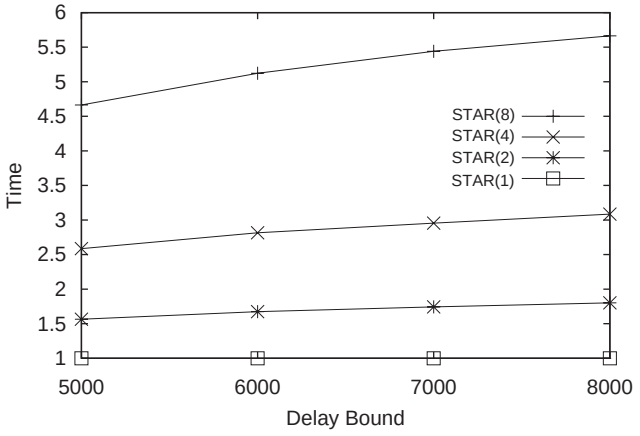

Fig. 18. Effect of delay bound on running time.

In all the results shown so far, STAR uses $\lambda = 1$. The following graphs show the results when more delay ranges are used($\lambda > 1$).

Fig. 17 shows the cost as a function of the delay bound. In STAR(λ), *DtoI* partitions the delay bound into λ ranges and the `DBMC paths` are computed for every range. The effect of the number of segments becomes obvious when $\delta < 7000$. Note that the delay is equal to the distance in our simulations and 7000 is a characteristic number equal to the width plus the length of the simulated area.

Fig. 18 shows the running time overhead of using more than one segment. Combined with Fig. 17, it can be concluded that even when $\delta > 7000$, it may still be worthwhile to partition the delay bound into 2 segments. This will take 50% more running time to decrease the cost by 1–2% when the delay bound is loose, and 3% when it is tight.

Compared with the results shown in Fig. 17, Fig. 19 shows that segmentation is even more effective when the delay bound is tight and IcPd is the distribution of link costs and delays. First, since the link cost is randomly inversely proportional to the link delay in IcPd, it is less likely that the minimum cost path is the minimum delay path. Hence there

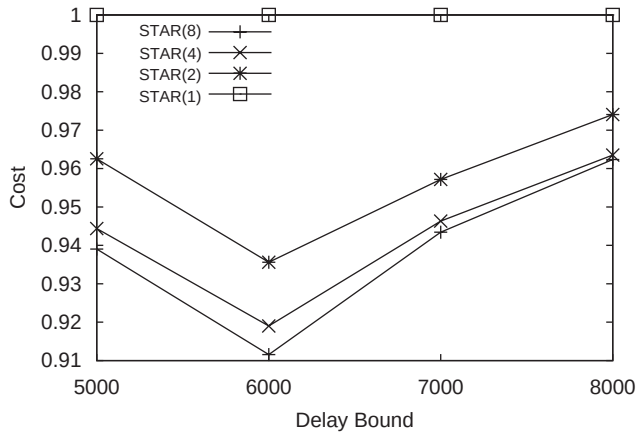

Fig. 19. Effect of delay bound on cost in IcPd mode.

is more need to trade off the costs with the delays of the paths and to store the best paths for different delay requirements. Second, the segmentation of delay bounds is most effective when the delay bound is neither too large nor too small. As noted in Section 4.1, if the delay bounds are too large the DBMC paths will always be the minimum cost path; if they are too small, the DBMC paths will not be found anyway.

In summary, we have demonstrated the effectiveness of the two major techniques of STAR(x). The effect of 'Trim and Reconnect' is demonstrated first with $x = 1$, and the effect of 'Segmentation' is demonstrated by considering $x > 1$. Segmentation finds the optimal DBMC paths while 'Trim and Reconnect' re-structures the topology of the multicast tree. The optimality of the algorithm depends on the effectiveness of both techniques whereas the relative contribution of each technique may be different for different situations. For example, if the cost of the link is randomly proportional to the delay, then the path with the shortest delay is likely to be the path of the lowest cost. In the case of random cost and delay distribution, it can be argued as well that the path with the smallest number of links is approximately the path of the lowest cost and the lowest delay. In these cases, as long as the delay bound is not tight, using DBMC paths will not help much in reducing the cost. Instead, re-structuring the tree is more important. As shown in Fig. 9, KPP and KSLIM, which select better DBMC paths than STAR(1), still produce trees with larger costs, and the cost gap increases with the number of multicast nodes and the complexity of the multicast trees.

When the delay bound is tight, the effect of fluctuation of cost and delay cannot be overlooked, as many paths of low costs will be disqualified for breaching the delay bound. These trends are shown in Fig. 10, as the performance of KPP, BSMA and KSLIM improves with the tightening of the delay bound, while the performance of SLIM+ deteriorates with the tightening of that bound. The importance of using DBMC paths can also be demonstrated by comparing the results of STAR(1) with STAR(x) ($x > 1$). As shown previously, when the delay bound is tight or if the cost of a link is likely to be low when the delay is high, finer segmentation results in better approximation of DBMC paths and thus better cost of the multicast tree.

