

Evaluating Steiner tree heuristics and diameter variations for Application Layer Multicast

Knut-Helge Vik, Pål Halvorsen, Carsten Griwodz

Simula Research Laboratory and University of Oslo, Oslo, Norway

Abstract

Latency reduction in distributed interactive applications has been studied intensively. Such applications may have stringent latency requirements and dynamic user groups. We focus on application-layer multicast with a centralized approach to the group management. The groups are organized in overlay networks that are created using graph algorithms that create a tree structure for the group. A tree has no cycles and uses a small routing table, as opposed to a connected overlay mesh.

We investigate a group of spanning tree problems that are referred to as Steiner tree problems, and we have a particular focus on reducing the diameter of a tree, which is the maximum pairwise latency in a tree. In addition, we focus on reducing the time it takes to execute membership changes. In that context, we use core-selection heuristics to find well-placed client nodes, and edge-pruning algorithms to reduce the number of edges in an otherwise fully meshed overlay. Our edge-pruning algorithms strongly connect well-placed client nodes to the remaining group members, to create new and pruned group graphs. Consequently, when a tree algorithm is applied to a pruned group graph, it is manipulated into creating trees with a smaller diameter.

We devised new Steiner-tree heuristics that reduced the diameter, and also proposed new edge-pruning algorithms to make the tree construction faster. These heuristics and algorithms were implemented and analyzed experimentally along with several spanning-tree and core-selection heuristics found in the literature. We found that a full-mesh of shortest paths makes it difficult for Steiner-tree heuristics to find better trees than spanning tree algorithms. The network seen from the application layer is in fact a full mesh of shortest paths. In addition, we found that faster Steiner-tree heuristics that do not explicitly optimize the diameter are able to compete with slower heuristics that do optimize it.

Email address: `knuthelv,griff,paalh@ifi.uio.no` (Knut-Helge Vik, Pål Halvorsen, Carsten Griwodz).

1 Introduction

In recent years, many new types of distributed application have appeared. This is mainly due to the large improvements in computer technology, which has resulted in more resources available over the Internet. The media types may range from text to continuous media such as video streams. Distributed interactive applications, such as virtual environments and online games, currently have millions of users and generate more money than the film industry [1,2].

Although distributed interactive applications may differ greatly, they share many of the same requirements. Firstly, groups of users (but not necessary all users) of the same application must be able to interact. Their interactivity imposes restrictions on network latency, especially in highly interactive virtual environments [3,4]. Further, because the users in virtual environments interact, there is a need for a many-to-many communication function. Many-to-many communication, combined with restrictions on network latency, results in requirements on the latency between any pair of users (pairwise latency).

Many distributed interactive applications have highly dynamic user groups. For example, users in online games may join and leave groups continuously as they move around in a virtual environment. Whenever group membership is determined anew, the many-to-many communication paths need to be updated. Here, the main challenge is to design algorithms that create efficient (low latency) event distribution paths, that take into account the physical location of the users that are interacting.

Today, many distributed interactive applications are centrally managed. Bearing this in mind, we here focus on an architecture with centralized management. A completely centralized architecture gives the application provider full control. However, if all the data travels through the server, the latency is potentially too high for users that are located far from the server. When the latency increases as a result of the distance of the user from the server, it might be better to allow the data to travel between the users directly. This will reduce both the overall latency and the pairwise latency experienced by the users. Our goal is to identify efficient means by which data can flow among the users while at the same time taking into account the group dynamics. It is in this context that we are studying group communication algorithms that organize the users in distribution patterns that have varying properties.

We use application layer multicast to achieve group communication, although many distributed interactive applications support IPv4 multicast. Reasons for operating at the application layer are that IPv4 multicast 1) is not supported by all Internet service providers, 2) cannot be used efficiently with TCP, 3) does not easily support frequent membership change, 4) cannot pre-

vent snooping, and 5) has a rather limited address space available. To avoid these problems, we build overlay networks. Such overlay networks are built between the users' computers and are (inherently) fully meshed. Techniques for estimating link costs are often applied to overlay meshes [5], but this is outside the scope of our paper.

We study a range of new and existing graph algorithms that organize nodes in trees while conforming to some optimization goal. A tree is acyclic and thus needs very small routing tables. Furthermore, a tree reduces the administration cost and bandwidth consumption compared to a connected mesh. We here concentrate on algorithms that minimize the pairwise latency in distribution trees. The maximum pairwise latency of a tree is known as the diameter. More specifically, we study a class of spanning tree problems called Steiner-tree problems, and evaluate many Steiner-tree heuristics that reduce the diameter. We used algorithm ideas from the well-known minimum-cost Steiner-tree heuristics shortest-path heuristic (SPH) [6], distance-network heuristic (DNH) [7] and average-distance heuristic (ADH) [8]. From these algorithms we devised new Steiner-tree heuristics that reduce the diameter. The algorithms SPH and DNH are themselves based on ideas from Prim's minimum spanning tree (MST) and Dijkstra's shortest path tree (SPT), while ADH is based on Kruskal's MST [9]. We also used ideas from existing spanning tree algorithms on how to keep track of the pair-wise latencies in a graph [10].

All the algorithms are implemented, and the performance is analyzed using experiments. We found that a full-mesh of shortest paths makes it difficult for Steiner-tree heuristics to find better trees than spanning tree algorithms. In this case, many of the Steiner-tree heuristics only has a best-case (and worst-case) performance equal to an MST. The network seen from the application layer is in fact a full mesh of shortest paths. Therefore, these Steiner-tree heuristics should not be used when a full mesh of shortest paths is the input graph. We also reduced the full mesh using pruning algorithms and used the pruned graph as input to the Steiner-tree heuristics. However, we still found that faster Steiner-tree heuristics that do not use shortest path information and do not explicitly optimize the diameter are able to compete with slower heuristics that do.

The rest of the paper is organized as follows. In section 2 we introduce distributed interactive applications with examples and latency requirements. Furthermore, in section 3 we discuss the diameter of overlay networks, group communication in the Internet, graph algorithm requirements, and a centralized approach to membership management. Sections 4 and 5 introduces the graph algorithms that we experimentally tested in this paper. Furthermore, section 6 explains the experiments and shows the results. The related work is given in section 7, and finally, the conclusions in section 8 summarizes our contribution.

2 Distributed interactive applications

In this paper, the main application for our graph algorithms is distributed interactive applications. These applications achieve live interaction between multiple participants over the Internet. Recently, they have become more popular and also gained increased interest among researchers.

2.1 Application examples

Concrete examples of distributed interactive applications are multi-player online games, audio/video conferences, and virtual reality applications linked to education and entertainment.

Among these, it is the multi-player online games that has received most attention recently. They allow thousands of users to interact concurrently in a persistent virtual environment. The most common categories of online games are first-person shooter games, role playing games and real-time strategy games. These games often cost millions to develop, but the game companies still manage to earn the spendings back and make millions through sales and advertisements. The audio/video conference applications are used to setup meetings where the participants are geographically separated. For example, many multinational companies use video conference systems. Currently, these audio/video conference systems are not readily available for the general public, but rather needs additional equipment to work. In the virtual reality applications a participant controls an avatar (application character) that interacts with a virtual world (hence the name, virtual reality). Examples are combat-, flight- and boat-simulators, virtual museums, shopping malls etc. Virtual reality applications and multi-player online games are now very similar. It is the game companies and the military that are the driving forces for improving virtual reality applications.

The common denominator among the distributed interactive applications is the lack of solutions for easy and cheap deployment over the Internet. Currently, there are few if any peer-to-peer multi-party distributed interactive applications. Rather, they mostly rely on a centralized (client-server) architecture, where the communication flows through a server. An important challenge is to develop systems that allows a peer-to-peer style communication, since it will probably make the distributed interactive applications more scalable and consequently cheaper to deploy.

2.2 *Application requirements*

In this paper, we focus on the latency requirements of distributed interactive applications. The characteristics of game traffic have been analyzed several times before, and in [4], the latency requirements were measured to be approximately 100 ms for first-person shooter games, 500 ms for role-playing games and 1000 ms for real-time strategy games. In video/audio conferencing and voice over IP (VoIP) with real-time delivery of voice data, users start to become dissatisfied when the latency exceeds 150-200 ms, although 400 ms is acceptable in most situations [11]. The virtual reality applications have latency requirements that fall into one or more multi-player online game categories.

On the basis of these observations, we conclude that communication paths (overlays) should be constructed such that the maximum pair-wise latency (diameter) falls within the requirements of a given application. The diameter requirements may vary from strict to loose even within one application.

3 **Overlay network diameter**

The diameter of an overlay network is determined by the longest shortest path. We are investigating graph algorithms that design overlay networks in ways that reduce the diameter, in particular Steiner tree problems found in the literature. The main advantage of Steiner tree algorithms is that they are aware of member-nodes and non-member-nodes, which is important when grouping clients (see section 5). Before we introduce the algorithms, we highlight the challenges and choices an algorithm designer must make when using group communication in the Internet. Then, we introduce centrally managed group communication and the specific goals.

3.1 *Group communication in the Internet*

Group communication in networks has been studied for quite some time already. Especially, the reduced latency and increasing bandwidth available on the Internet has enabled many new application types that may use group communication. However, we are yet to see the peak of group communication over the Internet. Particularly, distributed interactive applications like multi-player online games and interactive peer-to-peer applications are going to be major applications for group communication in the years to come.

Group communication in the Internet poses challenges related to asymmetry,

heterogeneity, resource availability and latency. One way of modelling some of these challenges is to apply graph theory. In graph theory, a network may be modelled as a *directed* or *undirected* graph. Directed edges (arcs) allow asymmetric links, while undirected edges are symmetric. Currently, asymmetric links are the most common situation for clients in the Internet. However, the symmetric link assumption of undirected graphs is only considered unrealistic in highly asymmetric networks, and in applications that require high bandwidth and low latencies. Current multi-player online games support few if any such flows of content or events, rather, the streams are very thin [12].

It is possible to use multicast to accomplish content and event distribution through a network. And, in that respect, a connected acyclic graph (tree) has several advantages over a connected mesh. A tree has small routing tables and saves network bandwidth. In addition, it has low administration costs when the membership is dynamic. However, the pair-wise latencies do increase, and if a non-leaf node is disconnected from a tree, the tree is also disconnected resulting in two subtrees. Generally, there are two main directions of tree building related to group communication in networks. Trees that are built for a *single source* and trees built for *multiple sources*.

For single source situations, a shortest-path tree (SPT) [9] is often used. An SPT give minimum latency routes between a pre-chosen source and all destinations. However, SPTs are costly because they consume much of the network resources. Especially forwarding nodes close to the source may experience massive stress issues because they have a high *degree*, where the degree of a node is the number of incident edges in a graph. Alternatively, a minimum-cost spanning-tree (MST) [9] may be used. An MST is constructed such that the *total cost* is minimized, i.e., the sum of the edge weights. However, the latencies between the source node and destinations are higher than in SPTs. Clearly, there is a tradeoff between the source destination latencies and the total cost of a tree. *Shallow-light* trees [13] are trees with a single source that simultaneously approximate well both an MST and an SPT. A shallow-light algorithm computes a tree that is at most a constant times heavier than the MST, and with the property that the diameter of the tree is at most a constant times the diameter of the input graph.

For multiple source situations, it is vital to reduce the pair-wise latency, such that every source is within a constant latency bound of every destination (see section 2.2). That is, the *eccentricity* of the destinations should be minimized, where the eccentricity is defined as the maximum pair-wise latency from a single node. In our group communication scenario, every node in the network is a source that distributes data. In situations where all nodes are sources, the tree should be constructed as a *shared-tree*. For such a shared-tree scenario, it is not enough to only consider a tree viewed from a single source or a set of sources. Every node is both a source and a receiver, thus the worst case

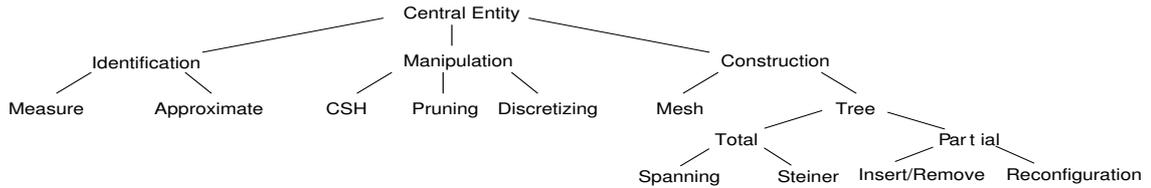


Fig. 1. Centralized membership management and the process tree of the central entity.

eccentricity in a shared tree equals the diameter. Therefore, in a shared tree scenario, the diameter is a very important metric.

