Extended Core-Based Framework for Delay-Constrained Group Communication

Ayşe Karaman and Hossam Hassanein

Telecommunications Research Lab
School of Computing
Queen's University
Kingston, ON K7L 3N6, Canada
{karaman, hossam}@cs.queensu.ca

Abstract
Multipoint communications is the simultaneous transmission of data streams from a number of sources to a set of receivers in a group according to predetermined metrics. The core-based approach in multipoint communication enhances potential solutions in terms of QoS-efficiency and feasibility of the results in inter and intra-domain routing. In this paper, we first analyze the solution space for constrained multipoint communication problems under the core-based approach. We show that the range of solutions examined by the models proposed to date is restricted to a subset of the entire solution space, which restricts the potential efficiency of the results. We propose *SPAN*, a core-based framework processing on our identified extended solution space for constrained multi-source group applications. *SPAN* consists of core selection and tree construction as two modular components complimenting one another to achieve more efficient solutions in distributed processing. *SPAN* is also asymmetric, hence potentially operates in domains in which link weights are not necessarily equal in both directions. We analyze the computational and message complexity of our framework and show its feasibility for distributed deployment. Our performance results show that SPAN consistently outperforms its counterparts in the literature in terms of cost and QoS-efficiency.

1. Introduction

Multipoint-to-multipoint communication, or multipoint communication as we refer to throughout the text, is the simultaneous delivery of data stream from a number of sources to a set of receivers for more efficient transmission. A multipoint group is a collection of nodes that are receivers of the same sequence of messages (data streams). These messages may or may not belong to the same source node. Multipoint groups may be static (with no membership updates) or dynamic (where nodes can join/leave the group dynamically). Multipoint communication has numerous applications in Internet group communication. The transmission of the multimedia streams is demanding on network resources due to the size and potential priority of the application flow — calling for the minimization of the cost of transmission. As well, multimedia communications are usually characterized by Quality-of-Service (QoS) constraints in that they are often delay-sensitive. Effective group communication solutions must, therefore, provide cost efficient transmission of the stream to each receiver in the group within a given delay bound. The prominent QoS problem then is constrained cost-minimization — minimization of the total cost of the transmission while meeting the delay-bound of the application.

Dominant unicast routing platforms on the Internet provide local information to domain nodes so that the information available to each router is only the value of the minimum-distance path from itself to each other router in the domain. The knowledge of the entire domain is unavailable to any particular router, and thus the processing environment is distributed. Our context in this research is routing for multipoint communication under constrained cost minimization in distributed networks for multi-source groups.

The objective of multipoint routing is to build most efficient paths for the flows to be delivered to receivers. This includes the construction and operation of paths in a tree structure. Multicast trees in early protocols were rooted at the source of the stream. Latter protocols, proposed core-based trees rooted at a "core" node in the domain, which is not necessarily a source any more. The main motivation behind the core-based approach is scalability for sparsely distributed multicast groups and reduced tree maintenance overhead. The core-based approach for route construction provides improved efficiency for constrained applications for sparsely populated groups in the distributed routing platform [CZD95, WE94].

Core-based routing architectures in literature considered the minimization of the number of cores during the placement of cores in the domain [Sa96, ZFL02]. Simpler route structures in terms of the number of links constituting the routes lead to reduced exploitation of the paths and network resources, higher reliability, reduced tree maintenance cost, and flexibility of the path under topological instabilities and path updates due to dynamics of the group memberships. However, minimum number of cores does not necessarily lead to minimal path structures. Note here that, minimization of the number of links on a multipoint communication path accounts for each link on the path regardless of their exploitation by multiple sources in the group unlike cost minimization which accumulates cost of each link across their utilization by the sources. In this study, we consider the total hop-count on the group communication trees as an optimization objective in addition to cost minimization. In an attempt to achieve this, we consider maximal link sharing across the sources, as alternative to minimal cost of transmission of the stream. We collectively name the constrained optimization problems in multi-sources groups as delay-constrained multipoint communication (DCMC) throughout the text.

In this paper, we investigate solutions for delay-constrained multi-source, multipoint communication groups applying the core-based architecture. Our analysis of the solution space for this problem indicates a broader range which is not explored by existing models. We identified the solution space as being composed of singular and non-singular spaces, with all existing schemes exploring only the singular solution space. In the singular space, each core, serves all sources in the group and uniquely defines a shared tree. In the non-singular space, a core is not necessarily serving all the sources and each receiver can be served by multiple cores varying across sources in the group. We introduce *SPAN*, a distributed model processing in the non-singular space and extending the range of solutions searched by its precedents. Our model describes the first distributed, asymmetric framework on the literature to provide solutions for constrained, core-based multi-source communication groups. Our empirical results show the high efficiency of our model compared to its counterparts.

The main contributions of this paper are:

1) We identify the range of solutions to a DCMC problem in the core-based approach. We characterize the solution space into singular and non-singular solution spaces which partition

the entire solution space. We show that the non-singular solution space, which is not explored by the models in the literature to date, offers potentially more efficient results.

2) We develop *SPAN*, a generic core-based framework for DCMC solutions. SPAN consists of core selection and tree construction as two modular components to compliment one another sequentially to achieve more efficient solutions in distributed processing. Our empirical results show that SPAN consistently outperforms its counterparts in the literature clearly demonstrating the significance of our theoretical findings indicating the potential contribution of models processing in the extended range of solutions.
3) We analyze the computational and message complexity of our framework and show its feasibility for distributed deployment.

Within the core-based approach, our framework separates the DCMC problem into two parts: core selection and tree construction. Our model, SPAN, first selects the cores to lead to a set of shared-trees for efficient multipoint communication. The tree-construction module then takes over for building the ultimate multipoint communication path. Our motivation is not only reducing the complexity of the solution into two different processes, but also splitting the complexity of management of the resulting paths into core-dominated tree portions by delegating this task to the cores from the sources in one process rather than multiple processes across sources. Our proposed framework, *SPAN,* has the following characteristics:

1) Spans the extended solution space for both singular and non-singular solutions for the results.
2) Distributed: feasibly executes on local distance-vector information on the nodes with no reliance on an external protocol.
3) Asymmetric: The model considers, during the route construction phase, the link directions in their utilization throughout the actual communication session and hence effectively optimizes for solutions in asymmetric domains.
4) Supports processing on multiple, possibly QoS-based, metrics, and, to this end, is the first distributed QoS model in the literature to provide constrained, distributed, core-based solutions for group communications.
5) Modularly incorporates the core selection and tree construction components into a routing framework in which each component can be further elaborated separately achieving QoS-efficiency of the solutions.

This paper is organized as follows. In the next section, we initially layout the problem description and potential approaches to the solutions. We then review previous work, and introduce the terminology used in the paper. In Section 3, we analyze the solution domain for DCMC problems. We first characterize the potential solutions, and identify the limitation of the solution explored by the existing models in the literature. We then prove that, the extended solution space offers higher efficiency for wide range of DCMC problems thereby provides the potential for improving the efficiency of the protocols operating in the extended space. Section 4 introduces the proposed *SPAN* framework. Following an architectural overview, we present algorithmic descriptions of the core selection and tree construction components of the model, in Sections 4.2 and 4.3, respectively. In Section 4.4, we discuss the distributed deployment of *SPAN* to complete its operational description. In Section 4.5, we analyze the computational complexity *SPAN*, as well as its message communications overhead during protocol operations. A performance model for our DCMC framework is described in Section 5. Performance Results show that *SPAN* highly outperforms its counterparts in the literature, as well as its “dual”, *SINGULAR*, that processes within our architectural framework, but within the restricted solution space used in existing modes. We conclude the paper in Section 6.

2. Preliminaries

2.1 Problem Domain

In multipoint communication data packets from some source need to visit each one of its destinations once throughout a group session. Loops are neither required nor allowed on their routes, and the communication path has the structure of a tree or multiple trees. Multicast trees in early protocols were rooted at the source of the stream. Latter protocols proposed core-based trees each rooted at a "core" node in the domain, which is not necessarily a source. In either class of architectures, the problem breaks into the construction of a tree rooted at a given node to span a given set of leaves to meet the design criteria.

One way of meeting the delay-bound of applications in delay-constrained communication groups is to build point-to-point delay-shortest paths between source and separately each receiver. This is the well-known *shortest path tree* (SPT) problem and has polynomial solutions [D59, B57, FF62]. The trees constructed in all multipoint communication protocol suites [B97, B97b, DEFJ94, M94, M94b, WPD88] widely accepted so far are shortest path trees. Note that the problem of constrained-SPT, attempting to approximate the constrained-cost minimization problem to minimize the delay-bound cost-distances from the tree root separately to each receiver is NP-complete as well [GJ00]. Furthermore, route construction problems optimizing on any two separate constraints are naturally NP-complete [WC96].

In delay-constrained cost minimization, the problem is *constrained-SMT*, which is the minimization of the sum of on-tree link costs, this time also meeting the condition that no receiver is beyond a given delay-distance from the tree root. Constrained-SMT is also NP-complete since SMT, a known NP-complete problem [K72], reduces to it through a transformation introducing an "unbinding" delay bound. Among the constrained-SMT heuristics proposed in the literature [CHH97, KPP93, KPP93b, J98, PZG98, WC95], those processing on full domain information generated results close to optimal solution. In particular, BSMA, a centralized algorithm proposed by [PZG98] returned the most efficient results [SRV97] for the minimization of total tree-cost on successfully constructed delay-constrained trees. Wider availability of domain information results in better approximations close to optimal. Therefore, high-performing constrained-SMT heuristics are centralized, and are not feasible for distributed implementation.

The core-based approach enhances potential solutions by efficient placement of cores in the domain especially for groups sparsely distributed in the domain. In this approach, the delivery trees are not necessarily rooted at a source any more, but at a domain node that ranks high in efficiency with respect to the operating metrics. Cost-efficiency and reduced maintenance overhead of the communication path are achieved through simplified and separately managed tree structures. The approach also offers the placement of cores in distinct autonomous systems allowing tree management "locally" on information available within an Autonomous System (AS) within the domain. Each core is a known reference to the path for the potential member nodes throughout the domain. The more the number of references are, the more the number of alternative connection points to the multipoint path for new joining members who may select a path based on topological proximity and cost efficiency of potential connections. Furthermore, multiple cores can provide a solution for a multipoint communication group where no feasible solution exists in the single core scheme due to delay-bound violations for multi-source applications. The multi-core approach, returning a number of cores in the domain improves the solution range of and is strictly preferable to the single-core approach. Previous studies sub-optimized the number of cores and hence the shared-trees for the optimization of path structure [Sa96, ZFL02]. However, minimal number of cores does not necessarily lead to minimal number of links. We argue that the number of links is a more accurate measure of the path structure, which is particularly significant for groups with dynamic memberships [DZ96], and attempt to minimize of the total number of links on the multipoint path as an additional goal. The problem of constrained link minimization differs from constrained cost minimization since, unlike the delivery cost, link count, is not cumulative over the sources and attributes to the overall communication path. Link count is a measure of overall multipoint path structure irrespective of the utilization of the links by distinct sources. Cost, on the other hand, measures the link utilization across the sources in the group. Constrained link-minimization is still NP-complete since SMT on even link weights, which also is NP-complete [W87], reduces to it with a transformation similar to the above.

2.2 Previous Work

The multipoint routing problem has initially been studied in the multicasting domain for transmission from one source to a given set of receivers so that delay-distances are minimized. Latter protocols proposed core-based trees, this time rooted at a core node in the domain, which is not necessarily the source. Earlier core-based models restricted to single-core solutions [B97, B97b, DEFJ94]. Further research focused on multi-core trees for the improvement of tree accessibility to the multicast sources and new receivers, regardless of the number of sources in the group. OCBT [SG97] is the first multi-core architecture in the literature. In its design specifications, OCBT develops a unique shared tree which contains multiple cores, and emphasizes on maintaining the tree structure as the tree is updated during protocol operations, diverting from the efficiency concerns of path construction and maintenance throughout the protocol operations. The model disregards a delay bound of the application and is not applicable for DCMC problems. Zappala, Fabbri and Lo [ZFL02], in their *Senders-to-Many* architecture, partition the receiver set among a set of cores and generate a unique core cluster to serve the group. Senders-to-Many is distributed and does not consider a given delay-bound. Each core and the receivers in the partition corresponding to that core constitute a tree rooted at the core. A source willing to deliver its stream simply sends its data packets to the core from whereon they are transmitted to the receivers via the core-trees. In [KH04], we provide a detailed review of these protocols.

Salama's [Sa96] architecture operates similarly to Senders-to-Many in that it develops a set of core-based trees each spanning a receiver partition, this time meeting a given delay bound in the solution generated if such a solution exists. In both cases, a set of core trees are formed, which combined together span all the receivers in the group. These core trees are shared by all sources in the group for the delivery of their stream to the receiver set.

The scheme in [ZFL02] used hop-count as the cost metric in minimizing the number of cores. Salama's multi-core based algorithm [Sa96], known as *GREEDY*, considers multiple sources in the group among its design objectives, and generates constrained solutions that meet a given delay-bound. The model develops core trees each spanning a receiver partition for all the sources in the group. Salama's solution is particularly relevant since it is the only routing architecture in literature providing multi-core solutions for delay-constrained multi-source communication groups. A distinctive property of *GREEDY* is that, it assumes bi-directional trees during core selection so that a data packet initiated from a source on the tree reaches a particular receiver via the shortest-delay path on the tree with no need for visiting the root of the tree on its way to the receivers. A data packet incoming to a tree node is forwarded to every other on-tree link regardless of whether the outgoing link(s) lead to receivers as further destinations. That is, the on-tree source of the stream acts as the tree root during the transmission of its stream. Consideration of bi-directional trees during design widens the solution space for delay-constrained problems. However, bi-directional tree design considers the delay-distances not only between the candidate cores and group members but also between the node pairs including any domain nodes. This restricts the feasibility of the algorithm strictly to centralized deployment. For the same reason, the model optimizes on transmission in both directions of a link, and is ineffective in asymmetric networks. *GREEDY* operates on delay as a single metric, with no consideration of cost or hop-count in tree formation. Indeed, both *GREEDY* and Senders-To-Many select cores without consideration of additional metrics.