7. Conclusion

The DBMC multicast problem is about conserving the network resources without compromising the QoS. STAR constructs the multicast tree assuming that the complete topology of the network is available. The key idea of STAR is that if the interaction among a pair of non-multicast nodes is not essential, then there is no needs to compute the path connecting them. Based on this idea, we proposed an optimization scheme which only requires the paths originating from the multicast nodes. Since, by definition, the set of multicasting nodes is a subset of the nodes in the network, STAR always takes less time to compute and less space to store the path information than the all-pair-shortest-path algorithms.

STAR optimizes the multicast tree by trimming and reconnecting subtrees containing multicast nodes. Although it adopts the same iterative scheme as other algorithms [1,2,16], it is more flexible in finding the substituting subtree. Thus the probability of optimization is increased. The result is that, STAR can be two orders of magnitude faster and yields multicast trees with lower costs.

The path computing procedure of STAR partitions the delay bound as KPP does[12] except that it is based on *Dijkstra's Algorithm*. This algorithm is as optimized as the approaches based on computing the shortest paths between all pairs of nodes. Yet it allows STAR to compute only those paths originating from the multicast nodes. Moreover, in a network where all the multicast nodes have access to the global network topology, each node can perform the path computation in parallel, after which the nodes can send the results to the source for optimization. Since computing paths takes more time than the optimization, parallel path computing may further reduce the computation time of the multicast tree. This also allows new nodes to join and leave the multicasting group dynamically.

STAR has been tested for different network topologies, delay-cost distributions, delay bounds, network sizes and multicast sizes. These parameters do affect the simulation results to various degrees. However, STAR performs well in all situations, and thus is a good candidate for the general solution of this problem that is related to various application areas, such as teleconferencing, network management, etc. STAR can be applied directly to a simple network, or in the context of a large hierarchical network. In the latter case, each multicast node may view the network with different details. Thus it makes more sense that each multicast node computes the shortest paths based on its own partial information. The computed paths, although only based on incomplete information of the network (thus may be subop-

timal), are valid paths that can be sent to a common node to construct and optimize the multicast tree.

References

[1] T. Alrabiah, T. Znati, An efficient and easily deployable qos-based routing scheme for "online internet multicasting", J. Comput. Comm. (Special Issue of Computer Communications on Integrating Multicast into the Internet) 24 (2001) 496–511.

[2] T. Alrabiah, T. Znati, Delay-constrained, low-cost multicast routing in multimedia networks, (Special Issue, "Routing in Computer and Communication Systems"), J. Parallel Distrib. Comput. 61 (9) (2001) 1307–1336.

[3] T. Ballardie, P. Francis, J. Crowcroft, Core-based trees (CBT) an architecture for scalable inter-domain multicast routing, Comput. Comm. Rev. 23 (4) (1993) 85–95.

[4] E.W. Dijkstra, A note on two problems in connection with graphs, Numer. Math. 1 (1959) 269–271.

[5] C. Diot, B.N. Levine, B. Lyles, H. Kassem, D. Balensiefen, Deployment issues for the ip multicast service and architecture, IEEE Network 14 (1) (2000) 78, 88.

[6] M. Doar, I. Leslie, How bad is naive multicast routing, in: Proceedings of IEEE INFOCOM, San Francisco, CA, 1993, pp. 82–93.

[7] C.W. Duin, A. Volgenant, Reduction test for the steiner problem in graphs, Networks 19 (1989) 549–567.

[8] D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, P. Sharma, L. Wei, Protocol independent multicast-sparse mode (PIM-SM): protocol specification, Internet RFC 2117, http://ds.internic.net/rfc/rfc2117.txt.

[9] R. Gau, Z. Haas, B. Krishnamachari, On multicast flow control for heterogeneous receivers, IEEE/ACM Trans. Networking 10 (1) (2002) 86–101.

[10] S. Kasera, S. Bhattacharyya, M. Keaton, D. Kiwior, S. Zabele, J. Kurose, D. Towsley, Scalable fair reliable multicast using active services, IEEE Network 14 (1) (2000) 48, 57.

[11] S. Khuller, B. Raghavachari, N. Young, Balancing minimum spanning trees and shortest-path trees, Algorithmica 15 (14) (1995) 305–321.

[12] V. Kompella, J. Pasquale, G. Polyzos, Multicast routing for multimedia communication, IEEE/ACM Trans. Networking 1 (3) (1993) 286–292.