3.2 Graph algorithm requirements

Our goal is to identify graph algorithms that reduce or limit the diameter in trees while being able to cope with group membership dynamics. That is, the complexity and consequently the execution time of the algorithms should be low, such that the *reconfiguration time* is swift. In addition, we want to bound the *stress*, i.e., work-load, of each node in the tree, such that it falls within the computational capacity of a given node. The stress of a given node is linked to the number of incident edges it has in a graph, which is the *degree* of the node.

3.3 Centralized membership management

Many distributed interactive applications, like online games, applies a centralized architecture, where a centralized server stores the entire game world and its state. Similarly, our focus is on centralized membership management, where a central entity stores information about all the users in a distributed application. The central entity employs a process by which it identifies a fully meshed global graph that contains properties of the nodes and links in the overlay network, for example a coordinate discovery system [5]. The users are dynamically grouped and the central entity stores a fully meshed group graph for each group. A group graph is a subgraph of the global graph. The central entity creates and manages one distribution tree for each group. A distribution tree is built such that the data flows directly between the users in a group (see figure 2).

A central entity employs three processes for the membership management. The *identification* process identifies communication properties among users and determines how users are grouped. The *manipulation* process has optional techniques to reduce the time required for membership updates. The

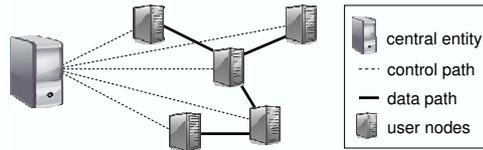


Fig. 2. Centralized architecture with control and data paths

techniques considered in this paper are core selection heuristics and prune algorithms (sections 4.1 and 4.2). In the *construction* process, the latest group graph information is used as input to a graph algorithm that creates/updates the group distribution graph. The construction algorithms that we consider here are tree algorithms that create distribution trees (section 5.4).

Figure 1 illustrates the central entity and its processes that together form the centralized membership management. The tree may be viewed as a process tree and a research area tree. The algorithms we present here are related to manipulation and construction. Extensive research that addresses identification may be found in [5]. Identification is independent of manipulation and construction, and its discussion lies outside the scope of this paper.

In the following sections, we introduce algorithms for centralized group management in distributed interactive applications.

4 Manipulation algorithms

The central entity employs a manipulation process in which specialized algorithms for graph manipulation are contained. These algorithms may, for example, edit the graph layout, search for certain nodes, change some link weights etc. The manipulation algorithms should be used such that the construction process runs more smoothly. In this paper, we present some core selection heuristics (CSHs) and pruning algorithms.

4.1 Core selection heuristics

An important group management technique is to elect one or more nodes to administrate clients that join and leave groups. When a limited set of nodes handles the membership management, it simplifies membership updates, and applications that have highly dynamic groups require fast and simple group management. Names given to such nodes include leaders, core nodes or rendezvous-points. Typically, the core node of a group is contacted for each membership change, such that it always has the up-to-date (membership) information. Protocols with core nodes are often defined as core-based

multicast protocols.

Core-based multicast protocols work on the assumption that one or more core nodes are selected as group management and forwarding nodes. These protocols need some core selection heuristic to select the cores. Several core selection heuristics have been proposed, and a comprehensive study is given by Karaman and Hassanein [14]. An overall goal is to select cores on the basis of certain node properties, such as, bandwidth and computational power. We base our core selection primarily on latency, but also the degree limitations of the available core nodes.

By applying core selection heuristics that identify well-placed nodes in a group, we design algorithms that exploit their degree capacity and reduce the diameter of the group tree. For example, degree-limited heuristics often exhaust the degree limits on centrally located nodes in the input graph. In our scenario, core nodes are included in the input graph to be non-member-nodes (aka. Steiner points) for the Steiner-tree heuristics.

We consider the best location for a core to be related to the location of the group member-nodes. The core selection heuristics presented here search for a set of vertices beginning with the *graph median* [15,16]. The graph median is the vertex (node) for which the sum of lengths of shortest paths to all other vertices is the smallest. The heuristics are [14]:

- *Topology Center*: Find s nodes that are closest to the topological center of the global graph.
- *Group Center*: Find s nodes that are closest to the group center of the group graph.

4.2 Prune algorithms

The complexity of an algorithm has a major influence on the execution time, which is important for our target applications. It is, however, difficult to reduce the complexity of an algorithm without greatly decreasing the quality of the outcome. But, the execution time does also depend heavily on the input graph. As means for modifying input graphs we apply *vertex pruning* and *edge pruning* as such techniques.

Steiner algorithms distinguish between member-nodes and non-member-nodes. They prune unused non-member-nodes inherently when building trees (vertex pruning). The vertex set V in an input graph G is built from the member-nodes in the group plus a limited set of non-member-nodes. The non-member-nodes are selected using the core selection heuristics introduced previously. If there are only member-nodes in the input graph, the problem reduces to finding an

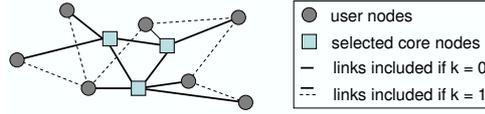


Fig. 3. Pruned graph using *add* Core Links Optimized.

MST. Hence, it is only useful to apply Steiner algorithms when the input graph includes non-member-nodes. By using group member-nodes and stronger core nodes selected using a core selection heuristic, the number of vertices in an input graph is bounded by the number of member-nodes, Z , and the number of non-member-nodes, i.e., core nodes, that are included. The result is the vertex subset V_Z of G , where $V_Z - Z$ is the set of non-member-nodes.

A fully meshed overlay network makes the size of the edge set particularly large. We investigate edge pruning algorithms that reduce the size of the edge set from a fully meshed input graph. We have shown previously that using a fully meshed member graph as input to a tree algorithm may double the execution time compared to a graph with a pruned edge set [17]. The goal of our edge-pruning algorithms is to create a new pruned graph with a smaller edge set, where a set of core nodes (non-member-nodes) is connected to the member-nodes. The core nodes are identified using a core selection heuristic. In this paper we present two promising edge-pruning algorithms:

- *add Core Links (aCL)* (see algorithm 1) takes as input a fully meshed group graph G and a set $O \subset V$ containing group nodes (users) that was identified by a core selection heuristic. Each node in $V - O$ includes its k best edges to the new pruned graph. Then, each node in O includes edges to all nodes in V into the new pruned graph. Step 1 is k Best Links (k BL) as defined in [18]. Step 2 was added to connect the core nodes "strongly" to the remaining nodes, as well as to ensure the connectedness of the new pruned graph. *aCL* produces a graph with $|E| = k * |V - O| + |O| * |V|$. After applying *aCL*, the new pruned graph forms, conceptually, a two-layer graph, where the core nodes are fully meshed and the remaining nodes have a degree that is limited by $k + |O|$.
- *add Core Links Optimized (aCLO)* (see algorithm 2) creates a new pruned graph with an even smaller edge set. It reduces the number of edges from the core nodes O to the remaining nodes $V - O$. Figure 3 illustrates a new pruned graph after using *aCLO*.

Algorithm 1 *add* Core Links

In: A fully meshed graph $G = (V, E, c)$, a set $O \subset V$ of core nodes (selected by a core selection heuristic), and an integer $k \geq 0$.

Out: A graph $G_P = (V_P, E_P, c)$, where $V_P = V$, $E_P \subset E$.

- 1: For each node $m \in V - O$, include k edges to $E_P \subset E$, where E_P contains the minimum weight edges connecting m to $V - O$. {kBL}
 - 2: For each core node $o \in O$, include an edge to every node $v \in V$.
-

Algorithm 2 *add Core Links Optimized*

In: A fully meshed graph $G = (V, E, c)$, a set $O \subset V$ of core nodes (selected by a core selection heuristic), and an integer $k \geq 0$.

Out: A graph $G_P = (V_P, E_P, c)$, where $V_P = V$, $E_P \subset E$.

- 1: For each node $m \in V - O$, include k edges to $E_P \subset E$, where E_P contains the minimum weight edges connecting m to $V - O$. {kBL}
 - 2: Create $|O|$ disjoint sets $l \subset L$ of nodes from $V - O$, where $|l| = |V - O|/|O|$. Each node $o \in O$ connects to all nodes in a set $l \in L$.
-

5 Construction algorithms

In the central entity, the construction process uses an available group graph which is created using the manipulation algorithms introduced above. It then chooses a graph algorithm that creates a connected graph for the distribution of the group events. In this paper, the construction process chooses a Steiner tree heuristic that creates a group tree. Below, we introduce the class of Steiner tree problems and the Steiner-tree heuristics we have used in our experiments.

5.1 Steiner tree problem

Graph algorithms that build trees (tree algorithms) have been heavily researched for many decades. They compute an acyclic graph (tree) from a connected input graph, while satisfying certain criteria for optimization.

Problem 1 *Steiner minimum tree in networks (SMT)*: *Given an undirected weighted graph $G = (V, E, c)$, where V is the set of vertices, E is the set of edges, and $c : E \rightarrow \mathbb{R}$ is the edge cost function, and there is a subset $Z \subseteq V$ of member-nodes; Find a connected undirected tree $T_Z = (V_Z, E_Z)$ of G , where $V_Z \subseteq V$, $Z \subseteq V_Z$ and $E_Z \subseteq E$, such that there is a path between every pair of member-nodes, and $\sum_{e \in E_Z} c(e)$ is minimized.*

The problem of finding a Steiner minimum tree in networks (SMT) is an *NP*-complete problem that was originally formulated independently by Hakimi and Levin [19] in 1971. Several exact algorithms and heuristic have been suggested, implemented and compared [20].

An SMT-algorithm finds a least cost tree connecting a set of member nodes ($Z \subseteq V$). The tree may contain a subset ($V - Z$) of non-member-nodes (Steiner points), that reduces the cost of the tree. SMT-heuristics are often applied to network layer multicast, where the routers are Steiner points and the clients are member-nodes. More recently, SMT-heuristics have been used for application layer multicast, where the Steiner points may, for example, be proxies or well-placed clients (cores).

Problem 2 *Degree limited Steiner minimum tree in networks (d-SMT)*: Given G and Z , a degree bound $d(v) \in N$ for each vertex $v \in V$; Find a SMT T of G and Z of minimum total cost, subject to the constraint that $d_T(v) \leq d(v)$.

A d -SMT-algorithm finds a least cost tree (like SMT-algorithms), where each node obeys a given degree limit. The degree limit provides a degree or stress control to the nodes. The d -SMT problem has been studied by, for example, Bauer and Varma [21]. They introduced several heuristics that are directly derived from SMT heuristics.

From these Steiner tree problems, that yield least cost trees, other Steiner tree problems have been derived. Our focus here is on Steiner tree problems that may reduce the *diameter* of T . The *diameter* of T is defined as the longest of the paths in T among all the pairs of nodes in V_Z . In addition, we study algorithms that optimize for the *total cost*, i.e., the sum of the edge weights in T , while obeying a given diameter bound. In the following, we introduce problem definitions of Steiner tree problems that may reduce the diameter. Other studies of similar problems can be found in [22–25].

5.2 Steiner tree problems and the diameter

Most Steiner tree problems are *NP*-complete, and such are also many of the *diameter*-related Steiner problems. Algorithms that solve the exact problems are obviously slow and therefore useless in our dynamic scenario. Instead, we focus on polynomial time heuristics that produce close to optimal solutions.