2.3 Terminology

The dominant unicast routing platform on the Internet provides local information to domain nodes so that the information available to each router is only the values of the minimum-distance path from itself to each other router in the domain separately for each metric. We denote by $path_{<distance>}[i,j]$ the <*distance*>-shortest path, i.e., the sequence of nodes connecting the nodes i and j at minimum possible distance in unit specified in parameter <*distance*>. We extend this notation with the indication of <*metric*> as $path_{<distance>}[i,j].<metric>$ to return the "length" of the path, i.e., the sum of the weights of the links on this path in the unit <*metric*>. According to this, $path_{delay}[i,j].delay$ and $path_{cost}[i,j].delay$ return the point-to-point delay between i and j along the shortest-delay and shortest-cost paths connecting the nodes, respectively; whereas $path_{hop}[i,j].cost$ returns the cost of the hop-shortest path between the nodes. Similarly, we describe $path_{<distance>}[j].<metric>$ as the unicast table look-up for the value $path_{<distance>}[i,j].<metric>$ on the processing node i. We assume that the unicast routing platform makes available to any given domain router i the values $path_{<distance>}[i,j].<metric>$ in all combinations of <*distance*> and <*metric*> for each router j in the domain along with the immediate neighbor of i leading to this path. The knowledge of the path itself, i.e., the sequence of nodes constituting it is not available to router

We say that a receiver r is *dominated by* a core c *for* a particular source s if there exists a path p connecting s to r so that p passes through c without violating the delay bound. Equivalently, we say that a core c serves r_s. We use the notation $D(c,s)$ to indicate the set of receivers that are dominated by the core c for source s. Similarly, we indicate by $D(c,S')$ the domination of a set of receivers that are dominated by the core c for each source in a subset S' of the set of sources. We refer by *multipoint tree* the union of paths serving a $S' \subseteq S$ for all receivers in the group where each core tree constituting the path is identically serving all sources in S'. We call the set of core trees constituting a multipoint tree as a core cluster. On a multipoint path, the domination sets of the cores in the core cluster describing the path partitions the receiver set. Observe that a multipoint tree does not necessarily have a tree structure although it specifies distinct trees each of which is serving a particular source stream. In Figure 1-a, we present an example for a multipoint tree. The source and receiver sets in the multipoint communication group are $S=\{s_1,s_2\}$ and $\{r_1,r_2,r_3,r_4,r_5\}$ respectively. There are 3 cores in the group. Domination sets are $D(c_1, S) = \{r_1,r_2\}$, $D(c_2, S) = \{r_3\}$, $D(c_3, S) = \{r_4,r_5\}$. There are three core-rooted trees, each spanning the receivers in the domination sets of their respective cores. The multipoint tree is serving both sources in the group. There are two source-rooted trees, one for each source, each spanning the entire cores in the core-cluster corresponding to the multipoint tree. The links on source and core trees are indicated by solid and double lines, respectively. All links except those indicated by dashed line are on the multipoint path. Figure 1-b presents an alternate solution to the same problem in which different trees with non-identical core trees serve the sources. All links except (s_2,c_2) indicated by the dotted line specify the solution paths in this case. The links serving sources s_1 and s_2 are indicated by solid and dashed lines, respectively. Double lines indicate links serving both sources. The potential domination sets are $D(c_1, S) = \{r_1,r_2\}$, $D(c_2, s_1) = \{r_3, r_4\}$, $D(c_2, s_2) = \{\}$, $D(c_3, s_1) = \{r_5\}$, $D(c_3, s_2) = \{r_3, r_4,r_5\}$. Unlike the case in Figure 1-a, the receiver set is partitioned differently by the core clusters for each source, with one partition and the dominating core, c_1, common to both. Note that any solution to a DCMC problem can be formulated as a set of multipoint trees to specify the set of cores to be spanned on each source tree, and the set of receivers to be spanned on each core tree.

Consider a set of core-trees serving a particular multipoint communication group so that each core is serving a non-empty subset of source set. Alternatively, consider the class of solutions in which each core tree is serving the entire set of sources. In this case, a unique core-cluster – a unique set of core trees identically serves all sources in the group. That is, each core tree in a particular set of core trees is used for delivery of the stream by a source in the group as it is used by any other source. In the former case, the group is served by multiple core-clusters each serving a subset of sources. DCMC solutions in core-based architecture spanned solely the solutions involving a single core cluster across sources. As we show later, the efficiency of potential solutions considerably improves when the entire solution space is examined.

3. Solution Domain

In this section, we examine and separate the solution space of DCMC problems into two disjoint sets for the purpose of characterizing the solution heuristics. Before going further, we update the definition of a multipoint tree to exclude the case where the core cluster overlaps exactly with the receiver set. A multipoint tree is still defined as the union of core and source trees so that all core trees serve all the sources on the tree collectively for all receivers. In addition, we impose the condition that the set of cores in the cluster corresponding to the tree is not equal to the set of receivers, disallowing the domination set to characterize the tree being $\{ D(r_i,S)=\{r_i\}$ for all $r_i \}$. We call the set of solutions satisfying this property as *identity solutions*. Note that any set of paths to result into a feasible solution can be characterized as an identity solution, and any identity solution can also be characterized as a solution involving one or more multipoint trees. Any identity solution reduces the problem to a solution in source-based domain since an identity solution specifies no core-trees. Note that source-based and identity solutions apply tree construction on the same input—each tree to be constructed is rooted at a source and spans the entire set of receivers. The difference between the two solutions is that in the source-based case, the trees are core-trees and each source is serving itself as the core. The identity solution still defines source trees, with the cores being the receivers. In the core-based approach, the source-rooted solution with sources serving as the cores is within the solution space. The exclusion of identity solutions does not restrict the solution space for the set of feasible multipoint paths. On the contrary, the classification imposed by our definition forces the consideration of links which are shared by multiple receivers. As an example, the multipoint communication path in Figure 2 is characterized to include two multipoint trees, each serving one source. The trees are not overlapping on any link. The domination sets are $D(s_1, s_1) = D(s_2, s_2) = \{r_1, r_2\}$. Both sources are serving themselves as the cores.

We define a *feasible* solution as any non-identity solution, which connects all source-receiver pairs in the multipoint communication group without violating the delay bound. According to this definition, an optimal solution is the feasible solution, which returns the optimal value with respect to the optimization criterion. We partition the set of feasible solutions for a given DCMC problem into the following:

1) *Singular solution space:* the set of all feasible solutions that involve only one core cluster tree to serve the entire multipoint communication group.
2) *Non-singular solution space*: the set of all feasible solutions that involve two or more core clusters.

The above spaces clearly partition the entire solution space with the exclusion of identity solutions.

According to this, the solution in Figure 1-a is in the singular solution space where the ones in Figure 1-b and Figure 2, containing more than one set of core trees each serving a source node, qualify to be non-singular solutions. In the singular space, each core, serves all sources in the group and uniquely defines a shared tree. In the non-singular space, a core is not necessarily serving all the sources and each receiver can be dominated by multiple cores varying across sources in the group. Note that the union of source-rooted trees each spanning the entire receiver set is naturally in the non-singular solution space aside from the exceptional cases in which one of the sources is serving as the core for the entire source set including itself for the entire receiver set.

Existing solutions to the multipoint communication problem, which we reviewed in Section 2 only span the singular and identity solution spaces. However, as we show in the following proofs, the non-singular solution space can contain more efficient solutions. Indeed, the optimum solution can lie in the set of non-singular solutions for many DCMC problems.

Lemma 1. If a constrained-multipoint communication problem is known to have a feasible solution, this does not imply the existence of a solution in the singular space.
Proof: Any graph in which the feasible paths between the sources and receivers are not overlapping across multiple sources disproves the counter-argument to this lemma (see Figure 2 for a sample case). Aside from the connectivity of the graph, solely the link delays under the given delay constraint can restrict the solution exclusively to the multiple tree solution space of which a generic example is given in Figure 3. T_1 and T_2 in Figure 3 are network clusters each containing a disjoint set of sources in the group. Similarly, T_3 contains a set of receivers. T_1, T_2, and T_3 are only connected to one another through the subnetwork illustrated in the figure. Each link label I and J indicate exactly one value in the range $[i_1, i_2]$ and $[j_1, j_2, j_3, j_4]$ respectively. The delay-distance between the multipoint group members in T_1, T_2 and T_3, and the nodes connecting each of these clusters to the subnetwork of nodes n_1 through n_4 are restricting the maximum delay-distance for a path among nodes n_1 through n_4 to Δ' in order to meet the delay bound of the application. A solution in the singular space requires either one of the nodes n_1 through n_4 be the core to serve the sources in both T_1 and T_2 for the receivers in T_3. Whenever $j_n > \Delta'$ for all n, any solution is in the non-singular solution space. In this case, when $i_m \leq \Delta'$ for m=1..2 additionally, there is a feasible solution for the group connecting the sources in T_1 to receivers in T_3 through the path n_1-n_3, and the sources in T_2 to receivers in T_3 through the path n_2-n_4. These two paths do not overlap for a single core to serve both source clusters for the same set of receivers and the solution is in the non-singular space, justifying the proof. □

Lemma 2. If there exists a singular solution to a constrained-multipoint communication problem, then the optimum solution is not necessarily in the singular solution space.
Proof: The multipoint communication group illustrated in Figure 3 provides a generalized case to prove Lemma 2 as well. This time the link values are link costs, and no delay-bound being set. Note that the path costs in the figure can equivalently be accounted as hop-count between the nodes defining the link, without affecting the accuracy of our proof. Under the conditions specified, the optimum solution is unique and in the non-singular solution space. □

The search for solutions in the singular solution space examines the domination of each receiver for all sources in the group. An algorithm to span the solutions in the non-singular space, on the other hand, examines the domination sets separately for each source rather than across all sources. However, Lemmas 1 and 2, show the potential existence of feasible and optimum solutions outside the singular solution space regardless of the existence of solutions in this space for a particular constrained-multipoint communication problem. This supports the fact that a heuristic to provide a solution to this problem in the entire solution space is considerably preferable to a heuristic to provide a solution in the singular space. Current solutions provided for the problem span solely the singular and identity solution spaces. In the next section, we describe a generic framework for DCMC problems in which we search for solutions in non-singular, as well as singular solution spaces. We call this framework *SPAN*, as it spans the entire solution space.

4. Architectural Description

This section describes *SPAN*, our framework for constrained cost minimization solutions for multi-source groups in the core-based approach. The framework is distributed, asymmetric and processes on non-singular as well as the singular solution spaces. In Section 4.1, we outline the overall architecture. In Sections 4.2 and 4.3, we respectively present the core selection and tree construction components of our design. We describe the distributed implementation of *SPAN* in Section 4.4. Section 4.5 includes the complexity analysis of our model.

4.1. Design Overview

Within the core-based approach, our framework separates the DCMC problem into two parts: core selection and tree construction. Our model, *SPAN*, first selects the cores to lead to a set of shared-trees for efficient multipoint communication. The tree-construction module then takes over for building the ultimate multipoint communication path. Our motivation is not only reducing the complexity of the solution into two different processes, but also splitting the complexity of management of the resulting paths into core-dominated tree portions by delegating this task to the cores from the sources in one process rather than multiple processes across sources (see Figure 4):

1) *Core selection*: the selection of the core set for the multipoint communication group. The intuition is separating the receiver set into certain groups with respect to delay distances from sources to meet the delay constraint of the application, and building the sub-optimized paths separately for each of these receiver groups. The core selection process identifies certain cores in the domain each dominating a particular receiver group "locally" within the delay bounds.
2) *Tree construction*: the construction of core trees rooted at the cores to span the subsets of the receiver groups dominated by the cores, and the construction of source trees to span the core set. We relax the restriction for a unique delay-bound to be met across the tree and define this bound separately for each receiver to be spanned on a core-tree, and for each core to be spanned on a source-tree. In an attempt to maximize the links shared by multiple receivers on the resulting paths, we set the delay-bounds to be met on the source-trees to the minimum possible, and with this, leave maximum delay-residue for the delay-bound on the receiver trees.

We define the *leaf* set, i.e., the set of nodes to be spanned by a core tree as the subset of receivers where for a source tree the domain of leaf set is restricted to the set of the cores serving the group. A core tree spans all the receivers it is dominating for at least one source it serves. The entire set of cores serving a particular source *s* is spanned on the source tree rooted at *s*. In the resulting trees, the delay-bound of each leaf as of its distance from the tree root is met. Core selection and tree generation processes operate modularly with tree generation algorithms processing on the result of core selection to develop delay-constrained trees on the set of nodes fed them as input.

Note that the higher the delay-bound between the tree-root and the nodes to be spanned, the higher the potential efficiency of the tree structures to be formed in terms of the paths shared by multiple destinations and the cost-efficiency of the paths. In our architecture, when we select a core, we assign to it its maximal domination set for its full exploitation in the topological region it dominates. The set of cores serving the distinct receiver-set partitions for that source tend to be located in distant sub-domains from one another. So the source-to-core paths on a particular source-tree rarely intersect. By the same argument, there is potential improvement on the structure of a core-tree to which wider range of alternative paths are available to span a set of nodes relatively local to one another. As for the cost of the tree, the stream from a source to be delivered throughout a core-tree towards potentially multiple receivers travels the "longer" paths on the core-tree than on the single path between the source and the core. Therefore, we favor the delay-distances on core-trees to those on source-trees and leave the entire delay-residue for core-trees.

Cost-efficient multipoint paths is a major goal of our design. In addition, we consider the path maintenance overhead and potentially dynamic memberships. We, therefore, look for a solution that minimizes the path structure in terms of the number of links on the multipoint path, and attempt to maximize the use of a core tree by multiple sources. Note that the problem of link minimization has no known polynomial-time solutions since SMT operating on unit link costs, as we noted in Section 2.1, is NP-complete and reduces to it.

4.2 Core Selection

A domain node is a candidate core only if it connects at least one source-receiver pair within the delay-bound of the application. *SPAN* initially identifies the potential cores into a pool as candidates by examining their domination characteristics: each candidate core tests itself on the local state information for its domination for the group, and reports its results to a designated coordinator node in the domain which selects of cores based on these results. The resulting core set selected among the candidate cores includes the cores that lead to a multipoint path to approximate the optimum solution.

Let $T_{c,s}$ be a core-tree rooted at core c to serve source s for all receivers in $D(c,s)$. According to this definition, $T_{c,s}$ "totally" serves s, its *defining source*. Let $T_{c,s}$ serve a source $s' \in S$, so that $D(c,s) \cap D(c,s') \neq \varnothing$ and $D(c,s) \neq D(c,s')$. We say, in this case, that $T_{c,s}$ serves s' partially. In an attempt to minimize the multipoint path structure in terms of the number of links, our approach is the maximization of the combined partial and total utilization of multipoint paths for simplification of ultimate trees in conjunction with efficiency of the transmission. Our architecture requires that a tree $T_{c,s}$ constructed to serve its defining source s for all the receivers in $D(c,s)$ also serves, with no modification on the on-tree paths, the rest of the sources for all receivers in set $D(c,s) \cap D(c,s')$ $\forall s' \in S$. For all $s' \in S$, a domination $r_{s'} \in T_{c,s}$ is served on $T_{c,s}$ along the path that serves r_s. According to this, $r_{s'} \in T_{c,s}$ implies $r_s \in T_{c,s}$. Furthermore, $r_{s'} \in T_{c,s} \Leftrightarrow r \in D(c,s) \cap D(c,s')$.