[13] L. Kou, G. Markowsky, L. Berman, A fast algorithm for steiner trees, Acta Inform. 15 (1981) 141–145.

[14] R. Meddeb, A. Girard, C. Rosenberg, The impact of point-to-multipoint traffic-concentration on multirate networks design, IEEE/ACM Trans. Networking 10 (1) (2002) 54–66.

[15] J. Moy, Multicast routing extensions for OSPF, Comm. ACM 37 (8) (1994) 61–66.

[16] M. Parsa, Q. Zhu, J. Garcia-Luna-Aceves, An iterative algorithm for delay-constrained minimum-cost multicasting, IEEE/ACM Trans. Networking 6 (4) (1998) 461–474.

[17] T. Pusateri, Distance vector multicast routing protocol, Internet Draft.

[18] S. Raghavan, G. Manimaran, C.S.R. Murthy, Algorithms for delay-constrained low-cost multicast tree construction, Comput. Comm. 21 (18) (1998) 1693–1706.

[19] H. Salama, D. Reeves, Y. Viniotis, Evaluation of multicast routing algorithms for real-time communication on high-speed networks, IEEE J. Selected Areas Comm. 15 (3) (1997) 332–334.

[20] Q. Sun, H. Langendoerfer, Efficient multicast routing for delay sensitive applications, Proceedings of Second Workshop Protocols Multimedia Systems (PROMS'95), 1995, pp. 452–458.

[21] Q. Sun, H. Langendoerfer, An efficient delay-constrained multicast routing algorithm, J. High-Speed Networks 7(1) (1998).

[22] B. Wang, J. Hou, Multicast routing and its qos extension, IEEE Network 14 (1) (2000) 22, 36.

[23] B.M. Waxman, Routing of multipoint connections, IEEE J. Selected Areas Comm. 6 (9) (1988) 1611–1622.

[24] S. Yan, M. Faloutsos, A. Banerjea, Qos-aware multicast routing for the internet: the design and evaluation of qosmic, IEEE/ACM Trans. Networking 10 (1) (2002) 54–66.

Shu Li got his BS in Peking University, China and his PhD in Chemistry and MS in Computer Science both in University of Pittsburgh, PA. After graduation in 2001, he started to work as a software engineer in CoManage Corp., PA. He then worked as a research assistant at the University of Pittsburgh. His research involved algorithms for multicasting problem and simulation of power aware real time systems. In 2003, Shu Li entered Prof. Burlingame's lab in UCSF to improve the mass spec search engine for protein identification (Protein Prospector). He is now with Arqule, Inc., MA, working on high-through analytical chemistry.

Rami Melhem received a B.E. in Electrical Engineering from Cairo University in 1976, an M.A. degree in Mathematics and an M.S. degree in Computer Science from the University of Pittsburgh in 1981, and a Ph.D. degree in Computer Science from the University of Pittsburgh in 1983. He was an Assistant Professor at Purdue University prior to joining the faculty of The University of Pittsburgh in 1986, where he is currently a Professor of Computer Science and Electrical Engineering and the Chair of the Computer Science Department. His research interest include Real-Time and Fault-Tolerant Systems, Optical Networks, High Performance Computing and Parallel Computer Architectures. Dr. Melhem served on program committees of numerous conferences and workshops and was the general chair for the third International Conference on Massively Parallel Processing Using Optical Interconnections. He was on the editorial board of the IEEE Transactions on Computers and the IEEE Transactions on Parallel and Distributed systems, and is currently on the editorial board of the Journal of Parallel and Distributed Computing and the Computer Architecture Letters. He is serving on the advisory boards of the IEEE technical committees on Parallel Processing and on Computer Architecture.

Taieb F. Znati received a Ph.D. degree in Computer Science from Michigan State University in 1988, and a M.S. degree in Computer Science from Purdue University, in 1984. He is currently a Professor in the Department of Computer Science, with a joint appointment in Telecommunications in the Department of Information Science. Dr Znati's current research interests focus on the design of network protocols for wired and wireless communication networks to support applications' QoS requirements. Dr. Znati currently serves as general chair of IEEE INFOCOM 2005, general chair of SECON 2004, the first IEEE conference on Sensor and Ad Hoc Communications and Networks, general chair of the Annual Simulation Symposium, and general chair of the Communication Networks and Distributed Systems Modeling and Simulation Conference. Dr. Znati is a member of the Editorial Board of the International Journal of Parallel and Distributed Systems and Networks, a member of the Editorial Board of the Journal on Wireless Communications and Mobile Computing, the Journal on Ad-Hoc Networks, and a member of IEEE Transactions of Parallel and Distributed Systems, and Wireless Networks, the Journal of mobile communication, computation and information. He is currently serving as a Senior Program Director for networking research at the National Science Foundation.