Problem 3 *Steiner minimum diameter spanning tree problem (Steiner-MDST)*: Given G and Z ; Find a Steiner spanning tree T of G and Z such that the maximum weight shortest path (*diameter*) $p \in T$, $\sum_{e \in p} W(e)$ is minimized.

A Steiner-MDST algorithm builds a tree of minimum *diameter*. Ho, Lee, Chang and Wong [26] considered the case in which the graph G is a complete Euclidian graph induced by a set of points in the Euclidian plane. They call this special case the *geometric* Steiner-MDST problem. They prove that there is an optimal tree in which a single Steiner-point is connected to all the vertices in Z , and that it is solvable in polynomial time.

Hence, for a complete graph, a simple heuristic for building a close-to-optimal Steiner-MDST is to find a single node located close to the center of the graph that connects to the remaining nodes through shortest paths. The topology of the resulting tree T is that of a star. Consequently, the work-load (stress) of the center node becomes significant as the degree increases. The degree is the

number of incident edges a node has. Thus, a solution is not viable unless the center node has a considerable amount of resources.

Problem 4 *Bounded diameter Steiner minimum tree problem (BDSMT):*

Given G and Z , and a diameter bound $D \in \mathbb{R}$; Find a Steiner minimum tree T on G and Z , where $\sum_{e \in E_T} c(e)$ is minimized and whose diameter does not exceed D .

A BDSMT-algorithm builds a tree of minimum *total cost* within a diameter bound. An advantage of BDSMT over Steiner-MDST is that it is possible to tune the tree diameter while minimizing the *total cost*. This property is vital in cases of lighter application requirements, because the time complexity of the BDSMT-heuristic decreases with looser diameter bounds. However, one problem with BDSMT remains the potentially high node degree of central nodes in the tree when the diameter bound D is stringent.

Problem 5 *Steiner minimum diameter degree limited spanning tree problem (Steiner-MDDL):* *Given G and Z , a degree bound $d(v) \in N$ for each vertex $v \in V$; Find a Steiner tree T of G and Z of minimum diameter, subject to the constraint that $d_T(v) \leq d(v)$.*

A Steiner-MDDL-algorithm builds a tree of minimum diameter while obeying the degree limits. The Steiner-MDDL problem avoids the stress issue that beset spanning tree problems that do not have limitations on degree. One issue with the Steiner-MDDL problem is that it is a minimization of the maximum diameter within a degree limit, which increases the complexity of a heuristic.

Problem 6 *Bounded diameter degree limited Steiner minimum tree problem (BDDLSMT):* *Given G and Z , a diameter bound $D \in \mathbb{R}$, and a degree bound $d(v) \in N$ for each vertex $v \in V$; Find a Steiner minimum tree T on G and Z , where $\sum_{e \in E_T} c(e)$ is minimized, subject to the constraint that the diameter does not exceed D , and $d_T(v) \leq d(v)$.*

A BDDLSMT-algorithm builds a tree of minimum *total cost* within a diameter bound, subject to a degree limit. Like the BDSMT heuristics, a BDDLSMT-heuristic is able to produce trees with a diameter that is in accordance with the diameter bound it received. In addition, the BDDLSMT problem solves the stress issues of BDSMT by using degree limits for each vertex. However, the added degree constraint makes BDDLSMT a harder problem to solve than BDSMT.

5.3 Steiner tree problems and the radius

There exist related spanning tree problems that do not explicitly consider the diameter, but are cheaper in terms of the reconfiguration time. In the following, we introduce problems that bound the *radius* of T , where the radius is defined as the longest of the paths to a pre-defined source node in V . We believe that heuristics of the following problems combined with a well-placed source (core) node may be able to compete with Steiner-tree problems that explicitly consider the diameter.

Problem 7 *Steiner minimum radius spanning tree problem (Steiner-MRST)*: *Given G and Z ; Find a Steiner tree T on G and Z , starting from a root node $s \in V$, where, for each $z \in Z$ the path $p = (z, \dots, s)$ minimizes $\sum_{v_i \in p} c(e(v_i, v_{i+1}))$, where $e \in E_T$.*

A Steiner-MRST algorithm builds a tree of minimum *radius* and is solvable in polynomial time. The optimal Steiner-MRST-algorithm is equal to connecting the root node $s \in V$ to the member-nodes in Z through shortest paths. If the root node $s \in Z$, and the input graph is a full mesh of shortest paths, the optimal Steiner-MRST is always Dijkstra's SPT. In addition, if the root node s is located close to the center of G , a Steiner-MRST heuristic and a Steiner-MDST heuristic would produce trees with similar diameter and radius.

Problem 8 *Bounded radius Steiner minimum tree problem (BRSMT)*: *Given G and Z , a root node $s \in V$ and a radius bound $R \in \mathbb{R}$. Find a Steiner minimum tree T on G , where, for each $z \in Z$ the path $p = (z, \dots, s)$, with weight $\sum_{p_i \in p} c(p_i) \leq R$.*

A BRSMT-algorithm builds a tree of minimum *total cost* within a radius bound from a given root node s to all destinations. The BRSMT-problem is *NP*-complete, and is similar to the shallow-light tree problem, introduced in section 3.1. It is applicable to a shared-tree environment if the root node s is close to the center of G , since it is then approximating a BDSMT.

Problem 9 *Steiner minimum radius degree limited spanning tree problem (Steiner-MRDL)*: *Given G and Z , a degree bound $d(v) \in N$ for each vertex $v \in V$; find a Steiner tree T on G and Z , starting from a root node $s \in V$, where, for each $z \in Z$ the path $p = (z, \dots, s)$ has minimum cost and is subject to the constraint that $d_T(v) \leq d(v)$.*

A Steiner-MRDL algorithm builds a Steiner tree of minimum radius in which each vertex is subject to a degree limit. The Steiner-MRDL problem is *NP*-complete. One Steiner-MRDL-heuristic is to connect the root node $s \in V$ to the member-nodes in Z through shortest paths. If the root node s is located close to the center of G , a Steiner-MRDL heuristic and a Steiner-MDDL

heuristic would produce trees with similar diameter and radius.

Problem 10 *Bounded radius degree limited Steiner minimum tree problem (BRDLSMT)*: Given G and Z , a root node $s \in V$, a radius bound $R \in \mathbb{R}$. and a degree bound $d(v) \in \mathbb{N}$ for each vertex $v \in V$; Find a Steiner minimum tree T on G , where, for each $z \in Z$ the path $p = (z, \dots, s)$, with weight $\sum_{p_i \in p} c(p_i) \leq R$, and $d_T(v) \leq d(v)$.

Like the BRSMT-algorithm, a BRDLSMT-algorithm builds a tree of minimum total cost within a radius bound from a given source to all destinations. The BRDLSM problem is NP -complete, and solves the stress issues of BRSMT by using degree limits for each vertex. BRDLSMT is approximating a BDDLST when the root node s is close to the center of G . The advantage is that a BRDLSMT-heuristic is often less complex than a BDDLST-heuristic.

5.4 Steiner tree heuristics

Several Steiner tree heuristics have been developed over the years. In this paper we shall investigate three Steiner tree heuristic classes, that was called *path-heuristics*, *tree-heuristics* and *vertex-heuristics* by Winter et al [19]. From the three heuristic classes we, respectively, studied the algorithm details of the three SMT-heuristics *shortest path heuristic* (SPH), *distance network heuristic* (DNH) and *average distance heuristic* (ADH). From these SMT-heuristics, we have devised new Steiner tree heuristic variations that address the diameter and reconfiguration time issues we face in distributed interactive applications.

A Steiner tree path-heuristic starts from a pre-chosen source and includes the member-nodes one by one, until the tree spans all member-nodes. Typically, the tree grows based on the addition of (shortest) paths between member-nodes in the tree and member-nodes not yet in the tree. SPH is an SMT path-heuristic and was suggested by Takahashi and Matsuyama [6]. SPH runs an SPT algorithm $|Z| = p$ times (once for each member-node), and stores the shortest path information. Next, it builds the tree starting from a given source, and for each iteration, it connects a member-node through the minimum cost path to the tree. SPH is implemented by a straight forward modification of Prim's MST algorithm [9]. We can see that the bottleneck of SPH is the determination of the shortest paths. Consequently, the SPH has a time complexity of $O(pn^2)$ ($|V| = n$) since shortest paths from each member-node can be determined by for example Dijkstra's SPT algorithm [9] in $O(n^2)$. SPH is sensitive to the choice of the source node from which the tree is constructed. Variations include running SPH more than once, each time from a different source. These repetitive SPH variations yield better solutions, but the execution time is increased. In this paper we select the source using the group center heuristic,

and run every SPH variation only once.

The Steiner tree tree-heuristics are based on the idea of constructing an initial tree that spans all member-nodes, and then optimize it towards a close-to-optimal SMT. Commonly, a minimum spanning tree variant is used to obtain the initial tree. DNH is an SMT tree-heuristic and was suggested, among others, by Kou, Markowsky and Berman [7]. DNH (like SPH) runs an SPT algorithm for each member node, and stores the shortest path information. Then, it builds a distance network graph using only the member nodes, that is, a complete graph with $|Z| = p$ vertices and edge weights equal to the shortest path lengths. It then replaces the individual edges in the distance network graph with the original paths in the input graph. Finally, an MST is run from a given source to find the SMT. As in SPH, the bottleneck of DNH is the determination of the shortest paths. Consequently, the time complexity of DNH is also $O(pn^2)$. The error bound for DNH is the same as SPH, but, SPH often produces better solutions [27]. Moreover, it has been shown that the time complexity of DNH can be reduced by growing shortest path trees from each member-node simultaneously. However, the cost is more complex data structures. In this paper we do not use an optimized DNH.

The general idea behind Steiner tree vertex-heuristics is to identify "good" non-member-nodes (Steiner points). It has been shown [19] that one big difficulty of the SMT problem is to identify non-member-nodes that belong to an SMT. Once the Steiner points are given, the SMT is an MST for the subnetwork induced by the member-nodes and selected Steiner points [19]. ADH is an SMT vertex-heuristic and was suggested by Rayward-Smith [8]. It is based on Kruskal's MST algorithm [9]. The idea is to connect already constructed subtrees of a solution by shortest paths through some centrally located vertex. ADH computes the shortest paths between all pairs of vertices, which is $O(n^3)$ (for example, Floyd Warshall's algorithm [9]), and stores the shortest path information. ADH starts with a forest of trees, where each tree contains only a single member-node. A value $f(v)$ is determined for each $v \in V$, and the vertex m with the smallest f -value is chosen. The m vertex connects a number r of trees that are closest to m , where r is decided when computing the f -value. The worst-case time complexity of ADH is dominated by the computation of all-pairs shortest paths and is $O(n^3)$. On average, ADH does perform better than SPH and DNH in terms of total cost, but at the expense of increased worst-case time complexity.

5.5 Variations of the shortest path heuristic

The path-heuristic variations we have implemented and tested are described below and listed in Table 1. Every heuristic builds the tree incrementally, and

is based on SPH with added functionality such that the diameter is reduced.

- *Minimum diameter shortest path heuristic* (md-SPH) is a heuristic of the Steiner-MDST-problem. In each round, a node is added to the tree that minimizes its eccentricity. md-SPH keeps track of the eccentricities like one-time tree-construction (OTTC) spanning tree algorithm [10]. A similar heuristic was proposed by Brosh et al. [25].
- *Bounded diameter optimized shortest path heuristic* (bdo-SPH) is a heuristic of the Steiner-BDSMT-problem. Each round adds the node that minimizes the total cost of the tree, within a given diameter bound. bdo-SPH keeps track of the eccentricities like OTTC [10]. A similar heuristic was proposed by Aggarwal et al. [23].
- *Bounded diameter randomized shortest path heuristic* (bdr-SPH) is a heuristic of the Steiner-BDSMT-problem. In each round, a member-node is randomly selected and added to the tree through the minimum cost edge within a given diameter bound. bdr-SPH is an adaptation of randomized greedy heuristic (RGH) [10] to the Steiner-BDSMT-problem.
- *Minimum diameter degree limited shortest path heuristic* (mddl-SPH) is a heuristic of the Steiner-MDDL-problem. mddl-SPH is identical to md-SPH but obeys the degree limits on each node much like mddl-OTTC [28].
- *Bounded diameter degree limited optimized shortest path heuristic* (bddlo-SPH) is a heuristic of the Steiner-BDDLSMT-problem. bddlo-SPH is identical to bdo-SPH but obeys the degree limits on each node much like dl-OTTC [28].
- *Bounded diameter degree limited randomized shortest path heuristic* (bddlr-SPH) is a heuristic of the Steiner-BDDLSMT-problem. bddlr-SPH is identical to bdr-SPH but obeys the degree limits on each node much like dl-RGH [28].