Consider the definition of *domination count* of a potential core-tree $T_{c,s}$ as follows:

$$\sum_{s' \in S} | D(c,s) \cap D(c,s') |$$

Literally, domination count of a (c,s) tuple specifies the number of source-receiver pairs the core tree $T_{c,s}$ is capable of serving when the receiver set being served is restricted to $D(c,s)$. The extended solution space offers, during core selection, efficient choice of a core in consideration of each source separately, diverting from the shared utilization of the trees for the efficiency of path construction, update and management during the communication session. However, higher utilization of $T_{c,s}$ in terms of the number of sources it is partially or totally serving improves the overall multipoint path structure through tree sharing as the case in core-based approach literature [B97, B97b, DEFJ94, CZD95]. Furthermore, higher utilization of shared trees in terms of the number of receivers each is dominating improves the efficiency of transmission as the case in multicast routing literature (see [B97b, DEFJ94, M94b, WPD88] among others). Therefore, high domination count is desirable as a core-selection criterion.

Figure 5 presents *SELECT*, the core selection component of *SPAN*. Let Δ be the delay bound of the application. We denote by $\Delta_{i,j}$ the maximum allowed delay between the nodes i and j to meet Δ on the path to be constructed. L indicates the leaf set, i.e., the set of nodes to be spanned on the specified tree. *SELECT* processes at the coordinator on the candidate cores' domination information reported to it. The algorithm, aiming to maximize the already constructed core-trees by sources in the group, iterates to search the non-singular as well as the singular solution space for the core with the highest domination count. Following the initialization of the ultimate core and domination sets in line E1, *SELECT* starts out with the computation of the domination counts for each source-core pair (lines E2-6). The main loop in lines E8-35 iterates to select a core-tree $T_{core,source}$ with the highest domination count until the entire group is served by the selected trees. A candidate core being a source is a tie-breaker when multiple candidates return the same domination count. Initially in this loop, the core-source tuple satisfying this criterion is selected (lines E9-15). This selection determines a core tree in terms of its root, *core*, and its defining *source*. The inner loop in lines E21-34 performs the following:

a) modification of the source trees: *core* is a leaf of a source *s* if and only if $T_{core,source}$ is serving *s* for at least one receiver in its leaf set. Upon satisfactory test of this condition in line E24, *core* is added to the set of leaves on the source tree *s* (lines E25-26). As we noted earlier, we set the delay-bound between a source and the core on a source tree to the minimum possible to allow maximum delay residue on the core trees. With this, the delay-bound of core on the source tree *s* is $path_{delay}[s,core].delay$ as set in lines E22 and E27.
b) specification of $T_{core,source}$: the computation of the delay bounds of each leaf of the core tree $T_{core,source}$. The subset of receivers in $D_u(core,source)$ (line E19) is inherently the leaf set of $T_{core,source}$. Let *S'* the subset of sources that $T_{core,source}$ is serving totally or partially for a receiver *r*. Then, the maximum possible delay-bound of *r* on $T_{core,source}$ is $\Delta_{core,r} = \min_{s \in S'} \{\Delta - \Delta_{s,core}\}$ as computed in lines E23 and 28.
c) update of domination status: the recently dominated r_s are excluded from the domination sets of the current candidate cores, and the domination counts are updated accordingly (lines E29-32, E36) within further iterations for core selection.

As mentioned earlier, the construction of the core and source trees follow the core selection on the results of the core selection. We model the tree development processes as coordinated by the tree roots, namely the selected cores and the multipoint sources. Once we select the cores, we notify the cores instantly of their leaf sets and the delay-bound of the leaves (line E35), and the sources of the relevant information (lines E38-39).

4.3 Tree Construction

In this section, we present the tree construction component of *SPAN*. The required input to a constrained-tree construction algorithm is the root, leaves, i.e., the set of nodes to be spanned on the tree, and the delay-bounds to be met for each leaf. To achieve efficiency in the resulting tree, we relax the condition of a uniform delay-bound and allow the delay-bound to vary across the nodes to be spanned on the tree. *SELECT* generates as part of its processing the overall input for the tree construction algorithm as we specified in the previous subsection. Before we introduce our tree construction algorithm, we remark that a *successful* tree construction algorithm must satisfy the following conditions:

1.) *The resulting path has a tree structure:* every on-tree node except the root has exactly one parent, and root is the ancestor of every on-tree node.
2.) *The tree meets the delay-bound*: $delay_l \leq \Delta_l$ for each leaf *l*.

We use a modification of the incremental SMT heuristic proposed by [TM80] for constrained-tree construction due to its feasibility for distributed implementation and operability on local information of the domain nodes, with relatively low message exchange in asymmetric networks. The generic intuition is iterative addition of the off-tree leaf, which currently is at minimal distance to any node on the tree until all leaves are spanned. Consider the function $delay_i$ returning the delay-distance of an on-tree node i to the tree root via the path connecting it to the tree root. Let Δ_i represent the maximum allowed delay on a tree path connecting the node i to the root. With the modification of the distance-to-tree function for a given on-tree node i and an off-tree node j as

$$distance\_to\_tree[i,j] = \begin{cases} path_{<distance>}[i,j].<metric> & ;\text{ if } path_{<distance>}[i,j].delay + delay_i \leq \Delta_j \\ \infty & ;\text{ otherwise} \end{cases} \quad \text{(I)}$$

the heuristic generates a constrained-tree, which still adds the next “node-closest-to-tree”, but here also meeting the delay-bound of the application. In our case, we have the availability of the information on alternative paths connecting any pair of nodes and thus connecting a leaf to a particular on-tree node.

In Figure 6, we present the *CONSTRUCT_TREE* algorithm, the tree-construction component of *SPAN*. Depending on a parameter such as hop-count or cost to be optimized, the algorithm results in the distributed construction of a delay-bound tree approximating the optimal solution in the form of tree state information on the constituent routers. The variable <*distance*> is a “switch” to specify hop-count or cost as the metric being processed on for minimization. We use the modified-distance function (I) and choose the minimum delay-bound-distance path between the delay-minimum and <*distance*>-minimum paths connecting the nodes. The algorithm starts with the *root* as the tree (lines C3-4) and iterates to add one leaf node at-a-time to the tree until the entire leaf set is spanned (lines C5-23). At each iteration, *CONSTRUCT_TREE* processes to select the leaf currently off-tree and closest in <*distance*> to the tree. The algorithm maintains the following status information on the minimum delay-bound distance path connecting l to the tree separately for each un-spanned leaf l:

i.) min_dist_l: minimum of all delay-bound distances between an on-tree node and l,
ii.) $on\_tree\_node_l$: the on-tree node connecting l to the tree at cost min_dist_l,
iii.) $code_l$: <*distance*> or “delay” depending on the metric determining the minimum-distance path.

Note that, for a given $l \in L$, $on\_tree\_node_l$ and $code_l$ are sufficient to uniquely identify the minimum-distance path for l since the end-points of a shortest-distance path along with the metric of the distance are sufficient to identify a shortest-path.

The procedure *ADD_PATH* (Figure 7) triggers the on-tree “end-point” of the new path to be constructed and operates to traverse the nodes for the establishment of the path. During its operation, *ADD_PATH* traverses nodes to transmit the required information to trigger the process at the recipient node. We represent by the suffix “*_msg*” the message communicated between domain nodes. According to this, *ADD_PATH_msg* is a message generated by an instance of the *ADD_PATH* sent to the next node to execute the process. The generic notation $<value>_L$ denotes the set $\{<value>_l : l \in L\}$ where <*value*> is Δ to denote the maximum allowed delay distance of the given node from the root, or one of the array variable *min_dist* and *code* to denote the corresponding values across the members of the set L. Each node receiving *ADD_PATH_msg* locates through the unicast routing-table lookup its neighbor on the minimal <*code*>-distance path to the *leaf* (line A2), and updates its tree state-table with this neighbor as its downstream node on the tree (line A3) by adding *child*, as its immediate downstream node (line A3). The block in lines A1-6 terminates with relaying *ADD_PATH_msg* to child for the propagation of the path construction (line A5) upon updating the delay parameter (line A4).

The procedure *MIN_DIST* (see Figure 8) is the module computing the minimum delay-bound path between the node it processes on and the given *leaf* based on formula (I). Depending on the processing metric specified in <*distance*>, *MIN_DIST* chooses a connection to *leaf* via the minimum-<*distance*> path if it warrants the specified delay bound, Δ_{leaf} (lines M2-5), otherwise returns the delay-minimum path (lines M6-9). The value *code* returned by *MIN_COST* distinguishes between <*distance*> and delay – the metric used to select the connecting path (lines M4 & M9). *min_dist* specifies the minimum distance attained between the two nodes in terms of the operating metric.

PATH_INFO (Figure 9) is the process for the transmission of the state information on the added path to *root* for further processing of *COSTRUCT_TREE*. *PATH_INFO* is initiated by *leaf* upon successful arrival of *ADD_PATH_msg* and execution of *ADD_PATH* at this node. For each one of the remaining *leaves* $l \in L$, *leaf* computes the minimum delay-bound <*distance*> to *l* (lines P2-9) and relays the message to its parent. Each node receiving *PATH_INFO_msg* similarly computes its minimum distance to each node in *L* and updates the message accordingly. When *PATH_INFO_msg* arrives at *on_tree_node*, it is simply transmitted to *root*.

At each main iteration for the addition of a new leaf to the tree (lines C5-23), *CONSTRUCT_TREE* first initiates *ADD_PATH* for the establishment of the path to connect the leaf selected at the previous iteration (line C6). The algorithm then waits for *PATH_INFO_msg* (line C7) to update its tree-state information for its further processing. We impose a constant time-out period between issuing *ADD_PATH_msg* and the arrival of *PATH_INFO_msg* for protocol operations. When *PATH_INFO_msg* is received, *CONSTRUCT_TREE* updates the minimum possible <*distance*> connecting each one of the remaining leaves to the current tree (lines C8-13). The algorithm later iterates on the current leaf set to choose the leaf closest to the tree as the one to be added next (lines C15-21). The selected leaf is excluded from the set of off-tree leaves by updating *L* (line C22). We, therefore claim that *CONSTRUCT_TREE* is a successful tree construction algorithm.

Lemma 3. CONSTRUCT_TREE is a successful tree construction algorithm.
Proof: Throughout the execution of *CONSTRUCT_TREE*, the tree being formed maintains exactly one parent for each on-tree node other than the root. The path formed at each step connects a new leaf to a currently on-tree node, and the formation of the path maintains this node as the ancestor of all nodes processed. This process iterates on the root as the initial tree, ensuring every node processed further is a descendant of the root. The resulting set of paths is a unique tree on which the given root is the ultimate ancestor of all nodes processed. The resulting tree of *CONSTRUCT_TREE* satisfies condition (1).

At each iteration of its main loop, *CONSTRUCT_TREE* adds the leaf closest to the tree and the loop iterates as long as there still are leaves to be spanned. In any iteration *CONSTRUCT_TREE* does not add more than one leaf to the tree. This situation, i.e., the addition of multiple members of the leaf set to the tree within a single iteration of the algorithm, would occur if there is an un-spanned leaf as an intermediary node on the path being added. However, this would be a contradiction since, in this case, the intermediary leaf would be closer to the tree than the leaf at the end-point of the path being added. Therefore, at each one of its executions, the main loop in *CONSTRUCT_TREE* (lines C5-23) adds exactly one leaf to the tree. Observe that, whenever there is a solution meeting the delay bound, the root is always a potential connecting node for all leaves, and thus there always exists an on-tree connecting node. Also observe that, the formation of the new path between the connecting on-tree node and the leaf currently being added to the tree is the only process updating the tree structure. *CONSTRUCT_TREE* is designed to choose between two alternatives – delay-shortest and <*distance*>-shortest paths between the given pair of nodes. The new path formed to add a leaf l inherently considers and obeys Δ_l. The delay bounds for all leaves are met when they are initially added to the tree. The only possibility that a delay-bound can be violated is when an existing tree path is updated during the formation of a new one. This situation is specified in line A3 of *ADD_PATH*, which potentially updates the parent of an on-tree node, which is on another tree path. If this change occurs during the formation of a delay-shortest path, then all changes on the on-tree paths are for only the shorter-delay paths and the resulting tree of this step does not violate the delay-constraint of any one of its existing leaves. This leaves the only possibility for delay-violation be due to an on-tree node, say n, switching from its delay-shortest connection to <*distance*>-shortest connection to the root. For part of its upstream connection to be updated, node n has to be an intermediary on the path being formed. However, this occurs only when n is closer in its distance to the leaf being added than the on-tree end-point of the path newly being formed, which is a contradiction. Thus, the updates on existing tree paths are only for shorter-delay paths, and the ultimately formed tree meets the delay-bounds of each leaf it spans, satisfies condition (2). Therefore, *CONSTRUCT_TREE* is *successful* and meets its design specifications. □

4.4 Distributed Deployment

As we stated earlier in Section 4.1, our framework partitions the multipoint routing problem into core selection and tree construction processing sequentially as two modular components. The core selection component of our model compares the candidate cores against a selection criterion and chooses the highest-ranking candidate as the next core. We design the distributed deployment of our core selection as coordinated by some node in the domain. A *coordinator* node known throughout the domain acts as a "hub" for the accumulation of the relevant information, and the execution of the cross-comparison process on the candidates for the ultimate selection. The coordination of the entire process by a domain node is necessary for feasible deployment in distributed platforms especially in terms of the message exchange overhead. Such coordination has been even proposed for the single-core selection schemes based on cross-comparison (see [KH03] for a comparison of core-selection algorithms). In the generic sense, each candidate core tests itself against the selection criterion using the local information available to it and reports its result to the coordinator. The coordinator is also the node running the process to select the cores based on the information reported to it.

In our case, the core selection criterion is the domination set size of which the computation for an asymmetric deployment requires as input minimum delay-distances *from* each source *to* a particular candidate and the minimum delay-distances *from* the candidate separately *to* each receiver. A domain node c dominates a receiver r for a source s, and hence $r \in D(c,s)$ if and only if $path_{delay}[s,c].delay + path_{delay}[c,r].delay \leq \Delta$. Assuming the availability of the information on full source and receiver sets only to the coordinator, our model relies on a source-initiated message broadcast throughout the network for the discovery of nodes that serve the particular source for a non-empty subset of receivers within the delay bound. The coordinator initiates the entire process by notifying each source of the receiver set and the delay bound of the application. This message triggers on its recipient source, say s, the generation of message $SUBDOMAIN_s$ separately for each outgoing link of s. Along with the receiver-set and the delay-bound information, the message $SUBDOMAIN_s$ carries a time-stamp to keep track of the delay-distance it travels. Source s sets the value on the time-stamp to the delay distance between itself and an immediate neighbor before sending the packet on the outgoing link to that neighbor. Figure 10 shows the process carried out by a recipient of the message

SUBDOMAIN$_s$. Observe that *SUBDOMAIN*$_s$, which is flooded throughout the domain, reaches a particular domain node firstly through the delay-shortest path connecting *s* to it. Thus, the first *SUBDOMAIN*$_s$ message reaching a domain node *c* communicates to *c* all required information particular to *s*, including $path_{delay}[s,c].delay$. Further copies of this message received from the same source are redundant and hence are simply discarded (see line S1 in Figure 10). If *SUBDOMAIN*$_s$ is the first to be processed on its recipient node, then the delay-distances are examined. If the delay-distance the packet traveled is already beyond the delay-bound, the packet is discarded (line S2) since it violates the delay-bound of the application on its delay-shortest path. If the time-stamp on the packet is still less than the delay bound, the domination set of candidate core *c* for source *s* is computed for potential receivers in the set (lines S3-4). If the domination set is empty, *SUBDOMAIN*$_s$ is still discarded (line S5) since any further path traveled by this packet can not lead to a candidate core serving source *s* for a receiver in the group. The domination set, $D(c,s)$ being non-empty (line S6) indicates node *c* as a candidate core. In this case, *c* reports $D(c,s)$, $path_{delay}[s,c].delay$, and $path_{delay}[c,r].delay\ \forall r\in D(c,s)$ to the coordinator (line S7), and relays the message *SUBDOMAIN*$_s$ on each of its outgoing links except the one it received the packet after incrementing the time-stamp of the packet with the delay-distance of the destination neighbor from itself (line S8-11). This process transmits the domination sets of all potential cores to the coordinator, which is necessary and sufficient for the entire execution of the core selection algorithm on the coordinator. The directed delay-distances between the source-core and core-receiver pairs are also transmitted for the computation of the delay-bounds of the leaves on each core and source tree.

The coordinator then makes the core selection resulting in the optimal domination sets, which specify the source and core trees as well as the leaves to be spanned by each tree and the delay-bound of each leaf. Afterwards, each tree root is notified of the corresponding leaf-set. Our architecture models the trees developed and maintained independently from each other, and the entire process is coordinated at the tree root where it is initiated. Distributed tree deployment requires that every root maintains the state information *CONSTRUCT_TREE* (Figure 6), which describes and $delay_n$ at node *n* to each on-tree node *n*. The *CONSTRUCT_TREE* algorithm runs at the tree root for *i*.) the selection of the un-spanned leaf closest to the tree in the current state (lines C14-21), *ii*.) the coordination of the tree update to add the newly selected leaf to the tree (line 6), and *iii*.) the update of its state information with the current form of the tree (lines C7-13). The selection of a new leaf is the comparison of the distances of each leaf to the tree and relies on the state information maintained at the root.

When a new leaf and its connecting path are determined, the root notifies the appropriate *on_tree_node* using *ADD_PATH_msg,* which specifies the *leaf* being added, the *code* that specifies the metric of the minimum-distance path, the path delay information and the current set of off-tree leaves. *ADD_PATH_msg* travels along $path_{code}[on\_tree\_node, leaf]$ carrying a time-stamp initiated to $delay_{on\_tree\_node}$. As it traverses each next-hop neighbor on the path, the processing node *n* increments the value on this stamp with $delay[n,m].delay$ where *m* is the next recipient of *ADD_PATH_msg*. When *ADD_PATH_msg* reaches a *leaf* and thus the path formation is complete, *PATH_ACK*, an acknowledgement, is initiated and travels on the reverse path, examining the minimum tree-distances computed for each leaf on the nodes it traverses and selecting the minimum of these values separately for each leaf to transmit the minimum distance of each remaining leaf to the newly updated tree portion. Once *PATH_ACK* reaches the *on_tree_node*, the results are forwarded to the root where the state information regarding the tree-to-leaf distances are updated with the incoming information. Throughout its operations, *SPAN* considers the link utilizations in both directions during the group communication session, and thus *SPAN* is an asymmetric framework.

4.5 Complexity Analysis

In this section, we analyze the complexity of *SPAN*. We first revisit its architectural structure for the organization of the components. Then we examine the computational-time complexities of the processes along with the amount of message transmission. We assume $O(1)$ time for unicast and tree multipoint table state look-up, table update, and the search and update operations on the sets. Utilization of direct addressing on array structures justifies this assumption feasibly considering the size of potential value domains of the variables to be represented.

Considering *SPAN* as a protocol unit, its operation is triggered by a multipoint group information transmitted to coordinator, and terminated upon the construction of the source and core trees. *SPAN* follows the following sequence of steps in the given order:

1) Coordinator triggers sources in the group: the coordinator communicates to sources the group information, S and R, and application delay bound, Δ.
2) Sources initiate *SUBDOMAIN*: each source s in S initiates $SUBDOMAIN_s$ as described in Section 4.4.
3) Reports from candidate cores transmitted to *coordinator*: upon receiving $SUBDOMAIN_s$ each qualifying candidate c reports to *coordinator* its domination set $D(c,s)$ and relating delay information as specified in line S7.
4) *SELECT* executes at coordinator to generate the core set, determine the leaf sets and leaf delay-bounds of each core and source tree, and communicate this information to the sources and the cores for tree construction.
5) Coordinator triggers sources and cores for tree construction: for each tree to be constructed, the root is communicated as part of the processing of *SELECT* (lines E35, E38-39) the set of leaves and delay-bounds of each leaf. Note here that, since cores are notified during the execution lifetime of *SELECT*, the execution lifetime of *CONSTRUCT_TREE* processing on cores to generate the core trees overlap with that of *SELECT*.
6) *CONSTRUCT_TREE* iterates at each source and core for the following additional to its local execution:
    a. Triggers the connecting *on_tree_node*
    b. *ADD_PATH* propagates for the construction of the path to add the selected leaf to the tree
    c. *PATH_INFO* propagates back to the root for the transmission of the information on the recently added path

The group information from the coordinator to sources transmits on $\Theta(|S|)$ messages in step (1). $SUBDOMAIN_s$ is processed to iterate on the receivers set (lines S3 and S7) and the degree of the processing node (line S8), and executes in time $O(|R|+e)$ where e is the maximum node degree in the domain. In the worst case, the delay bound of the application is large enough not to be binding, so that any domain node is a candidate core to serve all sources in the group. In this case, each recipient of a $SUBDOMAIN_s$ is relaying it to each one of its outgoing links if and only if it is the first copy from source s within the delay bound. Then, for a given source s, $O(e|N|)$ links are traversed by $SUBDOMAIN_s$ in steps (2) and (3), and the total number of links exploited by these messages across all sources is $O(e|N||S|)$ where N is the domain size. A recipient n of $SUBDOMAIN_s$ reports to the coordinator its delay-distance from source s and the delay-distances separately to each receiver it is dominating for source s if and only if $D(n,s)$ is non-empty (line S7). The number of candidate core reports as a results of this stage of operations is proportional to the number of domain nodes and the size of the source set. Thus, the transmission of candidate core results on core-selection criteria to the coordinator for this stage amounts to $O(|N||S|)$.

Since line E5 of *SELECT* compares the members of two sets of both of which the domain set for the elements is R, it can be implemented in time $O(|R|)$. The loop executing this statement in line E4 iterates on set S, thus executes in time $O(|S||R|)$. The main loop nesting this statement in line E2 iterates on the elements of C and S. The overall complexity of *SELECT* in lines E2-6 is $O(|S|^2|R||C|)$. The core selection criterion used by *SELECT* is the domination count of each candidate core. In each iteration, the core dominating the maximum number of receivers across all sources is chosen from the pool of the candidates. Observe that, whenever there is a delay-bound solution, the minimal set of candidates contains each source serving itself for all receivers in the group. Therefore, the size of the ultimate core set is bound by the number of sources in the group, i.e., $|C_u| \leq |S|$. The repeat loop in lines E8-37 processes to select exactly one core-tree at each one of its iterations, and terminates when all r_s $\forall$ $r \in R$, $s \in S$ are served by the selected core trees. With this, the repeat loop iterates as many times as $\Theta(|C_u|) = O(|S|)$. The inner loop in lines E10-15 iterates on sets C and S, and the body of his loop executes in time bound by a constant. The execution time of this loop is $\Theta(|S||C|)$. The maximum number of ultimate domination sets produced for one core is at most as many as $|S|$. The size of a domination set, on the other hand, is bound by $|R|$. With this, line E35 executes in time $O(|S|+|R|) = O(|R|)$. The inner loop of repeat in lines E21-34 is nesting another loop in lines E29-32 which executes in time $O(1)$ for each member of C. The outer loop itself processes on sets S and $D_u(core,source) \subseteq R$, with the rest of its body executing in time $O(1)$. The overall complexity of the block in lines E21-34 is $O(|S||R||C|)$, and has a higher bound than that of the rest of the block nested by the repeat loop, making the repeat loop execute in time $O(|S|^2|R||C|)$. The rest of the algorithm that involves the notification of the roots of the source trees to be constructed in lines E38-39 has the complexity $O(|S|^2)$ and not binding (steps 4 and 5). The overall complexity of *SELECT* is $O(|S|^2|R||C|)$, which is the minimum attainable complexity of a core selection algorithm processing in the singular and non-singular solution spaces [KH03].

The coordinator notifies the sources and the cores for the construction of corresponding trees in $|S|+|C_u| \leq 2|S|$ messages (lines E35, E38-39). At its initialization step in lines C1-4, *CONSTRUCT_TREE* executes in time $\Theta(|L|)$ due to the loop in line C1. The main loop of the algorithm in lines C5-23 iterates as many times as the size of the leaf set. Line C6 executes in time $O(1)$. On assumption of a time-out period for the arrival of *PATH_INFO_msg* in protocol operations, the delay between the transmission of the messages in line C6 and the arrival of reports from these nodes in line C7 is bound by a constant. Upon receiving *PATH_INFO_msg*, the loop in lines C8-13 executes in time $O(1)$ and thus its computational complexity is $O(|L|)$. The for loop in lines C15-21, having a similar structure, also takes $O(|L|)$ time. Therefore, the while loop executes in time $O(|L|^2)$ which is the complexity of *CONSTRUCT_TREE*. The leaf set is a subset of R and $C_u \subseteq S$ in the formation of respectively the core and source trees. Thus, the computational complexity of *CONSTRUCT_TREE* is respectively $O(|R|^2)$ and $O(|C|^2)$ for the construction of core and source trees (step 6).

The execution of *ADD_PATH* in step 6-a, mainly for the state look-up and update of the unicast and multicast routing tables is bound by a constant. *ADD_PATH* runs on $O(a)$ nodes for each leaf being added where a is the node-diameter of the domain, and the total number of its executions in different processes during the lifetime of *CONSTRUCT_TREE* is $O(a|L|)$.

MIN_DIST invoked by *PATH_INFO* is mainly a table look-up function and with the implementation on direct addressing as we mentioned, executes in time $O(1)$. *PATH_INFO* iterates solely on the leaf set (line P2) for comparison and update of the minimum distance-to-tree information for each element of this set. The complexity of *PATH_INFO* is $O(|L|)$. *PATH_INFO*, being processed exactly by those nodes which processed *ADD_PATH* in reverse order, is executed as many times as *ADD_PATH*. The time complexity of the *SELECT*, processing for the core selection is highest among the processes in *SPAN* and thus makes the time complexity of the overall system as $O(|S|^2|R||C|)$. The *SUBDOMAIN* messages used to transmit the required domain information to the coordinator dominate the amount of message transmission for *SPAN*. This amount warrants feasible deployment in distributed platform, and *SPAN* carries out its execution on local unicast routing states with no reliance on an external data gathering protocol.

5. Performance Evaluation

The optimum solution to minimize the cost of the resulting multipoint path to serve the group within the delay bound of the application is the set of constrained-SMTs, each rooted at one source, and spanning the entire set of receivers in the group. As we pointed out in Section 2.1, this solution is NP-complete.

Existing solutions for group communications are mostly restricted to single source and/or unconstrained cases and the problem is considered separately for core selection and tree construction rather than within the context of a comprehensive model. As we stated in Section 2.2, the prominent solution in literature to the DCMC problem is *GREEDY* [Sa96]. In its architectural description, *GREEDY* operates on bi-directional trees so that every on-tree node on a given tree forwards every incoming data packet to everyone of its on-tree links except the one where it received the packet from regardless of whether the outgoing link(s) lead to receivers as further destinations. We modify *GREEDY*, into *m-GREEDY*, which, in account of the bi-directional utilization of the generated trees, traverses the tree for the source of the current data stream assumed as the root, and forwards the stream *only* to those links leading to downstream receivers in the traversed version. Observe that our modification performs at least as good as *GREEDY* itself at the cost of tree maintenance overhead additional to that of *GREEDY*. We also consider *m-GREEDY* as a potential alternative to *SPAN* for its cost-efficiency compared to its original version.

For a direct comparison of our model to its distributed counterparts in the singular solution space, we generated a model, *SINGULAR*, which applies the entire architecture of *SPAN* this time to process in the singular solution space. *SINGULAR* differs from *SPAN* in its domination count which now is an attribute of a core rather than a core-source pair as the attribute uniquely describing a core-tree, i.e., $r_s \in T_{c,s} \Leftrightarrow r \in D_u(c,S) \quad \forall r \in R,\ s \in S,\ c \in C_u$. According to this, the domination specifies the number of receivers dominated by the core for all sources in the group. The candidate core c returning the highest value for $|D(c,S)|$ as the primary criterion and c being a source or closest in average <*distance*> to source set as the secondary criterion is selected to be the next member of the core set.

We identify two performance measures for the models tested:

a) *Cost*: total transmission cost. The transmission cost for a particular source is the sum of the cost of the links traversed during the transmission of one packet from that source to the entire receiver set. Overall transmission cost is the sum of the transmission costs of all trees.
b) *Tree structure*: the total number of links on the resulting multipoint path. Each of the scenarios we examine describes the core-trees and source-trees as administered for their modification and maintenance independently from one another. We count the number of links as distinct in utilization and maintenance. The overall link-count of the entire multipoint path in turn is the sum of the link-counts of the individual trees with no regard of overlapping links across distinct trees.

Note that the measure *tree structure* differs from cost in that it does not consider the links on a particular core tree shared by multiple sources served on that core tree, whereas *cost* considers the cost of each such link is cumulated across the sources using them for the delivery of the stream. The computations of cost and tree structures are equivalent only if $|S|=1$.

In order to maintain the same "scale" for the symmetric (*GREEDY* and *m-GREEDY*) and asymmetric (*SPAN* and *SINGULAR*) models being evaluated, we avoided asymmetric domains and tested our samples on symmetric networks in which the link costs and delays are equal in both directions of the link. As mentioned earlier, *SPAN* and *SINGULAR* process on asymmetric domains, whereas *GREEDY* is necessarily symmetric. Our results compare *GREEDY* and *m-GREEDY* to their asymmetric alternatives within their intended design setup with no restriction on the functionality of the asymmetric models.

A primary consideration in our performance of delay-constrained problems is what we call the *critical delta,* $\Delta_{critical}$ values. These represent the delay bound of the application for each domain where critical delta is $\max_{s \in S, r \in R} \{path_{delay}[s,r].delay\}$, namely the minimum delay bound that leads to a successful solution on the problem instance. Consider also the definition of *maximum delta,* Δ_{max}, which is $\max_{s \in S, r \in R, c \in C} \{\ path_{delay}[s,c].delay + path_{delay}[c,r].delay\ \}$. Maximum delta specifies the "boundary" where the results to be returned by any of the algorithms are no longer affected by the delay-bound parameter. In other words, maximum-delta is the upper-bound for the delay-bound range, beyond which any algorithm would be feasible. The range $[\Delta_{critical}, \Delta_{max}]$ specifies, for a problem instance on a given model the minimal delay-bound range of all possible solutions.