5.6 Variations of the distance network heuristic

The tree-heuristics we implemented and tested are described below and listed in Table 1. Every heuristic is based on the idea of DNH, and the only difference is the spanning tree algorithm used to create the final tree on the distance network graph. DNH uses MST, but we test many different spanning tree algorithms found in the literature.

- *Minimum diameter distance network heuristic* (md-DNH) uses md-OTTC [28] to create the final tree on the distance network graph.
- *Bounded diameter optimized distance network heuristic* (bdo-DNH) uses OTTC [10] to create the final tree on the distance network graph.
- *Bounded diameter randomized distance network heuristic* (bdr-DNH) uses RGH [10] to create the final tree on the distance network graph.

- *Minimum diameter degree limited distance network heuristic* (mddl-DNH) uses mddl-OTTC [28] to create the final tree on the distance network graph.
- *Bounded diameter degree limited optimized distance network heuristic* (bddlo-DNH) uses dl-OTTC [28] to create the final tree on the distance network graph.
- *Bounded diameter degree limited randomized distance network heuristic* (bddlr-DNH) uses dl-RGH [10] to create the final tree on the distance network graph.
- *Degree-limited distance network heuristic* (dl-DNH) uses dl-MST to create the final tree on the distance network graph.

5.7 Variations of spanning tree algorithms

A spanning tree algorithm spans every node in the input graph in its tree. Every spanning tree algorithm introduced below have been experimentally tested by the authors in [28]. In which they were found to be the best among 12 different spanning tree algorithms that optimize the diameter. We apply these degree limited spanning tree algorithms to see the effect of adding Steiner points to the input graph.

- *Steiner degree limited one-time tree construction* (sdl-OTTC) is a spanning tree heuristic that addresses the BDDLSMT problem in this paper. sdl-OTTC is exactly dl-OTTC presented in [28] which is a modified OTTC [10].
- *Steiner degree limited randomized greedy heuristic* (sdl-RGH) is a spanning tree heuristic that addresses the BDDLSMT problem in this paper. sdl-RGH is exactly dl-RGH prestended in [28] which is a modified RGH [30].
- *Steiner minimum diameter degree-limited one-time tree construction* (smddl-OTTC) is a spanning tree heuristic that addresses the Steiner-MDDL problem in this paper. smddl-OTTC is exactly mddl-OTTC presented in [28] which is a modified OTTC [10].
- *Steiner degree-limited shortest-path tree* (sdl-SPT) [29] is a spanning tree heuristic that addresses the Steiner-MRDL problem in this paper.

<i>Algorithm</i>	<i>Meaning</i>	<i>Optimize</i>	<i>Algorithm basics</i>	<i>MN¹-aware</i>	<i>Constraint</i>	<i>Complex.</i>	<i>Problem</i>	<i>Ref.</i>
SPH	Shortest Path heuristic	total cost	Prim's MST	✓	-	$O(pn^2)$	1) SMT	[6]
DNH	Distance network heuristic	total cost	Prim's MST	✓	-	$O(pn^2)$	1) SMT	[7]
ADH	Average distance heuristic	total cost	Kruskal's MST	✓	-	$O(n^3)$	1) SMT	[8]
dl-SPH	Degree limited SPH	total cost	Prim's MST	✓	degree	$O(pn^2)$	2) <i>d</i> -SMT	-
dl-DNH	Degree limited DNH	total cost	Prim's MST	✓	degree	$O(pn^2)$	2) <i>d</i> -SMT	-
md-SPH	Minimum diameter SPH	diameter	md-OTTTC	✓	-	$O(n^3)$	3) Steiner-MDST	-
md-DNH	Minimum diameter DNH	diameter	md-OTTTC	✓	-	$O(n^3)$	3) Steiner-MDST	-
bdo-SPH	Bounded diameter optimized SPH	total cost	OTTTC	✓	diameter	$O(n^3)$	4) BDSMT	-
bdo-DNH	Bounded diameter optimized DNH	total cost	OTTTC	✓	diameter	$O(n^3)$	4) BDSMT	-
bdr-SPH	Bounded diameter randomized SPH	total cost	RGH	✓	diameter	$O(pn^2)$	4) BDSMT	-
bdr-DNH	Bounded diameter randomized DNH	total cost	RGH	✓	diameter	$O(pn^2)$	4) BDSMT	-
mddl-SPH	Minimum diameter degree-limited SPH	diameter	mddl-OTTTC	✓	degree	$O(n^3)$	5) Steiner-MDDL	-
mddl-DNH	Minimum diameter degree-limited DNH	diameter	mddl-OTTTC	✓	degree	$O(n^3)$	5) Steiner-MDDL	-
smddl-OTTTC	Steiner minimum diameter degree-limited OTTTC	diameter	Prim's MST	×	degree	$O(n^3)$	5) Steiner-MDDL	[28]
bddlo-SPH	Bounded diameter degree-limited optimized SPH	total cost	dl-OTTTC	✓	diam./degree	$O(n^3)$	6) BDDLSMT	-
bddlo-DNH	Bounded diameter degree-limited optimized DNH	total cost	dl-OTTTC	✓	diam./degree	$O(n^3)$	6) BDDLSMT	-
bddlr-SPH	Bounded diameter degree-limited randomized SPH	total cost	dl-RGH	✓	diam./degree	$O(pn^2)$	6) BDDLSMT	-
bddlr-DNH	Bounded diameter degree-limited randomized DNH	total cost	dl-RGH	✓	diam./degree	$O(n^3)$	6) BDDLSMT	-
sdl-OTTTC	Steiner degree-limited OTTTC	total cost	Prim's MST	×	diam./degree	$O(n^3)$	6) BDDLSMT	[28]
sdl-RGH	Steiner degree-limited RGH	total cost	Prim's MST	×	diam./degree	$O(n^2)$	6) BDDLSMT	[28]
s-SPT	Steiner Dijkstra's SPT	src eccentric.	Dijkstra's SPT	×	-	$O(n^2)$	7) Steiner-MRST	[9]
br-SPH	Bounded radius SPH	total cost	Prim's MST	✓	radius	$O(n^3)$	8) BRST	-
sdl-SPT	Steiner degree-limited Dijkstra's SPT	src eccentric.	Dijkstra's SPT	×	degree	$O(n^2)$	9) Steiner-MRDLST	[29]
brdl-SPH	Bounded radius degree-limited SPH	total cost	Prim's MST	✓	radius/degree	$O(n^3)$	10) BRDLST	-

¹ Steiner tree heuristics are member node (MN) aware.

Table 1. Tree algorithms.

6 Experiments

The observations made in the previous sections form the foundation of our experiments using graph algorithms to reduce the diameter.

6.1 Simulator

We implemented all algorithms presented in sections 4 and 5 in a simulator for application layer multicast. It mimics group communication in a distributed interactive application using a preselected central entity to handle the membership management. The central entity is always selected using the topological center heuristic.

In our experiments, we assume that some *identification* process in the central entity identifies a fully meshed graph where all edges have an associated weight. For this, we used the BRITE [31] topology generator to generate Internet-like router networks. We simulated an application layer overlay network, so the network graph was translated into an undirected fully-meshed shortest-path graph, where each router had one client associated to it. Furthermore, the central entity dynamically divides the users into groups such that each group has a fully meshed group graph. We here present results from simulations using networks with 1000 nodes. The network layout is a square world with sides equal to 100 milli-seconds.

The optional *manipulation* techniques (core-selection heuristics and edge-pruning algorithms) use the fully meshed group graph as input and create a new group graph. All the nodes join and leave groups throughout the simulation, causing group membership to be dynamic. When a join or leave request is received by the central entity, the *construction* process chooses a tree algorithm, and with the latest available group graph given as input, it constructs a new group tree. The tree algorithms that are considered in this paper rebuild the entire tree for every join and leave request. The group popularity is distributed according to a Zipf distribution. We assume that the central entity has the latest member view, and that it has full knowledge of the network. Furthermore, every tree algorithm starts building the tree from a source node. If not otherwise noted, we use the group center heuristic to select a source node. Further experiment parameters are listed in table 2.

<i>Description</i>	<i>Parameter</i>
Placement grid	100x100 milli-seconds
Number of nodes in the network	1000
Number of core nodes	100
Degree limits	3,5 and 10
Degree limits core nodes	degree limit * 2
Diameter bound	250 milli-seconds
Core node set size	group size/degree limit * 2

Table 2
Experiment configuration.

6.2 Target metrics

It is widely acknowledged that tree structures are suitable for event and content distribution. The main advantage of tree structures is that network resources are saved, the main disadvantage is that the pair-wise latency between clients does increase. We investigate the *diameter*, which expresses the worst-case latency between any pair of group members. The *reconfiguration time* of an algorithm is the time that is required to execute a group membership change. It must be low such that a centralized group manager can handle group dynamics and reconfigure a group tree quickly. In addition, we address client side stress issues, which in our graph theory approach is the *degree* in a constructed tree. We investigate graph algorithms that can limit the degree of clients in a tree structure.

6.3 Algorithm constraints

Algorithms that take several target metrics into account often do this by choosing one metric as its optimization goal, and then address the remaining metrics by adding constraints. In general, adding constraints to an algorithm increases the algorithm complexity if an optimal solution is targeted. Many constrained tree heuristics cannot guarantee that a constrained tree is found. That is also the case with the constrained tree heuristics in this paper. The *success rate* of an algorithm depends on the constraint and the input graph. For example, it is more difficult to find a degree limited tree in a *sparse* graph than in a *dense* one.

Figure 4 plots the *success rates* of selected Steiner tree heuristics given a fully meshed graph. The heuristics are subject to varying degree limits (dl) and diameter bounds (db). Heuristics with degree limits as only constraint, always find a tree when given a fully meshed graph and a degree limit > 1 . A tree has a minimum of two leaf nodes at all times, in case of the tree being a snake. Therefore, in a fully meshed graph, the leaf node always has an available degree

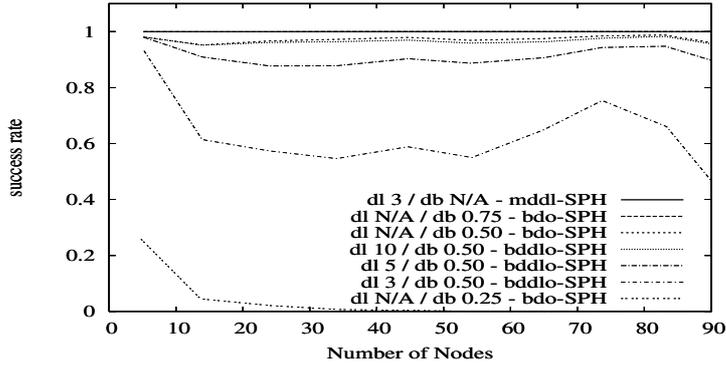


Fig. 4. Success rate of constrained algorithms subject to varying degree limits and diameter bounds.

and a link to all other member-nodes, such that it can continue building the tree. As expected, we see that the success rate for the Steiner-MDDL heuristic mddl-SPH is 100 % for degree limit 3.