We also normalized the cost and link-count results of each algorithm against the results of the "reference" model. According to this, the cost and link-count results of each of the other models are divided by the corresponding outcome obtained from the reference under the same measurement setup, and the indicated results on cost and link-count are relative performances to those of the reference. The reference model is *SPAN* in all cases except in the comparison to the optimal solution (Figure 11), in which case it is the optimal solution itself that is used as reference. The performances of the reference models are considered unity and not explicitly depicted in our figures.

The core selection components of *SPAN* and *SINGULAR* within our framework process on the domination count of the candidates and thus only rely on the delay-distances between the node pairs, leaving the consideration of <*distance*> as the optimization metric to the tree development phase. Regarding the alternate design goals we set on transmission cost and link-count, we obtained for each case on sparsely distributed groups two versions of results. We first run the core selection component of *SPAN* and *SINGULAR*, this produces the domination sets, on which we run *CONSTRUCT_TREE* independently on cost and hop-count. By also using the transmission cost or hop-count as the performance metric, we measured the multipoint paths generated by both on a particular sample for each one of these performance metrics. Our figures combine the two parameters on the same metric on two different outcomes over the same set of samples. In each one of the graphs, the legends with solid lines indicate the results where the path generation metric and the performance metric are the same. The "dual" of a given evaluation is the one in which the path generation metric is different than the one being measured, and is indicated by the same legend with suffix (D) and represented by a dashed line. We remark that none of the models outperformed their duals compared to *SPAN*. This indicates *SPAN*'s effectiveness in achieving a target optimization goal. In both metrics, the models followed the same pattern as their dual measurements, which is expected, and demonstrates the consistency in the tree development component of our model.

Except for the performance comparison with the optimum solution, we used sample domains of size 60, and tested groups sparsely distributed throughout the domain. Our domains have the average node degree in range (3.5, 5), and the average of average of node degrees of our sample domains is 4.46. We used Waxman's model [W88] for sampling the domains. In all cases, the sources and receivers in the group are randomly distributed in the domain. We maintained a 90% confidence level with 10% confidence intervals in our measurements.

In the first scenario, we compare the cost performance of all algorithms to optimum solution in domains of size 10. The control variable is the group size which varies in range 5..9. Each of the sample groups contains exactly 3 sources one of which is also a receiver. The delay-bound is $\Delta_{critical}$ in each case. Note that the results on performance comparison to the optimal solution indicate solely the relative behavior of the models compared. It is not intended to reflect the characterization on the operational setting of the core-based models, which is sparse mode. This group of results is obtained on necessarily small domains due to the computation-time limitations of the optimum algorithm, and on samples densely populated in the domain to reflect the multi-source multipoint group communication characteristics.

Figure 11 shows our comparative results to the optimum solution. *SPAN* performed within the range of 120-140% of the optimum solution and outperformed the other models. *GREEDY*'s performance improved as the group size increased and thus the resulting trees are "saturated", i.e., spanned a greater portion of the domain so that the on-tree links used by the "source-tree" and "core-tree" components of the paths overlapped in dense mode. *SPAN*'s relatively consistent performance in this scale indicates its efficient path construction to serve its design purpose.

The remaining figures depict the results of our evaluation on domains of sizes 60 for an accurate approximation of the performances of the models in their operational environments. Figures 12-14 test groups in which some of the sources are also receivers and thus the source and receiver sets are not exclusive. In Figure 12, we tested each model on varying source-to-receiver ratios. The receiver set size is fixed at 16, the varying ratio is achieved by $|S|$ ranging in [6, 8, 10, 12, 14, 16]. The x-axis in each figure indicates the source-set size. Figures 12(a) and 12(b) present the cases in which respectively half and all of the sources are also receivers. Figure 13 presents the performance of the models on overall group size as the control parameter. The group ratio, $|S|/|R|$ is fixed to 1/2. $|S|$ and $|R|$ each vary in fixed range of corresponding values of [2, 4, 6, 8, 10, 12] and [4, 8, 12, 16, 20, 24], respectively. The source and receiver sets overlap: half and all of the sources are also receivers in the samples of figures 13(a) and 13(b). Figure 14 demonstrates the effect of group overlap $|S \cap R|$ on performance of the different models, where all sources are also receivers. In Figures 14(a) and 14(b) the group size is respectively 12 and 24. The size of the source set and thus $|S \cap R|$ is varying as indicated in x-axis in each figure.

In Figure 15, we tested the models for their performance on varying group sizes when $S \cap R = \emptyset$. The group ratio is fixed at ½. $|S|$ and $|R|$ vary in respective ranges [2, 3, 4, 5, 6, 7] and [4, 6, 8, 10, 12, 14]. In Figure 15(a), the delay bound is set to $\Delta_{critical}$ whereas in Figure 15(b) it is set at the midpoint of the delay-bound range, i.e., $\Delta = (\Delta_{max} + \Delta_{critical})/2$. Figure 16 compares the effect of the delay-bound on the performance of the models. Δ varies in range $\Delta_{critical} + kV$ where V is the "length" of the minimal delay-bound range, i.e., $V = \Delta_{max} - \Delta_{critical}$, and $k \in [0, 0.2, 0.4, 0.6, 0.8, 1.0]$. In Figure 16(a), $|S|=4$, $|R|=16$ and

S∩*R*=∅, whereas in Figure 16(b), |*S*|=8, |*R*|=16 and *S*⊂*R*=∅, the group size, |*S*∪*R*|, being 20 in both cases.

A prominent observation is that *SPAN,* processing on a broader solution space, consistently outperformed its counterparts in its cost performance, indicating that *SPAN* is the most efficient model for multipoint routing for constrained group applications. *SPAN* is outperformed in this metric only by *SINGULAR*—its counterpart in our framework executing in the singular solution space, and only when the models processed for link-count optimization rather than cost. *SPAN* maintained its high cost-performance across all models at the critical delay bound values, justifying its primary design objective to serve group applications bearing delay constraints. *m-GREEDY*'s higher performance over *SPAN* on groups of extremely small sizes (Figure 13.1) is exceptional and insignificant since the groups of this characteristics fall beyond the targeted effectiveness of core-based architectures. A relatively consistent result of our evaluation is the diminishing cost performance of the models tested against *SPAN* for larger group sizes at critical delay-bound values, reflecting the comparative efficiency of our architecture in path exploitation in constrained cases.

Another significant finding of our study is the considerably poor performance of *GREEDY* in comparison to its counterparts. Our figures on cost performances depict *GREEDY* at its extreme end of the scale, leaving the relatively close performance of the remaining models in their "magnified" range for a sophisticated comparison. *GREEDY* does not consider core-trees and source-trees separately and constructs, for each receiver partition, a tree rooted at the core spanning also all the sources in the group, explaining its higher performance for the resulting tree structure compared to its alternatives. The resulting tree combines the core trees and source trees from our analysis perspective of the solution space, and the data stream from a particular source is redundantly delivered to the other sources as well as to the receivers, adding on the delivery cost. However, the transmission-time pruning of *GREEDY*'s composite trees in its modified version results in the efficient use of the trees for the "short-cuts" they provide between the on-tree source-receiver pairs, and *m-GREEDY* highly outperforms its original version.

SPAN's cost performance advantage becomes more aparent as the source and receiver sets overlapped. The model performed even better at larger group sizes under this setup. This implies that *SPAN* is particularly preferable for groups that contain sources of data streams which are also receivers of other streams. A typical example for such a case is video conferencing.

Our empirical results point to a correlation between the cost and hop-count performances of *m-GREEDY* and *SINGULAR*. That is, *GREEDY* outperformed *SINGULAR* in one of these metrics whenever it outperformed *SINGULAR* in the other metric. Furthermore, their performances followed a similar pattern of one another. This result is not surprising since the total the total link count on the collective set of trees, although is not a direct measure of the number of times the links are exploited during the transmission, is indicative on the cost of exploitation of the links. Note here that the cost underperformance of *GREEDY* does not violate this observation due to its very high amount of redundant transmission. This result is suggestive of a generalized correlation between the cost and tree structure performances of the models operating on the same solution domain, which in turn potentially broadens the range of heuristics to solve the problem of constrained cost minimization.

SINGULAR's performance is close to *SPAN*'s on groups with smaller number of source participants, and closest when the group ratio is low. Such a result is justified, since the non-singular solution space provides alternatives across multiple sources in the group, and the provisions of this space for efficient solutions compared to singular space diminishes for reduced number of sources.

The difference in the performances of *SPAN* and *SINGULAR* attributes solely to their respective core selection algorithms since both models use the exact same tree construction module. Thus, *SPAN*'s higher cost-performance is a direct verification of our findings in Section 3 that non-singular solution space offers potential improvement on the efficiency of the solutions. Note that *SINGULAR*'s performance converged to *SPAN*'s as the delay-bound is increased. The core selection algorithms of both models operate on domination counts, which is an attribute of the candidate cores' domination sets restricted by the delay-bound of the application. The difference in the two algorithms is eminent at tight delay-bounds. As the delay bound of the application gets larger and less binding on the range of solutions, both *SPAN* and *SINGULAR* tend to select cores among the candidates with large domination sets. At the extreme end when $\Delta=\Delta_{max}$ thus the delay constraint does not apply, every node in the domain is a candidate core with full domination for all sources, and *SPAN* and *SINGULAR* have exactly the same set of candidate pool to select from. The difference in their performance at $\Delta=\Delta_{max}$ (Figure 16) is therefore solely due to the random ordering of the candidates in the pool and the secondary criterion that followed the domination count each applied during the selection process.

SPAN's cost performance over *m-GREEDY* despite its low performance on hop-count compared to the well-structured trees of the *GREEDY* architecture further indicates the high potential gain in cost efficiency provided by the range of non-singular solutions. Pure *GREEDY*, on the other hand, exhibits very low cost-performance characteristics, which was the main reason we introduced its modified version *m-GREEDY*. Furthermore, neither *GREEDY* nor *m-GREEDY* operate on asymmetric networks. Both models require full domain information and are infeasible in distributed networks. *SPAN*'s feasibility in distributed deployment providing delegated tree management warrants the comparatively more complicated tree structure of this model especially for groups with dynamic membership characteristics thereby demanding on tree maintenance.

6. Conclusions

In this paper, we investigated the solutions for delay-constrained multi-source, multipoint communication groups applying the core-based architecture in sparse mode. Our analysis of the solution space for this range of problems indicated a broader range which is not explored by existing models. We identified the solution space as being composed of singular and non-singular spaces, with all existing schemes exploring only the singular solution space. In the singular space, each core, serves all sources in the group and uniquely defines a shared tree. In the non-singular space, a core is not necessarily serving all the sources and each receiver can be served by multiple cores varying across sources in the group. We presented *SPAN*, a distributed model processing in the non-singular space and extending the range of solutions searched by its precedents. Our model describes the first distributed, asymmetric framework on the literature to provide solutions for constrained, core-based multi-source communication groups. Our empirical results showed the high efficiency of our model compared to its counterparts, clearly demonstrating the significance of our theoretical findings indicating the potential contribution of models processing in the extended range of solutions.

SPAN is a basic model designed for processing in the non-singular and singular solution spaces at moderate computational complexity and communication among the nodes throughout its execution. During core selection, *SPAN* only uses the domination information of the candidate cores, and operates on local distance-vector information available at the routers. Our results are promising for more elaborate designs that are also based on the relative cost and hop-count distances between the candidate core and group member pairs in core selection phase in this class of models, which is an approach proved to be effective for cost efficiency [TR97].

While the placement of cores in the domain is crucial for the protocol operations, the candidate core pool to be considered for the selection process can alternatively be restricted to a certain subset of domain nodes with respect to their coverage of the topology and domination of the potential group members local to them. *SPAN* considers the candidate core set as an external input across its relevant message transmission and core selection component, and processes with no restrictions on its functionality on reduced sizes of potential cores.

Broadband group communication over the Internet is becoming ubiquitous over a wide range of services. These applications are usually delay-sensitive and demanding on network resources. The core-based architecture offers the significant advantage of partitioning the inter-domain route construction on QoS-demands of the applications into intra-domain problems by the placement of core nodes within the autonomous routing domains to coordinate transmission to receivers and/or to border routers for further transmission across ASs. *SPAN* presented in this paper operates on local distance-vector information available at the routers, with no modification on their functionality. Our models can further enhance the support of inter-domain groups for participants across ASs through the construction and management of the intra-domain routes coordinated by the efficient placement of cores in each of the domains.

The participants in a multipoint communication group often tend to join and leave at will during the communication session solely with notification. A promising future direction for QoS-support under membership dynamics is the investigation of the performance of multipoint path construction models in terms of path deterioration from the original structure, and path reformation through core migration [DZ96] on certain deterioration levels based on the devised metrics.

REFERENCES

[B97] Ballardie, A., *Core Based Trees (CBT version 2) Multicast Routing – Protocol Specification*, RFC 2198, September 1997.

[B97b] Ballardie, A., *Core Based Trees (CBT) Multicast Routing Architecture*, RFC 2201, September 1997.

[B57] Bellman, R., *On a Routing Problem*, Quarterly of Applied Mathematics, Princeton University Press, 1957.

[CZD95] Calvert. K. L., Zegura, E. W., Donahoo, M. L., *Core Selection Methods for Multicast Routing,* IEEE ICCCN, 1995.

[CHH97] Chung, S. J., Hong, S. P. and Huh H. S., "*A Fast Multicast Routing Algorithm for Delay-Sensitive Applications*, IEEE Globecom, November 1997.

[D59] Dijkstra, E., *A Note on Two Problems in Graphs*, Numerische Mathematik, Vol. 1, pp. 269-271, 1959.

[DEFJ94] Deering, S., Estrin, D., Farinacci, D., and Jacobson, V. *Protocol independent multicasting* (PIM), *dense mode protocol specification*. Technical report, IETF-IDMR, 1994

[DZ96] Donahoo, M. J., Zegura, E., W., *Core Migration for Dynamic Multicast Routing*, Proceedings of ICCCN, 1996.

[FF62] Ford, L. R., Fulkerson, D. R., *Flows in Networks*, Princeton University Press, 1962

[GJ00] Garey, M., Johnson, D., *Computers and Intractability*. W.H. Freeman, 2000.

[J98] Jia, X., *A Distributed Algorithm of Delay-Bounded Multicast Routing for Multimedia Applications in Wide Area Networks*, IEEE/ACM Transactions on Networking, Vol.6, No.6, December 1998, pp.828..836.

[KH04] Karaman, A., Hassanein, H. S., *Core-Selection Algorithms in Multicast Routing*, submitted for publication.