Heuristics with diameter bounds do face situations in which it is impossible to find a tree within the diameter bound, even when given a fully meshed graph. The worst-case edge in a fully meshed graph is an approximation to the lowest diameter bound possible for any diameter bounded heuristic. We can see from figure 4 that when the diameter bound is 0.250 second the success rate of the BDSMT heuristic bdo-SPH is pretty much 0 % for group sizes above 40. While, a diameter bound of 0.750 gives a success rate of 100 % for the same heuristic. When degree limits are added to the diameter bounds, we expect the success rate to drop. We see that for the BDDLST heuristic bddlo-SPH, a degree limit of 10 and a diameter bound of 0.750 still gives a success rate of 100 %. However, when the diameter bound is dropped to 0.500 second failures do occur, and the success rate is on average 95 %. When degree limits are added the success rate continues to drop.

To summarize, the Steiner-MDDL heuristics always find a tree when given a full mesh. But, the diameter bounded heuristics fail when given a diameter bound that is less than the worst-case edge in the input graph. For diameter bounded and degree limited heuristics it is even more difficult to find a constrained tree. When a heuristic fails to find a constrained tree, the options are to i) rebuild the tree from scratch with relaxed constraints, ii) abandon the constraints and add the remaining member-nodes through some shortest paths, or iii) relax the constraints dynamically while building the tree.

In our application scenario it is not an option to rebuild the tree from scratch, as it may potentially take a very long time. Furthermore, we do not want to completely abandon the constraints, because they are among our target metrics. Rather, we relax the constraints dynamically whenever a heuristic cannot continue the tree construction process. A low diameter is a target

metric, such that in our simulations we use a strict diameter bound of 0.250 to the diameter bounded algorithms. The heuristics are then frequently forced to relax the bound.

When a degree unlimited Steiner-MDST algorithm is applied to a full mesh made of shortest paths, the algorithm includes at most one Steiner point in the Steiner-tree, i.e., the Steiner point that is closest to the center. However, for a Steiner-MDDL algorithm the number of Steiner points added to the tree depend on the degree limits. Hence, given a full mesh of shortest paths, it is enough for a Steiner-MDST algorithm to find the one node that is closest to the center and connect it to the remaining member-nodes. While, for a Steiner-MDDL algorithm the number of Steiner points that is included in the input graph is a function of the degree limits.

The *aCL* and *aCLO* algorithms use a core node set O of Steiner points. They are added to the input graph to force the Steiner tree algorithms to reduce the diameter. We calculate the size of the core node set (O) using the degree limit d in the current experiment: $|O| = |V|/d * 2$. The function approximates the number of core nodes that is needed to ensure that the degree-limited tree algorithms are able to build a tree.

6.4 Steiner tree heuristic limitations on a full mesh

In a full mesh that is built as a shortest path graph, the shortest paths in the graph are all direct links. Running SPH and DNH on a full mesh of shortest paths has a major impact on their performance. Both heuristics run Dijkstra's SPT for each member node ($z \in Z$) and use the shortest path information when they build a tree. The only way of including a Steiner point is if the shortest path information contains one. In a full mesh built of shortest paths it is impossible for the shortest path information to contain anything other than a single hop, as long as ties are broken to the smaller number of hops. Therefore, SPH and DNH *fail* as Steiner tree heuristics when the input graph is a full mesh built of shortest paths. The shortest path information is available in $O(1)$, and removes the p SPT computations in SPH and DNH. Further, the final trees in both SPH and DNH are built exactly like Prim's MST, which is $O(n^2)$. Hence, in a full mesh SPH and DNH are reduced to an MST algorithm. In fact, any Steiner tree path-heuristic and tree-heuristic that solely uses shortest path information fails to be Steiner point aware when the input graph is a full mesh built of shortest paths.

From these observation we can deduce that it is *incomplete* to only consider shortest paths in a fully meshed shortest path graph. Rather, a Steiner tree heuristic must consider triangulation properties when building the Steiner

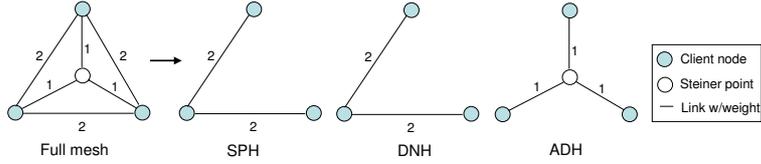


Fig. 5. Full mesh as input to SPH, DNH and ADH.

tree, such that it is possible to add a Steiner point. The vertex-heuristics, for example ADH, typically consider the location of each node in the graph, including the Steiner points, in relation to other member-nodes. That way, centrally located Steiner points may be included in a tree. When ADH is run on a full mesh built of shortest paths the computation of all-pairs shortest paths ($O(n^3)$) is avoided. Therefore, the worst-case time complexity of ADH is reduced to $O(n^2 * \log(n))$. Figure 5 illustrates an example in which SPH and DNH find the MST and ADH the SMT.

Figure 6 plots the average number of Steiner points in the group trees when a fully meshed graph is input. We found that no Steiner tree heuristic derived from SPH and DNH finds a Steiner point to include in the tree. We can conclude that our previous observations are correct, and that it is useless to apply Steiner tree heuristics derived from SPH and DNH to a fully meshed graph built from shortest paths. The only Steiner tree heuristic that finds Steiner points is ADH. As previously described, ADH optimizes for the total cost and is based on Kruskal’s MST algorithm. Tree algorithms that are based on Kruskal’s MST start with a forest of trees and connect them until there is only one left. For algorithms that aim at a low or bounded diameter it is not possible to use algorithms based on Kruskal’s MST unless there is some global knowledge, or some reference points between the subtrees while they are built. Every diameter algorithm the authors of this paper are aware of starts from a given source and builds one tree sequentially using data structures that update the current diameter, radius and/or eccentricities. Based on these observations we conclude that ADH can not be adapted to reduce the diameter in its current form. The other algorithm that includes Steiner points to the tree is sdl-SPT, which is merely a dl-SPT algorithm that uses an input graph that includes Steiner points. Therefore, one approach may be to use conventional spanning tree algorithms, add Steiner points to the input graph and remove the Steiner points with degree one (leaves) from the tree. Table 3 summarizes our performance observations regarding SPH, DNH and ADH on a full mesh built of shortest paths compared to a reduced mesh.

6.5 Fully meshed results

We have seen that the diameter heuristics derived from SPH and DNH cannot include Steiner points, hence they cannot perform better than the spanning

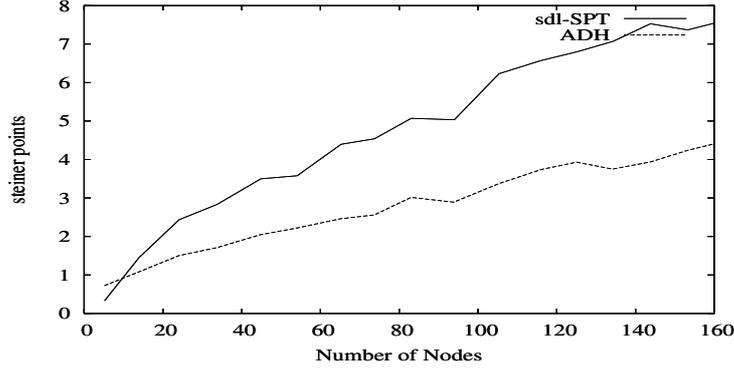


Fig. 6. Average number of Steiner points in the group trees when a fully meshed graph is input (degree limit=10).

Algorithm	Full mesh		Reduced mesh	
	Complexity	Performance	Complexity	Performance
SPH	$O(n^2)$	$\Omega(\text{MST})$	$O(pn^2)$	$\Omega(\text{SMT})$
DNH	$O(n^2)$	$\Omega(\text{MST})$	$O(pn^2)$	$\Omega(\text{SMT})$
ADH	$O(n^2 * \log(n))$	$\Omega(\text{SMT})$	$O(n^3)$	$\Omega(\text{SMT})$

Table 3
Algorithm performance.

tree algorithms they use as algorithm base. In the following, we present results from some selected spanning tree algorithms that are given a fully meshed input graph. Figure 7 plots the diameter of trees using sdl-SPT and dl-SPT with varying degree limits. The only difference is the input graphs, where sdl-SPT was given a number of Steiner points. We can see that sdl-SPT performs better than dl-SPT. The reasons are that, first of all, in our experiments the Steiner points that are added to the input graph are selected using the group center heuristic. Hence, if a Steiner point is added to the tree it is most likely located somewhere in the group center. Secondly, the added Steiner points increases the degree capacity centrally. Finally, the combination of Steiner point location and capacity helps sdl-SPT to create trees with lower diameter than the dl-SPT with no Steiner points. The remaining spanning tree algorithms are plotted in figure 8. We observe the same tendency for all of the spanning tree algorithms. Figure 9 plots the hop-diameter, and we can see that the number of hops does increase slightly as a result of adding Steiner points to the input graph. This is to be expected when more nodes are included in a tree. Figure 10 plots the average number of Steiner points in the trees for sdl-SPT and smddl-OTTC and varying degree limits. The number of Steiner points is quite high when the degree limit is 3, but, as we saw above, is actually reducing the diameter.

The main observation is that the spanning tree algorithms produce trees with lower diameter (seconds) when given a set of Steiner points in the input graph, while the hop-diameter increases slightly. It follows from the geometric version

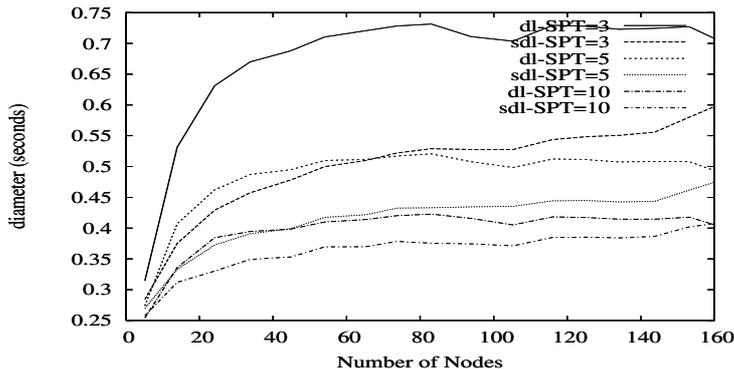


Fig. 7. Diameter (seconds) of sdl-spt and dl-spt.

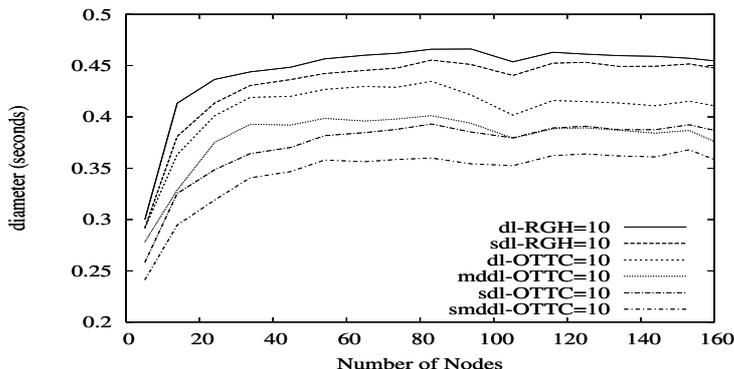


Fig. 8. Diameter (seconds) of spanning tree algorithms with and without Steiner points (degree limit 10).

of the Steiner-MDDL and Steiner-MDST, that if we had an infinite amount of Steiner points located close to the group center the degree-limited heuristics would perform as well as their degree-unlimited versions. In fact, as long as the diameter among the Steiner points is less than the diameter among the member-nodes, new Steiner points can be added without increasing the member-node diameter. Of course, we can not assume that we have that many Steiner points available. It is inevitable that new Steiner tree heuristics must be designed for the degree-limited diameter- and radius-related Steiner tree problems listed in section 5.

Figure 12 plots the *reconfiguration time* of the selected algorithms. The execution times are lowest for sdl-SPT and highest for smddl-OTTC. However, they are all faster than 10 milli-seconds for group sizes up to 120 member-nodes (see figure 11). The added Steiner points do not significantly increase the execution times of the algorithms. Hence, we can add Steiner points without noticeable penalties to our target metrics.