[KH03] Karaman, A., Hassanein, H. S., *Core-based Approach in Multicast Routing Protocols*, International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS), July 2003.

[K72] Karp, R., *Reducibility among Combinatorial Problems*, Complexity of Computer Computations, Editors: R. Miller and J. Thatcher, New York, Plenum Press, 1972, pp.85-103.

[KPP93] Kompella, V. P., Pasquale, J. C. and Polyzos, G. C., *Multicast Routing for Multimedia Communication*, IEEE/ACM Transactions on Networking, Vol.1, No.3, June 1993.

[KPP93b] Kompella, V.P., Pasquale, J.C. and Polyzos, G.C., *Two Distributed Algorithms for Multicasting Multimedia Information*, International Conference on Computer Communications and Networks, San Diego, CA, June 1993, pp. 343-349.

[M94] Moy, J., *Multicast Extensions to OSPF*, RFC 1584, March 1994.

[M94b] Moy, John, *Multicast Routing Extensions for OSPF*, Communications of the ACM, Vol.37, No.8, August 1994, pp.61-114.

[PZG98] Parsa, M., Zhu, Q. and Garcia-Luna-Acevez, J. J., *An Iterative Algorithm for Delay-Constrained Minimum-Cost Multicasting*, IEEE/ACM Transactions on Networking, Vol.6, No.4, August 1998, pp.461-473.

[Sa96] Salama, H. F., *Multicast Routing for Real-Time Communication on High-Speed Networks*, Ph.D. Thesis, Department of Electrical and Computer Engineering, North Caroline State University, 1996.

[SRV97] Salama, H.F., Reeves, D.S. and Viniotis, Y., *Evaluation of Multicast Routing Algorithms for Real-time Communications on High-speed Networks*, IEEE Journal of Selected Areas in Communication, Vo 15, April 1997, pp.332..345.

[SG97] Shields, C., Garcia-Luna-Acevez, J.J., *The Ordered Core Based Tree Protocol*, IEEE INFOCOM, Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies, 1997.

[TM80] Takahashi, H. and Matsuyama, A., *An Approximate Solution for The Steiner Problem in Graphs*, Math. Jap., 24, 1980.

[TR97] Thaler, D. G. and Ravishankar, C.V., *Distributed Center-Location Algorithms*, IEEE Journal on Selected Areas in Communication, 15(3), 1997, pp. 291-303.

[WPD88] Waitzman, D., Partridge, C., & Deering, S., *Distance Vector Multicast Routing Protocol*, RFC1075, November 1988,

[WC95] Wang, Z., Crowcroft, J., *Bandwidth-Delay Based Routing Algorithms*, IEEE Globecom, November 1995.

[WC96] Wang, Z., Crowcroft, J., *Quality of Service Routing for Supporting Multimedia Applications*, IEEE Journal of Selected Areas in Communications, 1996.

[W88] Waxman, B., M., Routing of Multipoint Connections, IEEE Journal of Selected Areas in Communication, 1988, pp.1617-1622.

[WE94] Wei, L. and Estrin, D., *The Trade-offs of Multicast Trees and Algorithms*, Proceedings of 1994 International Conference on Computer Communications Networks, September 1994.

[W87] Winter, P., *Steiner Problem in Networks: A Survey*, Networks, Vol.17, 1987, pp.129-167.

[ZFL02] Zappala, D., Fabbri, A. and Lo, V., An *Evaluation of Multicast Trees with Multiple Active Cores*, Journal of Telecommunication Systems, Kluwer, March 2002.

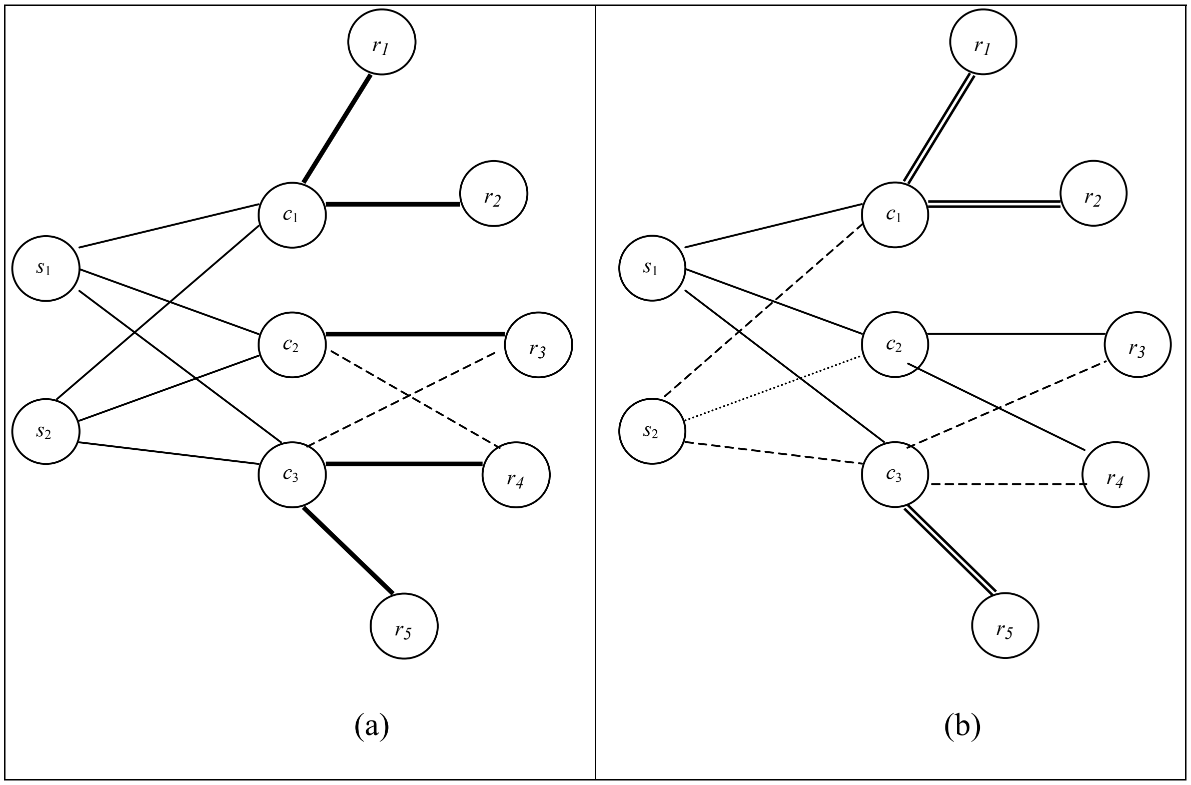


Figure 1. An example DCMC problem. a) solution involving a single multipoint tree serving the entire source set for all receivers, b.) an alternate solution involving two different multipoint trees to serve the group.

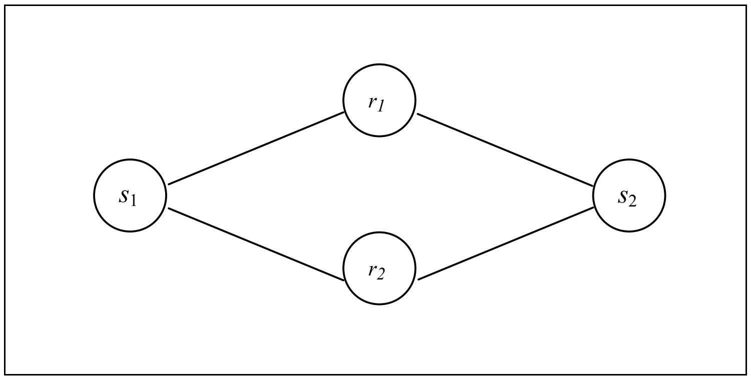


Figure 2. A multipoint communication group with 2 sources. Each source is also a core serving itself.

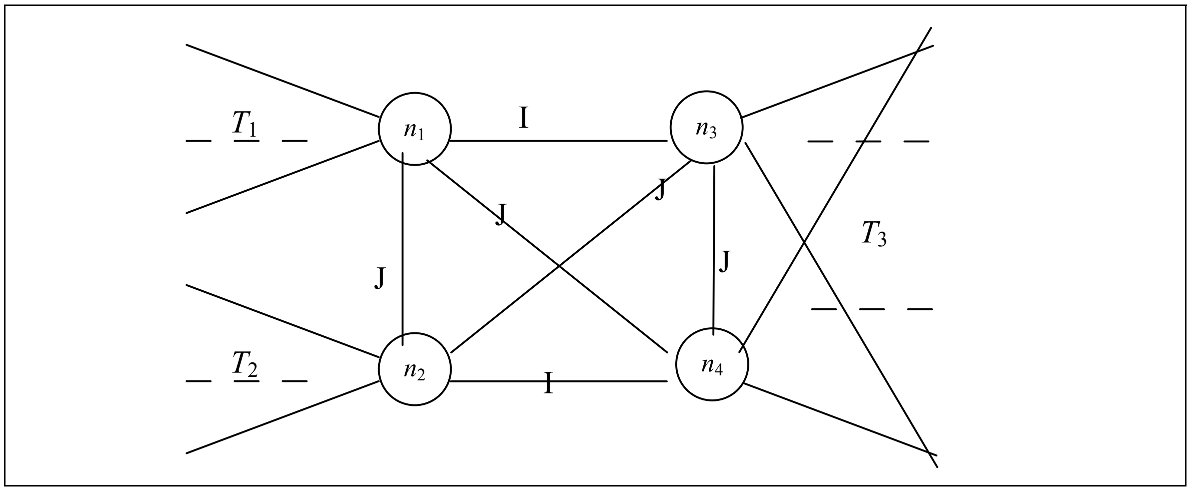
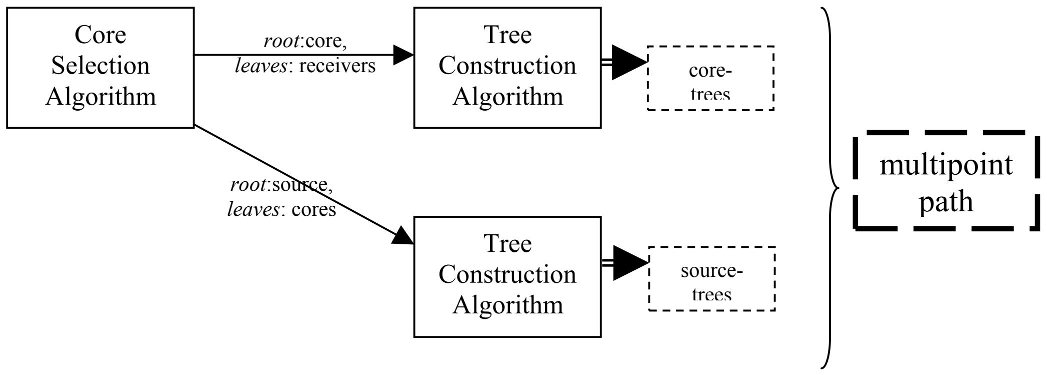


Figure 3. Example case for DCMC solution supporting Lemmas 1 and 2.


Core Selection Algorithm
root:core, *leaves*: receivers
Tree Construction Algorithm
core-trees
root:source, *leaves*: cores
Tree Construction Algorithm
source-trees
multipoint path

Tree construction Algorithm:

Input: tree root, leaf set, delay bound of each leaf
Output: delay-constrained SMT approximation –tree spanning the leaf set meeting the delay-bound of each.

core-tree: *root* is a core, *leaves* are receivers.
source-tree: *root* is a source, *leaves* are cores.

Figure 4. Overview of multipoint communication path construction architecture.

```
Input: $S, R, C$ // source, receiver and candidate core sets
           $D(c,s)$ for all $c \in C, s \in S$
           $path_{delay}[i,j].delay\ \forall\ (i,j): i \in S, j \in C$ and $i \in C, j \in R$
Output: $C_u$ // the set of cores ultimately selected
           tree construction information
E1.. initialize the output parameters to null;
E2.. for (each $c \in C$)   for (each $s \in S$)   {     // compute domination counts
E3..       $domination\_count_{c,s} = |D(c,s)|$;
E4..       for (each $s' \in S, s' \neq s$)
E5..           $domination\_count_{c,s}$ += $|D(c,s) \cap D(c,s')|$;
E6.. }

E7.. $total\_count = 0$;

E8.. repeat
E9..     $max\_count = 0$;
E10..    for (each $c \in C$)   for (each $s \in S$)   // select a core
E11..       if ( ($domination\_count_{c,s} > max\_count$) or (($domination\_count_{c,s} = max\_count$) and ($c$=$s$)) ) {
E12..            $max\_count = domination\_count_{c,s}$;
E13..            $core = c$;
E14..            $source = s$;
E15..       }

E16..   $total\_count$ += $max\_count$;
E17..   $C_u = C_u \cup \{core\}$
E18..   $domination\_count_{core,source} = 0$;
E19..   $D_u(core,source) = D(core,source)$;
E20..   $C = C \setminus \{core\}$;

E21..  for (each $s \in S$)  for (each $r \in D(core,source)$)
          // update the delay-bound of the leaves of the trees and the domination status for the selected core tree
E22..     $\Delta_{source,core} = path_{delay}[source,core].delay$;
E23..     $\Delta_{core,r} = \Delta - \Delta_{source,core}$;
E24..     if ($r \in D(core,s)$)    {
E25..          $D_u(core,s) = D_u(core,s) \cup \{r\}$;
E26..          $L_s = L_s \cup \{core\}$;
E27..          $\Delta_{s,core} = path_{\ delay}[s,core].delay$;
E28..          $\Delta_{core,r} = \min\ \{\Delta_{core,r}, \Delta - \Delta_{s,core}\}$;
E29..          for (each $c \in C, c \neq core$)   {
E30..             $D(c,s) = D(c,s) \setminus \{r\}$;
E31..             if ($r \in D(c,s)$)    $domination\_count_{c,s}$--;
E32..          };
E33..     };
E34..  };
E35..   send $D_u(core,s)\ \forall s$: $D_u(core,s) \neq \notin$; $\Delta_{core,r}\ \ \forall r \in D_u(core,source)$ to $core$;
              // trigger $core$ for construction of the specified core tree
E36..   for (each $s \in S$)  $D_u(core,s) = \emptyset$;
E37.. until ($total\_count = |R|*|S|$)   // all the receivers are dominated for all sources

E38.. for (each $s \in S$)  // trigger the sources for the construction of source trees
E39..      send $L_s$, $\Delta_{s,c}\ \ \forall c \in L_s$ to $s$;
```