To summarize, the Steiner tree heuristics derived from SPH and DNH should not be used on a full mesh of shortest paths because they do not perform better than any given spanning tree algorithm. The spanning tree algorithms we

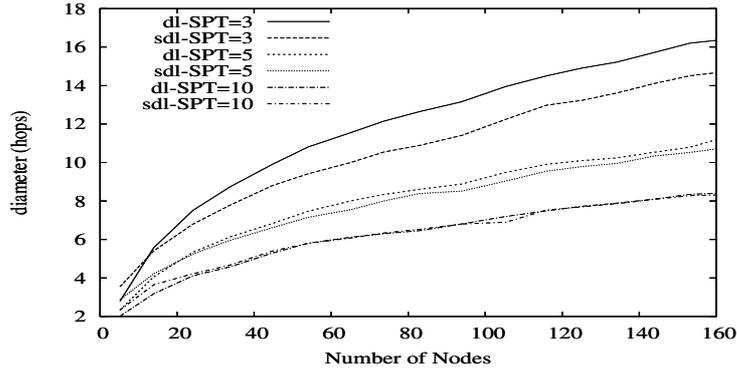


Fig. 9. Diameter (hops) of sdl-spt and dl-spt.

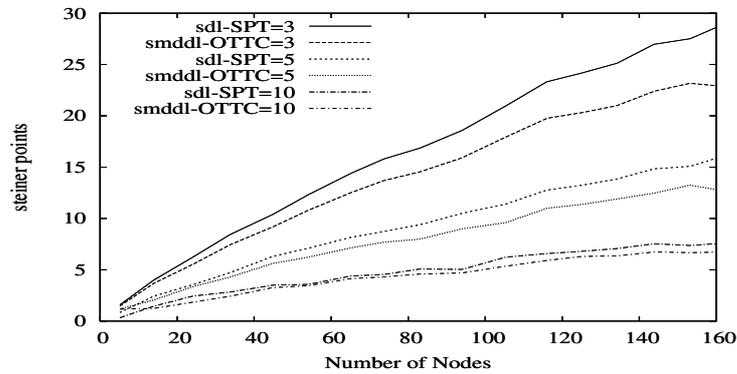


Fig. 10. Average number of Steiner points in trees (degree limits 3, 5 and 10).

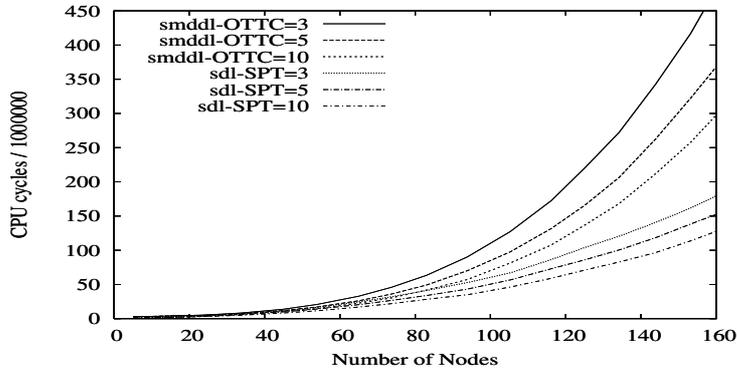


Fig. 11. Execution time of sdl-spt, dl-spt and mddl-OTTC (group size 120).

tested instead achieve smaller diameters with slightly increased hop-diameters when Steiner points are added to the input graph. The main reasons for this improvement are that Steiner points are found using the group center heuristic and that they increase the degree capacity centrally. We also observed that the *reconfiguration time* does not suffer noticeably when the number of edges and nodes in the input graph is increased.

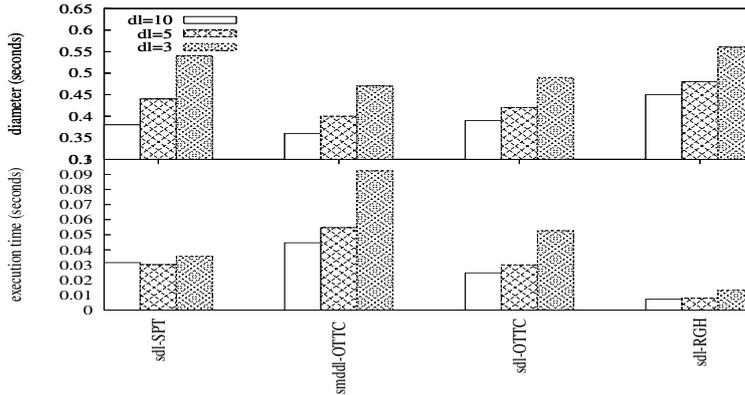


Fig. 12. Diameter and execution time with degree limits 3, 5 and 10.

6.6 Discussions for fully meshed results

The spanning tree algorithms that we tested are evaluated with respect to our target metrics in table 4. *sdl-RGH* is the fastest algorithm, but produces trees with the highest diameter. *smddl-OTTC* and *sdl-OTTC* are similar to each other, but *smddl-OTTC* is slightly slower and does not have the flexibility of a bounded-diameter algorithm. *sdl-SPT* was a surprisingly good alternative. It is a good algorithm for a source-based tree, and when the source is located in the group center *sdl-SPT* builds good shared trees as well.

<i>Algorithm</i>	<i>Diameter</i>	<i>Time</i>	<i>Degree</i>	<i>Rank</i>
<i>sdl-OTTC</i>	+++	++	+	++++
<i>sdl-SPT</i>	++	+++	+	+++
<i>smddl-OTTC</i>	++++	+	+	++
<i>sdl-RGH</i>	+	++++	+	+

Table 4

Tree algorithm characteristics using full mesh.

Our ranking is subjective and not related to specific application needs. All the algorithms fit different needs, and figure 12 shows that they vary in performance between diameter and reconfiguration time. *sdl-RGH* is a fast $O(n^2)$ -heuristic. When extending the tree, it chooses the next vertex at random and connects it via the lowest-weight edge that maintains the diameter constraint. The diameter constraint is only maintained towards the source, and is actually the radius. The algorithm works surprisingly well to produce trees with a relatively small diameter. *sdl-OTTC* extends the tree through the minimum-weight edge that obeys the diameter bound. It is slower than *sdl-RGH* because it performs a more time consuming maintenance of the diameter, but it produces trees with smaller diameter. *smddl-OTTC* always minimizes the maximum diameter, and is therefore even slower. *sdl-SPT* avoids diameter bounds and doesn't minimize the diameter, either. For many applications a bound may not be known, and minimization may not be necessary. *sdl-SPT*

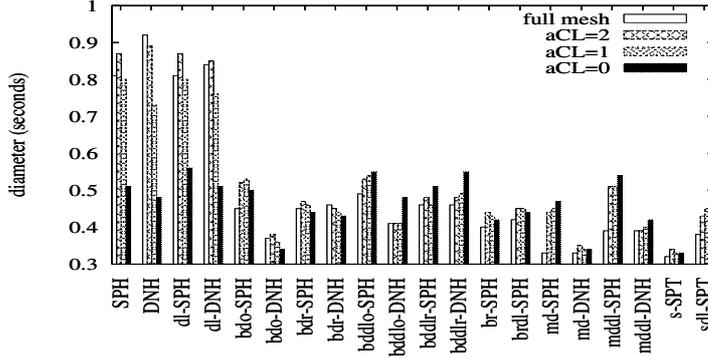


Fig. 13. Diameter (seconds) for aCL .

rather searches for source destination shortest paths, which is often desired by streaming applications.

6.7 Reduced graphs results

We pointed out in section 4 that graph manipulation can save time in the construction. In the following, we present results from combining a core selection heuristic and edge-pruning with tree algorithms. In this section, we use aCL and $aCLO$ ($k=2,1,0$) to reduce the input graph size, and the group center heuristic to find the core nodes, i.e., Steiner points. All the plots use a group size of 120 nodes with a degree limit of 10. We have included results from using a full mesh as a reference point. s-SPT and sdl-SPT are used as the representative spanning tree algorithms.

Figure 14 compares the *diameter* achieved, as applied to a fully meshed graph and aCL . As expected, SPH, DNH, dl-SPH and dl-DNH all produce trees with high diameter because they optimize the total cost. However, we see that for aCL with $k = 0$ all the algorithms produce trees with a lower diameter, and the total cost algorithms are down to a diameter of around 0.500 seconds. The algorithms that produce the lowest diameter are the degree-unlimited algorithms bdo-DNH, md-DNH and s-SPT. Among the degree-limited algorithms bddlo-DNH, bddlr-SPH, mddl-DNH and sdl-SPT produce the lowest diameter trees. Overall, the diameter related heuristics produce the lowest diameter when using a fully meshed input graph. However, when aCL is used, the diameter suffers on average only 15% even when $k = 0$, and the edge set is reduced with 80%, compared to the fully meshed graph.

Figure 14 plots the *maximum degree* in the trees when using aCL to reduce the graph. We observe that the algorithms with a high maximum degree do all produce trees with very low diameter. s-SPT constructs the trees with the lowest diameter. We observe that it consistently constructs trees that re-

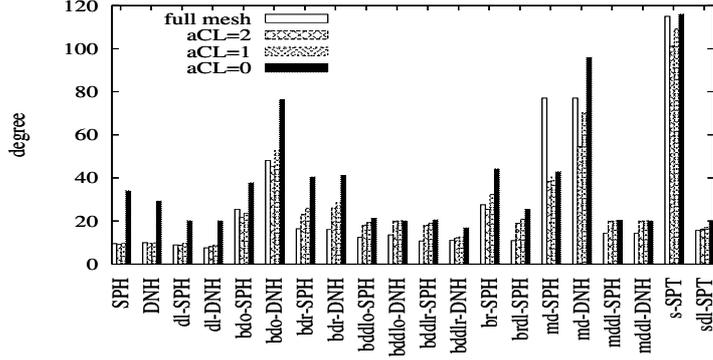


Fig. 14. Maximum degree when using full mesh and aCL .

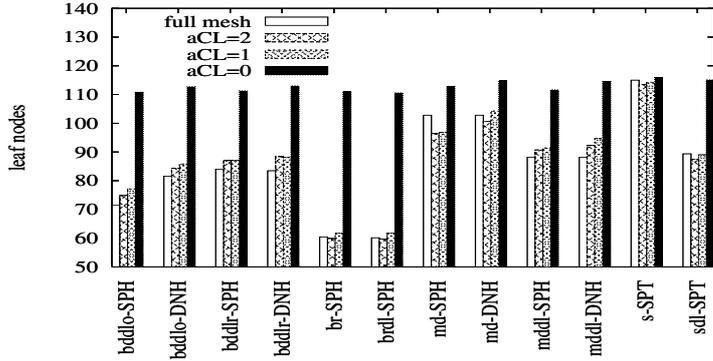


Fig. 15. Average number of leaves (member-nodes) when using aCL .

semble a star because the maximum degree is approximately the group size. Furthermore, the degree-limited algorithms do all construct trees within the maximum degree limit of the core nodes (Steiner points). From these observations, we deduce that a Steiner tree heuristic that optimizes the diameter must be able to exploit the degree capacity of centrally located nodes. However, in our application scenario the degree capacity is often limited, such that the degree-unlimited algorithms with low diameter and high maximum degree are not really an alternative. From figure 15 we see the average number of leaf nodes when using aCL . We observe that that when $k = 0$ every member-node is forced to be a leaf node, regardless of location. Hence, the degree of the member nodes is one, and the stronger core nodes (Steiner points) is higher.

Figure 16 plots the diameter for $aCLO$ as well. We observe that the diameter suffers on average just below 20% when $aCLO$ is used, instead of the full mesh. $aCLO$ with $k = 0$ reduces the edge set by 95 %, and the construction results are still competitive. Furthermore, figure 17 shows the hop-diameter. It is interesting that the hop-diameter is, on average, reduced when $aCLO$ is applied. A low hop-diameter is often desirable in peer-to-peer file sharing applications.