Figure 5. *SELECT*, the core selection algorithm of *SPAN*, processing on the singular and non-singular solution spaces.

```
CONSTRUCT_TREE
Input: root: root of the tree
          L: the set of leaves to be spanned
          $\Delta_l$ : delay-bound on the path between root and $l$ $\forall l \in L$
Output: the delay-bound tree
-------------------------------------------------------------
// initialization
C1.. for (each leaf $l \in L$)  { $min\_dist_l = \infty$;   $delay_l = \infty$ }
C2.. $delay_{root} = 0$;
C3.. on_tree_node = leaf = root;
C4.. code = "*";   // value of code doesn't matter at first iteration

C5.. while ($L \neq \emptyset$) {  // iterate until all leaves are on-tree
C6..      send ADD_PATH_msg ($L$, $\Delta_L$, code, leaf, $delay_{on\_tree\_node}$) to on_tree_node;
C7..      delay time-out for incoming PATH_INFO
C8..           for (each leaf $l \in L$)
C9..             if ($min\_dist_l > PATH\_INFO\_msg.min\_dist_l$) {
C10..                 $min\_dist_l = PATH\_INFO\_msg.min\_dist_l$;
C11..                 $code_l = PATH\_INFO\_msg.code_l$;
C12..                 $on\_tree\_node_l = PATH\_INFO\_msg.on\_tree\_node$;
C13..             }
C14..      min_dist = $\infty$;
C15..      for (each leaf $l \in L$)
C16..          if ($min\_dist_l < min\_dist$)  {
C17..              $min\_dist = min\_dist_l$ ;
C18..              $code = code_l$ ;
C19..              $on\_tree\_node = on\_tree\_node_l$ ;
C20..              leaf = $l$;
C21..          }
C22..      $L = L \setminus \{leaf\}$;
C23.. }
```

Figure 6. *CONSTRUCT_TREE* algorithm

```
ADD_PATH (leaf, code, $delay_{self}$, L, $\Delta_L$)
Input:
      $L$, $\Delta_L$: current leaf set information
      leaf: the leaf to be added to the tree
      code : the distance-metric of the path to be constructed leading to leaf
      $delay_n$: the delay distance
Output: The tree link establishment between self and $next_{<code>}$(self, leaf)
-------------------------------------------------------------
A1..  if ( $self \neq leaf$ )   {   // self is the node running ADD_PATH
A2..      child = $next_{<code>}$(self, leaf);   // unicast state-table look-up
A3..      add child as a downstream tree node;   // tree state update
A4..      $delay_{child} = delay_{self} + path_{<code>}[child].delay$;
A5..      relay ADD_PATH_msg ($L$, $\Delta_L$, code, leaf, $delay_{child}$) to child
A6..  }
```

Figure 7. The process *ADD_PATH* initiated by *CONSTRUCT_TREE*. $next_{<code>}$(*parent, leaf*) returns the neighbor of the *parent* on the shortest-path to *leaf* in metric specified in <*code*>. Returns *leaf* if *parent* = *leaf*.

```
MIN_DIST ($n$, leaf, $delay_n$, $\Delta_l$)
Input: n: on-tree node
        leaf: an un-spanned leaf
       $\Delta_{leaf}$ : delay-constraint between root and leaf
       $delay_n$ : the on-tree delay-distance of n from the tree root
Output: the path information connecting n and leaf at minimum-attainable distance.

M1..  $min\_dist = \infty$;

M2..  if ($path_{<distance>}[leaf].delay \le \Delta_{leaf} - delay_n$)  {
M3..       $min\_dist = path_{<distance>}[leaf].<distance>$;
M4..       $code = <distance>$;
M5..  }

M6..  if ($path_{delay}[leaf].delay \le \Delta_{leaf} - delay_n$ & $path_{delay}[leaf].<distance> < min\_dist$ ) {
M7..      $min\_dist = path_{delay}[leaf].<distance>$;
M8..      code = "delay";
M9..  }
M10..   return (min_dist, code)
```
Figure 8. The procedure *MIN_DIST* invoked by *PATH_INFO.*

```
PATH_INFO ($delay_{self}$, $min\_dist_L$, $code_L$, $on\_tree\_node_L$)
Input:      $L$, $\Delta_L$: current leaf set information
Output:    tree information to be reported to the root

P1..   if ( self is not the on-tree end-point node communicated by the root)
P2..        for (each leaf $l \in L$)   {
P3..             $min\_dist_l = MIN\_DIST(self, l, delay_{self}, \Delta_l).min\_dist$;
P4..                if ( (self = leaf) XOR ($min\_dist_l < PATH\_INFO\_msg.min\_dist_l$) )  {
P5..                    $PATH\_INFO\_msg.min\_dist_l = min\_dist_l$;
P6..                    $PATH\_INFO\_msg.code_l = MIN\_DIST(self, l, delay_{self}, \Delta_l).code$;
P7..                    $PATH\_INFO\_msg.on\_tree\_node_l = self$;
P8                }
P9..        }
P10..     relay PATH_INFO_msg ($delay_{self}$, $min\_dist_L$, $code_L$, $on\_tree\_node_L$) to upstream node;
P11..  }
P12..  else send PATH_INFO_msg ($delay_{self}$, $min\_dist_L$, $code_L$, $on\_tree\_node_L$) to root;
```
Figure 9. The process *PATH_INFO* initiated at *leaf.*

```
S1.. If $SUBDOMAIN_s$ is processed already then discard it and exit.
S2.. Read the time-stamp. If it is beyond $\Delta$ then discard and exit.
S3.. for ( each $r \in R$ )
S4..        if ($\Delta_{s,n} + path_{delay}[r].delay \le \Delta$) $D(n,s) = D(n,s) \cup \{r\}$
S5.. if ($D(n,s) = \varnothing$) discard $SUBDOMAIN_s$.
S6.. if ($D(n,s) \neq \varnothing$) {
S7..     send $path_{delay}[s,n].delay$ and
                   $path_{delay}[r].delay$ $\forall$ $r \in D(n,s)$ to coordinator
S8..     for (each link h except the one $SUBDOMAIN_s$ is received)  {
S9..             duplicate $SUBDOMAIN_s$
S10..            increment time-stamp on $SUBDOMAIN_s$ by delay on h
S11..            relay $SUBDOMAIN_s$ on h
S12..    }
S13..}
```
Figure 10. Local processing of $SUBDOMAIN_s$ on a recipient node *n*.

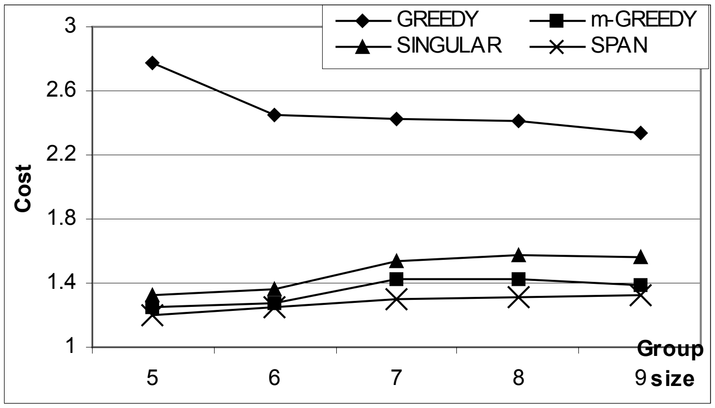


Figure 11. Comparison with optimal solution on group size. $\Delta=\Delta_{critical}$.

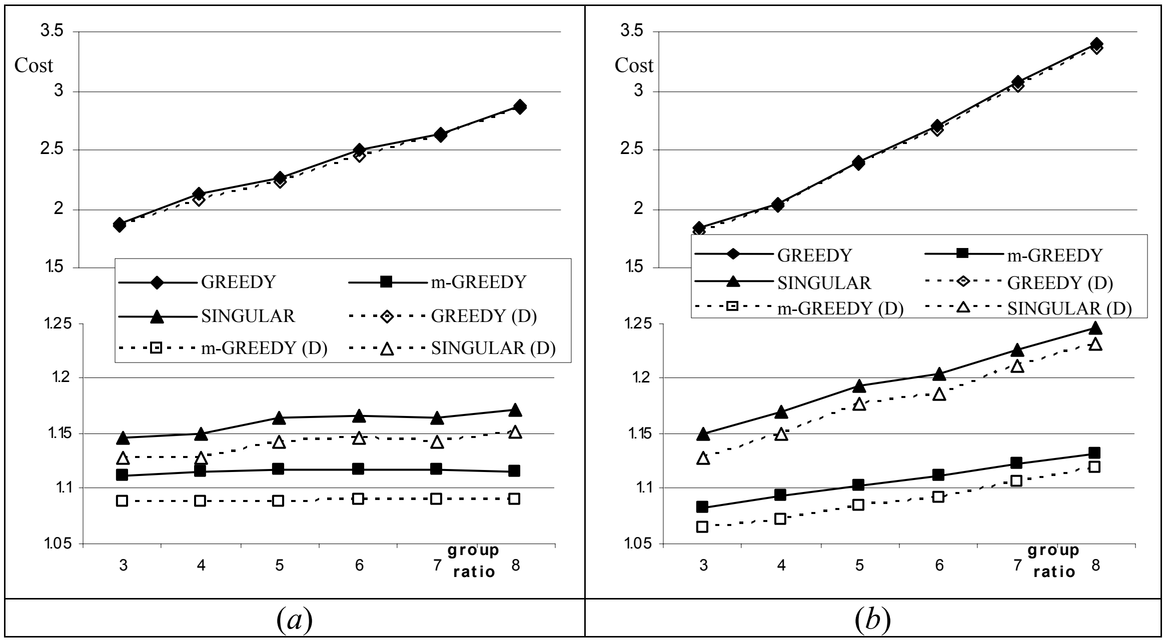


Figure 12.1. Evaluation for cost performance on group ratio. $|R| = 16$, $|S| \in [6, 8, 10, 12, 14, 16]$, $\Delta=\Delta_{critical}$. *a.*) Exactly half of the sources are also receivers. *b.*) all sources are also receivers.

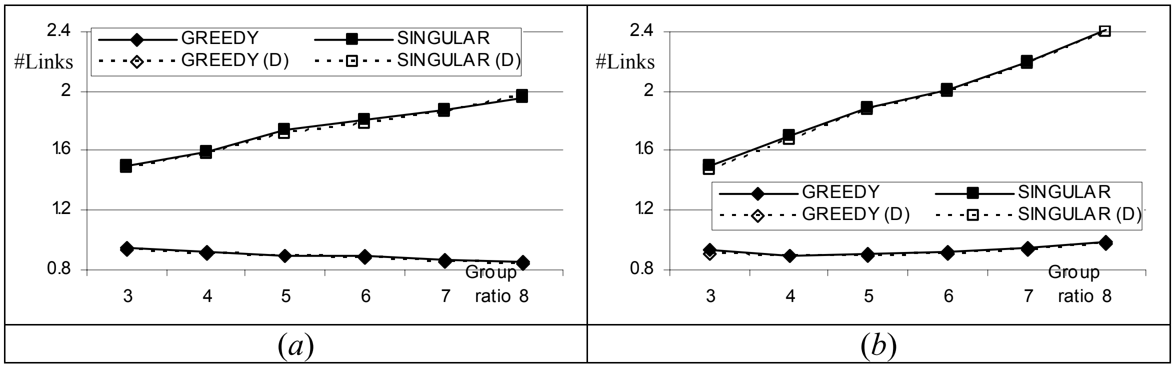
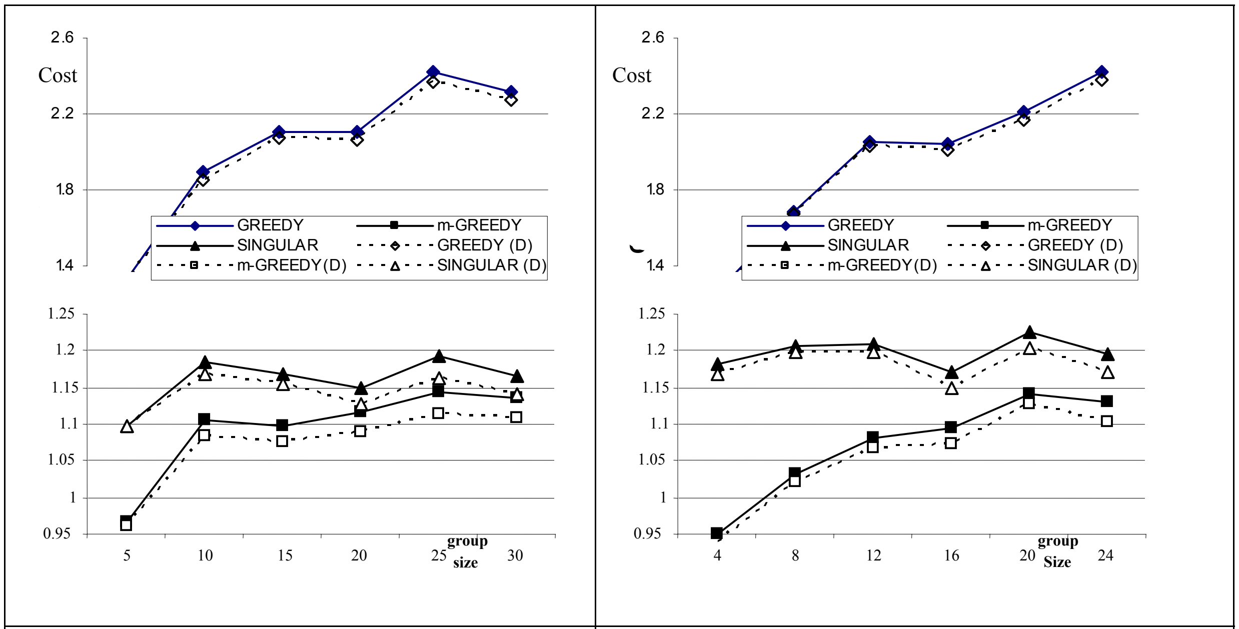
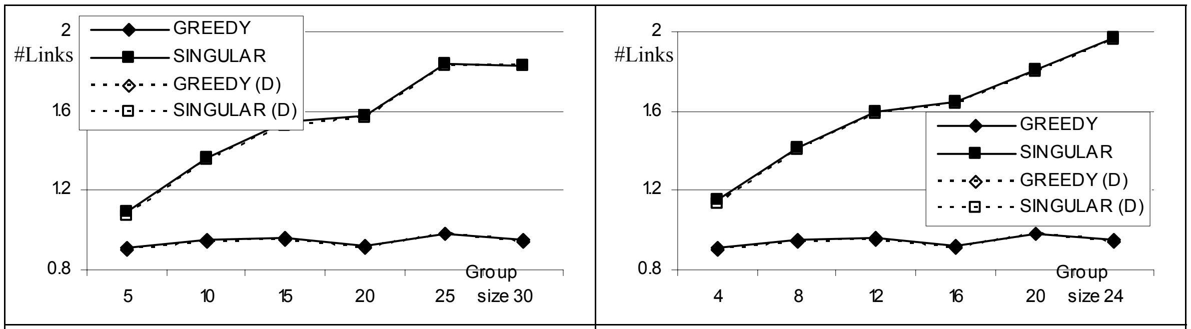


Figure 12.2. Evaluation for hop-count performance on group ratio. $|R| = 16$, $|S| \in [6, 8, 10, 12, 14, 16]$, $\Delta=\Delta_{critical}$. *a.*) Exactly half of the sources are also receivers. *b.*) all sources are also receivers.