The diameter and the reconfiguration times of the construction algorithms

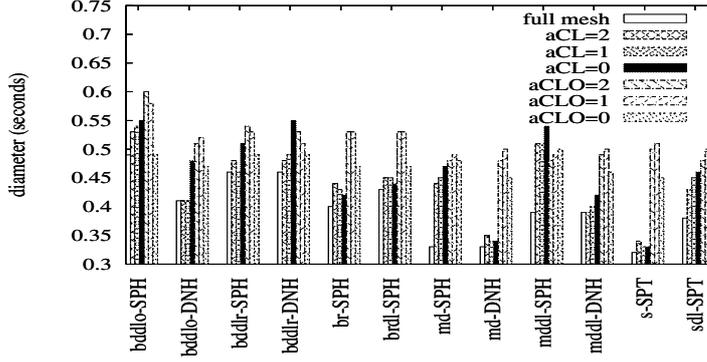


Fig. 16. Diameter (seconds) for aCL and $aCLO$.

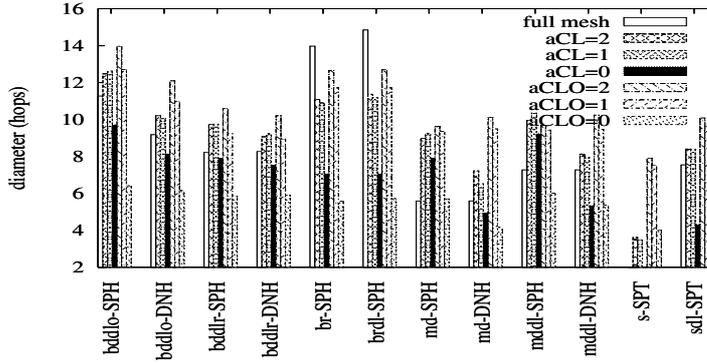


Fig. 17. Diameter (hops) for aCL and $aCLO$.

applied to aCL and $aCLO$ graphs are plotted in figure 18. The time that is consumed by the p SPT computations are highlighted in grey scale for each execution time bar. Generally, the pruning algorithms have a marginally positive effect on the reconfiguration time. As expected, the fastest algorithms are the spanning tree algorithms, here represented by s-SPT and sdl-SPT, both of which execute in $O(n^2)$. Among the Steiner heuristics, the fastest are the randomized heuristics bddlr-SPH and bddlr-DNH that both use dl-RGH as their algorithm base. The reconfiguration times for these randomized SPH and DNH variations are overshadowed by the computation of the p SPTs. The remaining algorithms are comparatively quite slow and all have a complexity of $O(n^3)$. It is interesting that there is no clear correlation between high reconfiguration time and low diameter. However, mddl-DNH does produce the lowest diameter at the price of high execution time.

We expect the *maximum degree* to decrease for the algorithms without degree limits when applying aCL and $aCLO$. In figure 19 we observe that the maximum degree is reduced to about 20 when $aCLO$ is used. Hence, degree-unlimited algorithms are an option for very low bandwidth streams, but only if $aCLO$ and the group-center heuristic are used to manipulate the input graph. However, for all of the degree-unlimited algorithms there are (almost) equally fast algorithm versions with degree limits.

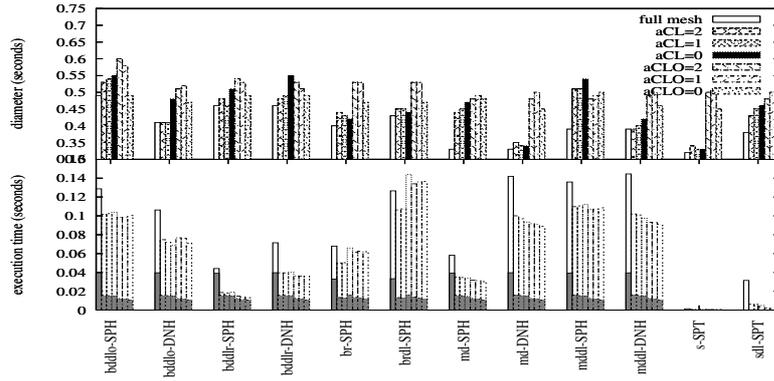


Fig. 18. Diameter and execution time with the overhead of computing the shortest paths (grey scale), for full mesh, aCL and $aCLO$.

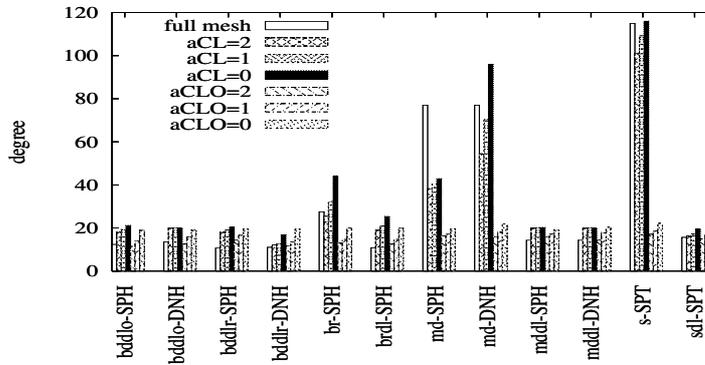


Fig. 19. Maximum degree for full mesh, aCL and $aCLO$.

6.8 Discussions for reduced graphs results

The Steiner tree heuristics are applied to reduced graphs because we want to see the effect on the diameter and reconfiguration time. The reduced graphs are built by combining the group center heuristic, which found the Steiner points, and the pruning algorithms aCL and $aCLO$, which created a reduced group graph. The results from the reduced graphs showed that the SPH and DNH based Steiner tree heuristics cannot outperform the naive approach of running a spanning tree algorithm on the same input graphs (with Steiner points). Among the degree-unlimited algorithms, s-SPT outperformed the minimum-diameter algorithms (md-DNH and md-SPH) both in terms of diameter achieved and reconfiguration time. Furthermore, among the degree-limited algorithms, the sdl-SPT algorithm performed similarly to the minimum-diameter degree-limited algorithm (mddl-DNH and mddl-SPH) but sdl-SPT is much faster.

We expected the reconfiguration times to decrease when aCL and $aCLO$ were used, but the reduction was limited to about 30%. However, if the shortest

paths are precomputed the randomized algorithms based on RGH proved to be very fast. Table 5 gives an overview of the Steiner tree heuristics performances when the shortest path information is available.

<i>Algorithm</i>	<i>Diameter</i>	<i>Time</i>	<i>Degree</i>	<i>Rank</i>
md-SPH	+	-	+	-
bdo-SPH	+	-	+	-
bdr-SPH	+	+	+	+
br-SPH	+	-	+	-
mddl-SPH	+	-	+	-
bddlo-SPH	+	-	+	-
bddlr-SPH	+	+	+	+
brdl-SPH	+	-	+	-
md-DNH	+	-	+	-
bdo-DNH	+	-	+	-
bdr-DNH	+	+	+	+
br-DNH	+	-	+	-
mddl-DNH	+	-	+	-
bddlo-DNH	+	-	+	-
bddlr-DNH	+	+	+	+

Table 5

Steiner tree heuristic characteristics assuming shortest path information.

aCLO reduced the maximum degree and allowed thereby the feasible use of the degree-unlimited algorithms. Table 6 gives an overview. The pruning algorithms bound the stress on the member-nodes. For example, when $k = 0$ for *aCL* and *aCLO*, every group member is a leaf node (degree one), and all the stress is put on the core nodes (Steiner points) that are assumed to have a higher capacity.

6.9 Discussions for all results

A tree algorithm for our construction process should produce trees with low diameter, keep the reconfiguration time fast and be able to obey degree limits. We have seen that the mddl-DNH algorithm produces trees with low diameter within the degree limits. However, the reconfiguration time is very high compared to the simple but efficient sdl-SPT algorithm. Remember, low reconfiguration time is particularly desirable during frequent tree updates, which is often the case for our target applications.

The common denominator for every SPH and DNH based Steiner tree heuristic we tested is the high reconfiguration time, which is largely due to their shortest path computations and the increased complexity of the data structures. For the full mesh, the shortest paths are given but the Steiner tree heuristics failed to include any Steiner points. The Steiner tree heuristics did work when

<i>Algorithm</i>	<i>Diameter</i>	<i>Time</i>	<i>Degree</i>	<i>Rank</i>
md-SPH	+	-	+	-
bdo-SPH	+	-	+	-
bdr-SPH	+	+	+	+
br-SPH	+	-	+	-
mddl-SPH	+	-	+	-
bddlo-SPH	+	-	+	-
bddlr-SPH	+	+	+	+
brdl-SPH	+	-	+	-
md-DNH	+	-	+	-
bdo-DNH	+	-	+	-
bdr-DNH	+	+	+	+
mddl-DNH	+	-	+	-
bddlo-DNH	+	-	+	-
bddlr-DNH	+	+	+	+
s-SPT	+	+	+	+
sdl-SPT	+	+	+	+
sdl-OTTC	+	+	+	+
sdl-RGH	+	+	+	+
smddl-OTTC	+	+	+	+

Table 6

Tree algorithm characteristics assuming shortest path information and using *aCLO*.

applied to reduced graphs, but then the shortest paths must be computed. In a centralized approach to group membership management the central entity has access to the global graph and every group graph. It is therefore possible for the central entity to pre-compute the shortest paths, which would reduce the reconfiguration time for the Steiner tree heuristics quite significantly.

The Steiner tree heuristics in this paper have been derived from the SPH path-heuristic or the DNH tree-heuristic. Similar algorithms addressing the same Steiner tree problems have been derived from both heuristics. Overall, we saw a tendency favoring DNH as the most suitable algorithm base of the two. The DNH-based heuristics produced trees with a lower diameter, whereas the degree and reconfiguration time are all similar to the SPH-based heuristics.

To summarize our observations, none of the SPH and DNH based Steiner tree heuristics in this paper work as Steiner tree heuristics on fully meshed shortest path graphs. We saw that the spanning tree algorithms sdl-SPT and smddl-OTTC are very good alternatives when applied to a full mesh. When the input graphs are reduced using *aCL* and *aCLO*, the DNH-based heuristics performed better than the SPH-based heuristics. Overall, we deduce that DNH is better suited for adaptation to reduce the diameter than SPH. The main drawback of every SPH and DNH based Steiner tree heuristic on a reduced mesh is the necessity of computing the shortest paths and the increased complexity of the data structures that this implies. The shortest path com-

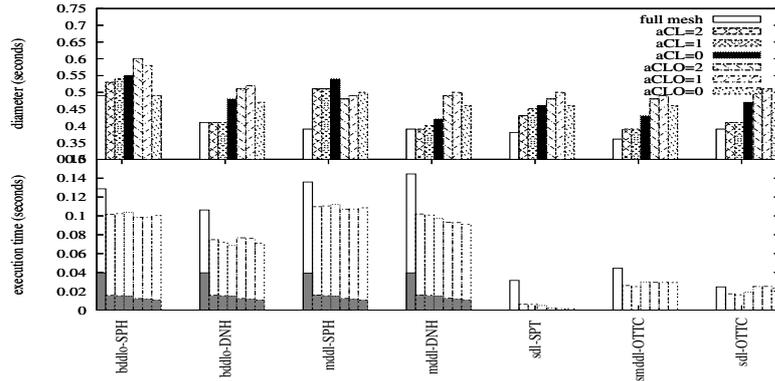


Fig. 20. Diameter and execution time with the overhead of computing the shortest paths (grey scale), for full mesh, aCL and $aCLO$.

putation is time-consuming and increases the reconfiguration time. Figure 20 shows the diameter (seconds) and execution time (seconds) for selected Steiner tree heuristics and spanning tree algorithms. We observe that their diameter is similar, however, the execution times of the Steiner tree heuristics are clearly higher. Our main conclusion is that the Steiner tree heuristics based on SPH and DNH are only suited for our application scenario if the membership dynamics is medium to low, and if the shortest path information is pre-computed and available on the server.

7 Related work

Considerable attention has been given to latency reduction in distributed interactive applications. Research areas such as graph theory (network layout), protocol optimizations (on all layers), group management (distributed and centralized), and multicast protocols are all necessary for the further enhancement of distributed interactive applications. In this paper, our focus has been on using application layer multicast with a centralized approach to group management. The groups are organized in overlay networks that are created using graph algorithms.

Currently, two general approaches are used to accomplish overlay multicast. One is peer-2-peer (P2P) networks that are designed for file and information sharing in highly dynamic networks, for example, BitTorrent and Gnutella [32]. Most P2P applications build overlay networks that ignore the underlying physical topology, which affects the service because the latency can become very high. The second approach focuses on improving overlay multicast protocols and offers more robust group communication. There exist overlay multicast protocols that are topology-aware and are designed to achieve lower latencies and better bandwidth usage [33]. Many overlay multicast protocols have been proposed, but there remains room for improvement, especially regarding the

construction of overlay networks.

Overlay protocols that use distributed hash tables (DHTs) are appropriate for file sharing applications. However, DHT protocols do not fit to event sharing applications because they do not consider pair-wise latency requirements. Hence, we do not regard DHT protocols to be appropriate for distributed interactive applications.

The Yoid project [34] provides an architecture for both space- and time-based multicast, and NICE [35] arranges group members into a hierarchy of layers and proposes arrangement and data-forwarding schemes. In Narada [36], end systems organize themselves into an overlay structure using a fully distributed protocol. ALMI [37] is centrally managed application-level group communication middleware, tailored toward the support of relatively small multicast groups with many-to-many semantics.

These proposed protocols are starting points, but they use overlay network construction algorithms that are either shortest-path or minimum-spanning trees. Hence, it is not sufficient to address the latency demands in distributed interactive applications. An exception is AMcast [38], which uses a set of distributed multicast service nodes (MSN). The authors focus on optimizing the access bandwidth of the MSN's interfaces and end-to-end delay, and propose several new tree algorithms, for example, the compact-tree and bounded compact-tree algorithm. We tested both of these algorithms, but found them to be too slow for our applications. Furthermore, the MSN placement problem is strongly linked to selecting core nodes using a core selection heuristic.

Furthermore, few, if any, protocols are able to maintain subsets of a larger set of nodes. An approach that looks at the maintenance of subgroups within a larger set of overlay nodes is PartyPeer [39]. This system creates subgroups by forming overlay multicast groups as subtrees of a tree that covers the entire set. However, the approach taken results in poorer performance, because subgroups are always created as subtrees of a single tree for the entire application.

In summary, there is a considerable body of work on overlay multicast protocols and efficient tree construction and maintenance. However, current approaches do not address frequent group membership changes and resource limitations of a node (degree) while at the same time minimizing the diameter for latency-bound communication.

8 Conclusions and future work

We have investigated group communication in relation to distributed interactive applications. Our investigation involved experiments with many Steiner tree heuristics, where our main target metric was a low tree diameter. We applied the heuristics to an application layer overlay network where the network is always a full mesh. This should give a tree algorithm the optimal conditions for finding the best tree. However, we observed that the Steiner tree heuristics that should be aware of Steiner points and add the ones that help optimize the tree, failed to do so on a full mesh made of shortest paths. We then tested the spanning tree algorithms *sdl-RGH*, *sdl-OTTC*, *smddl-OTTC* and *sdl-SPT* using input graphs that included Steiner points. The results showed that these algorithms produced trees with a smaller diameter when Steiner points were included. Moreover, the heuristics are fast, which is important in highly dynamic distributed applications. However, a Steiner-heuristic that uses a spanning tree algorithm and then prunes leaf Steiner points is a naive heuristic and a simplistic approach to addressing the diameter related Steiner tree problems. Better Steiner tree heuristics should be designed that work on fully meshed graphs.

In addition, we investigated algorithms for reducing the time it takes to execute membership changes. We found that the group center heuristic, and the edge-pruning algorithms *aCL* and *aCLO* are powerful means to manipulate the input graph. However, the Steiner tree heuristics only reduced their execution time slightly even though the edge set was reduced by 95 % in the most extreme case. But, every Steiner tree heuristic and spanning tree algorithm still performed comparatively well in terms of the diameter.

In our simulations, the node layout was a square world with sides equal to 100 milli-seconds (approximately Europe). The diameter that the best Steiner-tree heuristics yielded was below 400 milli-seconds for tree sizes up to 160 and a degree-limit of 10. This is outside the requirements for first-person shooter games, but is just inside for most distributed interactive applications (see section 2.2). When the degree limit is less than or equal to 5 the latency requirements are not met. A degree limit above 5 is not a problem for multi-player online games, because the data streams are very thin [12]. However, for video/audio conferences it may be an issue due to somewhat limited bandwidth capacity on average clients in the Internet.

Currently, we are investigating dynamic tree algorithms that insert and remove single nodes to reduce the reconfiguration time further [40]. In that respect, we plan to test our algorithms in a network simulator that we have implemented. Finally, we intend to test distributed alternatives to the centralized algorithms.

References

- [1] The Entertainment Software Association, ESAs 2006 essential facts about the computer and video game industry (January 2008).
URL <http://www.theesa.com/>
- [2] B. S. Woodcock, An analysis of mmog subscription growth (January 2008).
URL <http://www.mmogchart.com>
- [3] M. Claypool, The effect of latency on user performance in real-time strategy games, *Elsevier Computer Networks* 49 (1) (2005) 52–70.
- [4] M. Claypool, K. Claypool, Latency and player actions in online games, *Communications of the ACM* 49 (11) (2005) 40–45.
- [5] F. Dabek, R. Cox, F. Kaashoek, R. Morris, Vivaldi: a decentralized network coordinate system, in: *ACM International Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, 2004, pp. 15–26.
- [6] H. Takahashi, A. Matsuyama, An approximate solution for the Steiner problem in graphs, *Math. Japonica* 24 24 (6) 573–577.
- [7] L. Kou, G. Markowsky, L. Berman, A fast algorithm for Steiner trees, *Acta Informatica* 15 (1981) 141–145.
- [8] V. Rayward-Smith, A. Clare, The computation of nearly minimal Steiner trees in graphs, *International Journal of Mathematical Education in Science and Technology* 14 (1) (1983) 8pp.
- [9] M. Goodrich, R. Tamassia, *Algorithm Design: Foundations, Analysis and Internet Examples*, John Wiley and Sons, 2002.
- [10] A. Abdalla, N. Deo, P. Gupta, Random-tree diameter and the diameter-constrained MST, Tech. Rep. CS-TR-00-02, University of Central Florida, Orlando, FL, USA (2000).
URL <http://citeseer.ist.psu.edu/abdalla00randomtree.html>
- [11] International Telecommunication Union (ITU-T), One-way Transmission Time, ITU-T Recommendation G.114 (2003).
- [12] C. Griwodz, P. Halvorsen, The fun of using TCP for an MMORPG, in: *International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, ACM Press, 2006, pp. 1–7.
- [13] G. Kortsarz, D. Peleg, Approximating shallow-light trees, in: *SODA '97: Proceedings of the eighth annual ACM-SIAM symposium on Discrete algorithms*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1997, pp. 103–110.
- [14] A. Karaman, H. S. Hassanein, Core-selection algorithms in multicast routing - comparative and complexity analysis., *Computer Communications* 29 (8) (2006) 998–1014.

- [15] B. Y. Wu, K.-M. Chao, *Spanning Trees and Optimization Problems*, Chapman and Hall/CRC, 2004.
- [16] O. R. Oellermann, On steiner centers and steiner medians of graphs, *Networks* 34 (4) (1999) 258–263.
- [17] K.-H. Vik, C. Griwodz, P. Halvorsen, Applicability of group communication for increased scalability in MMOGs, in: *Workshop on Network and System Support for Games (NETGAMES)*, ACM Press, Singapore, 2006.
- [18] A. Young, C. Jiang, M. Zheng, A. Krishnamurthy, L. Peterson, R. Wang, Overlay mesh construction using interleaved spanning trees (Mar. 2004).
- [19] P. Winter, Steiner problem in networks: a survey, *Netw.* 17 (2) (1987) 129–167.
- [20] S. Voss, Steiner’s problem in graphs: heuristic methods, *Discrete Appl. Math.* 40 (1) (1992) 45–72.
- [21] F. Bauer, A. Varma, ARIES: A rearrangeable inexpensive edge-based online Steiner algorithm, in: *Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, 1996.
- [22] R. Hassin, A. Levin, Minimum restricted diameter spanning trees, *Discrete Appl. Math.* 137 (3) (2004) 343–357.
- [23] S. Aggarwal, M. Limaye, A. Netravali, K. Sabnani, Constrained diameter steiner trees for multicast conferences in overlay networks, in: *QSHINE ’04: Proceedings of the First International Conference on Quality of Service in Heterogeneous Wired/Wireless Networks (QSHINE’04)*, IEEE Computer Society, Washington, DC, USA, 2004, pp. 262–271.
- [24] Q. Zhu, W. Dai, Delay bounded minimum steiner tree algorithms for performance-driven routing (1993).
URL citeseer.ist.psu.edu/zhu93delay.html
- [25] E. Brosh, Approximation and heuristic algorithms for minimum delay application-layer multicast trees (2003).
URL citeseer.ist.psu.edu/article/brosh04approximation.html
- [26] J. M. Ho, D. T. Lee, C. H. Chang, C. K. Wong, Bounded diameter minimum spanning trees and related problems, in: *SCG ’89: Proceedings of the fifth annual symposium on Computational geometry*, ACM Press, New York, NY, USA, 1989, pp. 276–282.
- [27] B. M. Waxman, Dynamic Steiner tree problem, *SIAM J. Discrete Math.* 4 (1991) 364–384.
- [28] K.-H. Vik, P. Halvorsen, C. Griwodz, Multicast tree diameter for dynamic distributed interactive applications, in: *To appear in INFOCOM*, 2008.
- [29] S. C. Narula, C. A. Ho, Degree-constrained minimum spanning trees, *Computers and Operations Research* 7 (1980) 239–249.

- [30] G. R. Raidl, B. A. Julstrom, Greedy heuristics and an evolutionary algorithm for the bounded-diameter minimum spanning tree problem, in: SAC '03: Proceedings of the 2003 ACM symposium on Applied computing, 2003, pp. 747–752.
- [31] A. Medina, A. Lakhina, I. Matta, J. Byers, BRITE: Universal topology generation from a user’s perspective, Tech. Rep. BUCS-TR-2001-003, Computer Science Department, Boston University (Apr. 2001).
URL citeseer.ist.psu.edu/article/medina01brite.html
- [32] Peer to peer information from wikipedia.
URL <http://en.wikipedia.org/wiki/Peer-to-peer>
- [33] M. Kwon, S. Fahmy, Topology-aware overlay networks for group communication (2002).
URL citeseer.ist.psu.edu/kwon02topologyaware.html
- [34] P. Francis, S. Ratnasamy, R. Govindan, C. Alaettinoglu, Yoid project (2000).
URL www.icir.org/yoid/
- [35] S. Banerjee, B. Bhattacharjee, Analysis of the NICE application layer multicast protocol, Tech. Rep. UMIACSTR 2002-60 and CS-TR 4380, Department of Computer Science, University of Maryland, College Park (Jun. 2002).
- [36] G. Fox, S. Pallickara, The Narada event brokering system: Overview and extensions, in: Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA), 2002, pp. 353–359.
- [37] D. Pendarakis, S. Shi, D. Verma, M. Waldvogel, ALMI: An application level multicast infrastructure, in: Proceedings of the USENIX Symposium on Internet Technologies and Systems (USITS), 2001, pp. 49–60.
URL citeseer.ist.psu.edu/pendarakis00almi.html
- [38] S. Y. Shi, J. Turner, M. Waldvogel, Dimensioning server access bandwidth and multicast routing in overlay networks, in: International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV), 2001, pp. 83–92.
- [39] L. S. Liu, R. Zimmermann, Immersive peer-to-peer audio streaming platform for massive online games, in: International Conference (NIME), 2006.
- [40] K.-H. Vik, C. Griwodz, P. Halvorsen, Dynamic group membership management for distributed interactive applications, in: 32nd IEEE Conference on Local Computer Networks (LCN), 2007, pp. 141–148.