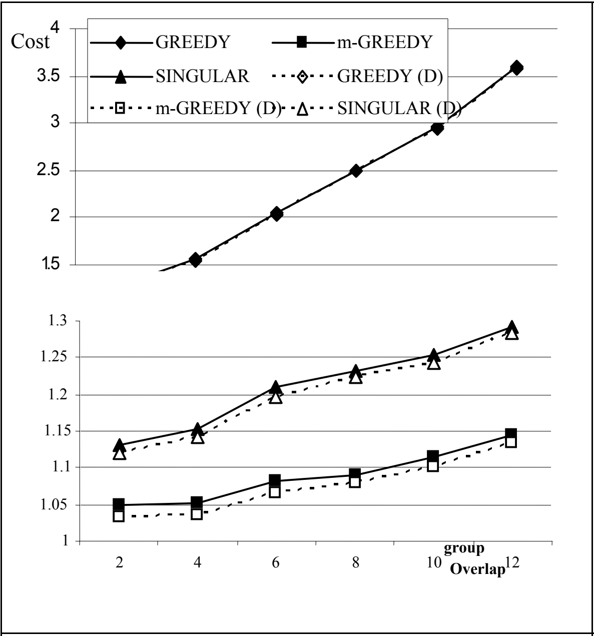
(*a*) (*b*)

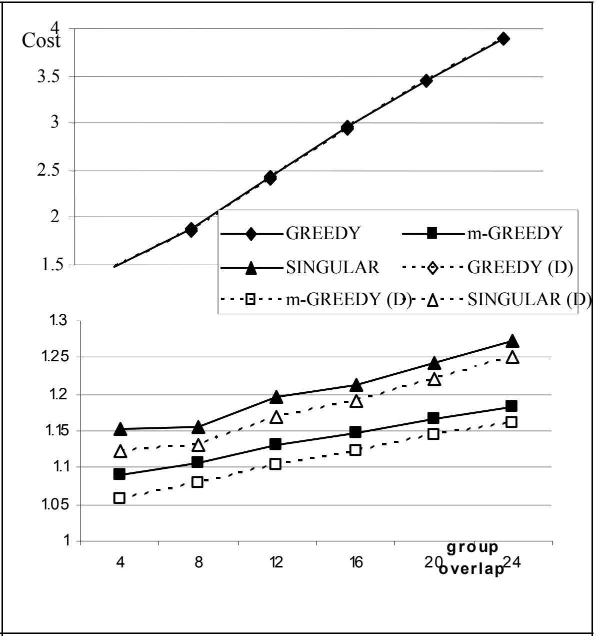
Figure 13.1. Evaluation for cost performance on group size. |*R*| ∈[4, 8, 12, 16, 20, 24], |*S*| ∈ [2, 4, 6, 8, 10, 12], |*S*|/|*R*| = ½. $\Delta=\Delta_{critical}$. *a*.) Exactly half of the sources are also receivers. *b*.) all sources are also receivers.



(*a*) (*b*)

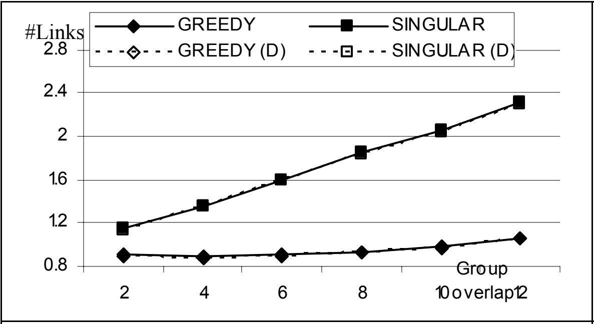
Figure 13.2. Evaluation for hop-count performance on group size. |*R*| ∈[4, 8, 12, 16, 20, 24], |*S*| ∈ [2, 4, 6, 8, 10, 12], |*S*|/|*R*| = ½. $\Delta=\Delta_{critical}$. *a*.) Exactly half of the sources are also receivers. *b*.) all sources are also receivers.

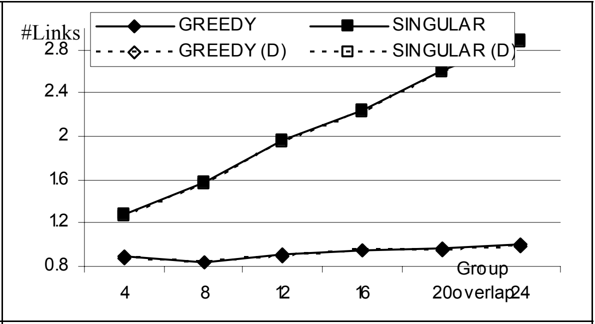




(*a*) (b)

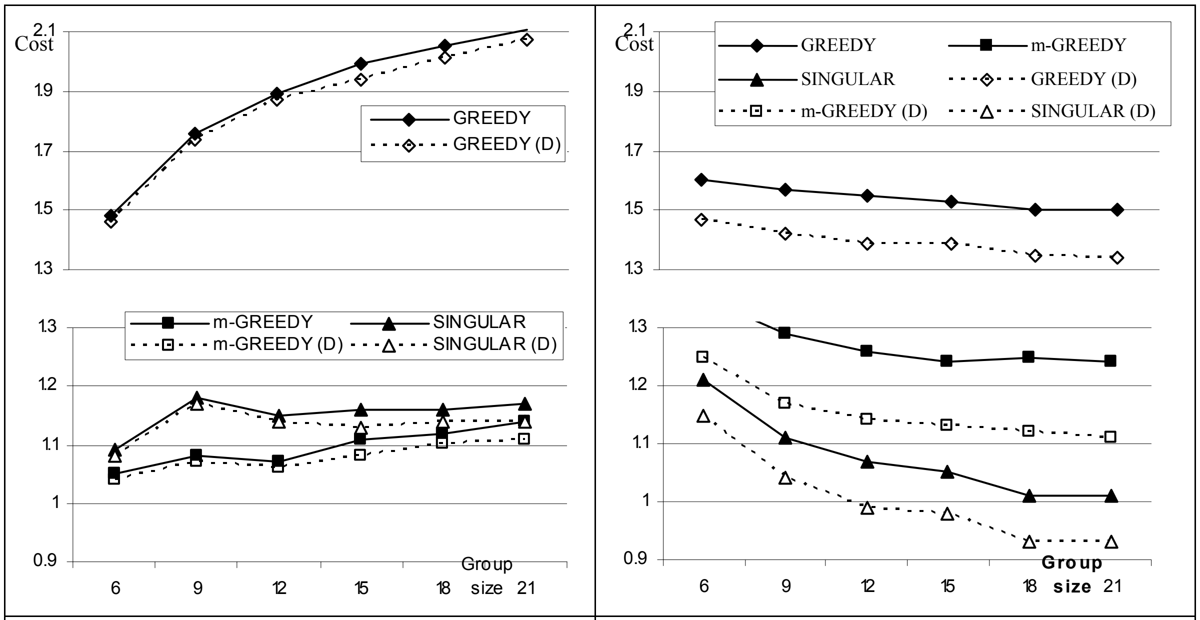
Figure 14.1. Evaluation for cost performance on group overlap. $\Delta=\Delta_{critical}$. All sources are also receivers. *a*.) $|R| = 12$, $|S| = |S \cap R| = [2, 4, 6, 8, 10\ 12]$,
b.) $|R| = 24$, $|S| = |S \cap R| = [4, 8, 12, 16, 20\ 24]$.





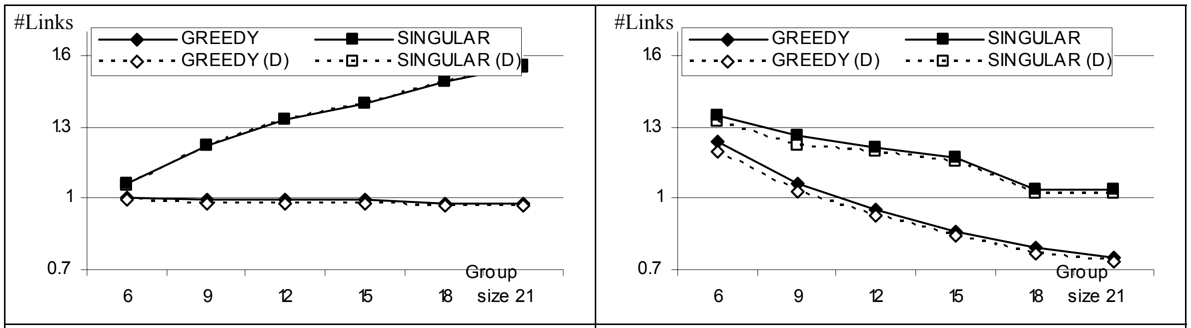
(*a*) (*b*)

Figure 14.2. Evaluation for hop-count performance on group overlap. $\Delta=\Delta_{critical}$. All sources are also receivers. *a*.) $|R| = 12$, $|S| = |S \cap R| = [2, 4, 6, 8, 10\ 12]$,
b.) $|R| = 24$, $|S| = |S \cap R| = [4, 8, 12, 16, 20\ 24]$.



(*a*) (*b*)

Figure 15.1. Evaluation for cost performance on group size. $|R| \in [4, 6, 8, 10, 12, 14]$, $|S| \in [2, 3, 4, 5, 6, 7]$, $S \cap R = \varnothing$, $|S|/|R| = ½$. $\Delta = \Delta_{critical}$. *a.*) $\Delta = \Delta_{critical}$. *b.*) $\Delta = (\Delta_{max} + \Delta_{critical})/2$.



(*a*) (*b*)

Figure 15.2. Evaluation for hop-count performance on group size. $|R| \in [4, 6, 8, 10, 12, 14]$, $|S| \in [2, 3, 4, 5, 6, 7]$, $S \cap R = \varnothing$, $|S|/|R| = ½$. $\Delta = \Delta_{critical}$. *a.*) $\Delta = \Delta_{critical}$. *b.*) $\Delta = (\Delta_{max} + \Delta_{critical})/2$.

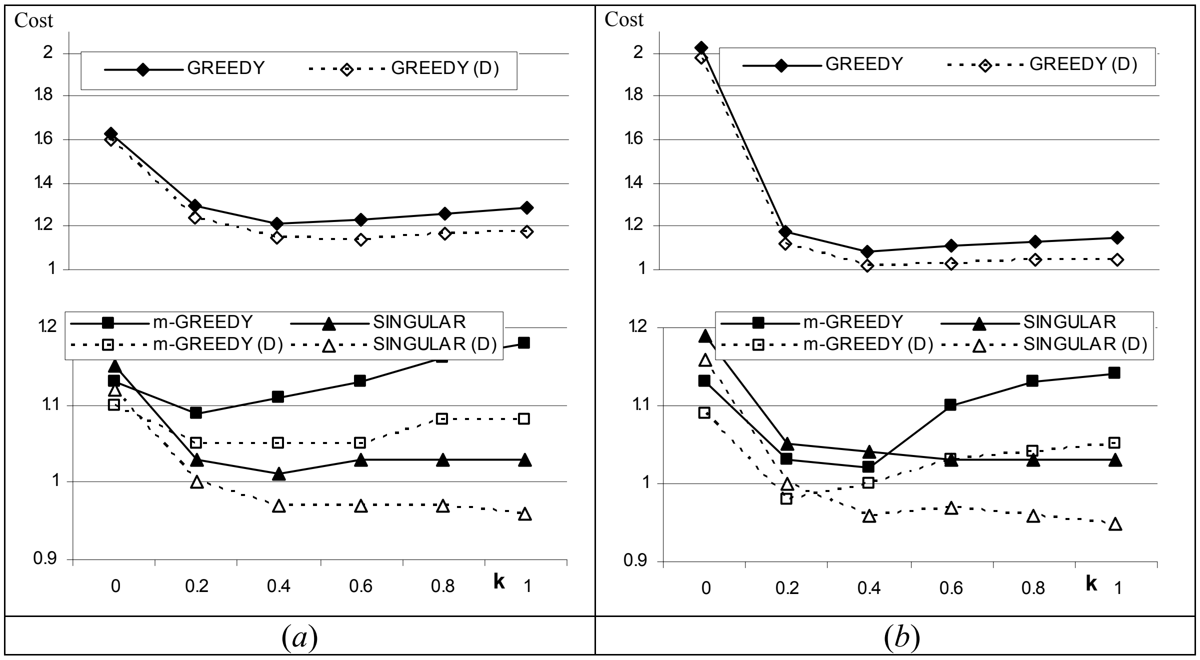


Figure 16.1. Evaluation for cost performance on delay-bound of the application. $\Delta=\Delta_{critical}+kV$ where $V=\Delta_{max}-\Delta_{critical}$, $k\in[0, 0.2, 0.4, 0.6, 0.8, 1.0]$. $|S\cup R|=20$. *a.*) $|S|=4$, $|R|=16$ and $S\cap R=\varnothing$.*b.*) $|S|=8$, $|R|=16$ and $S\subset R$.

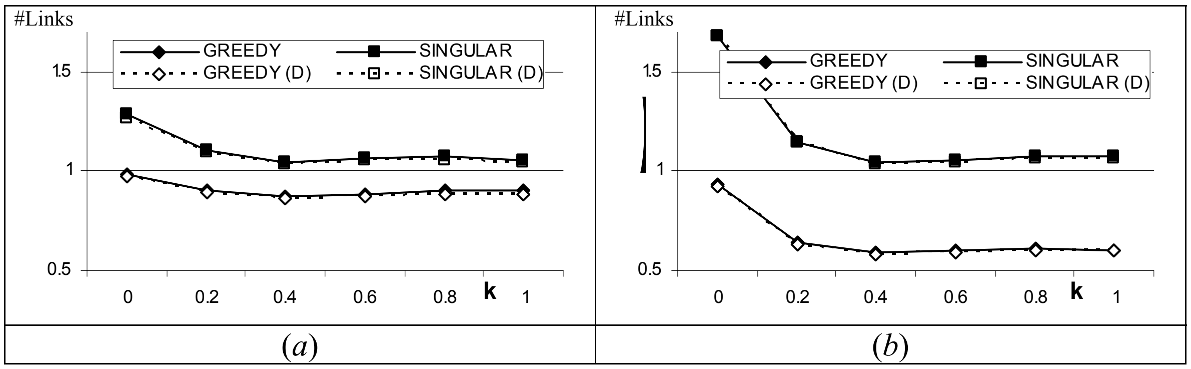


Figure 16.2. Evaluation for hop-count performance on delay-bound of the application. $\Delta=\Delta_{critical}+kV$ where $V=\Delta_{max}-\Delta_{critical}$, $k\in[0, 0.2, 0.4, 0.6, 0.8, 1.0]$. $|S\cup R|=20$. *a.*) $|S|=4$, $|R|=16$ and $S\cap R=\varnothing$.*b.*) $|S|=8$, $|R|=16$ and $S\subset R$.