Multicast tree diameter for dynamic distributed interactive applications

Knut-Helge Vik
Simula Research Laboratory and
University of Oslo
Oslo, Norway
Email: knuthelv@ifi.uio.no

Pål Halvorsen
Simula Research Laboratory and
University of Oslo
Oslo, Norway
Email: paalh@ifi.uio.no

Carsten Griwodz
Simula Research Laboratory and
University of Oslo
Oslo, Norway
Email: griff@ifi.uio.no

***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 using application-layer multicast with a centralized approach to the group management. The groups are organized in overlay networks that are created using graph algorithms.**

We investigate many spanning tree problems with particular focus on reducing the diameter of a tree, i.e., the maximum pairwise latency between users. 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 group 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 group nodes to the remaining group members, to create new and pruned group graphs, such that, when a tree algorithm is applied to a pruned group graph, it is manipulated into creating trees with a smaller diameter.

We implemented and analyzed experimentally spanning-tree heuristics, core-selection heuristics and edge-pruning algorithms. We found that faster heuristics that do not explicitly optimize the diameter are able to compete with slower heuristics that do optimize it.

I. 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.

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. 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 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 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 prevent 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, 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 needs very small routing tables, which is important in our scenario. We here concentrate particularly on algorithms that minimize the pairwise latency in distribution trees. The maximum pairwise latency of a tree is known as the diameter.

All the algorithms are implemented, and the performance is analyzed using experiments. Our results show that fairly simple tree heuristics still produce trees with a small diameter.

II. Overlay network diameter

The diameter of an overlay network is determined by its layout. We are investigating graph algorithms that design networks in ways that reduce the diameter, in particular spanning tree problems found in the literature. Before we introduce the algorithms, we highlight the importance of limiting the diameter (in overlay networks), and discuss the requirements of some applications that have millions of users.

A. Application requirements

Multi-player online games allow thousands of users to interact concurrently in a persistent virtual environment. Such games often have stringent latency requirements. The characteristics of game traffic have been analyzed several times before, and in [3], the latency requirements were measured to be approximately 100 ms for FPS games, 500 ms for RPGs and 1000 ms for RTS games. In 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. The maximum latency should not exceed 400 ms [9].

On the basis of these observations, we conclude that overlays should be constructed such that the diameter falls within the requirements of a given application. The diameter requirements may vary from strict to loose even within one application. Our goal is to identify graph algorithms that reduce or limit the diameter in distribution 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. We study multiple graph algorithms and observe how the application requirements correlate with the diameter of the resulting tree and the reconfiguration time.

B. Spanning 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: ***Spanning tree problem:*** *Given a connected undirected weighted graph $G = (V, E, c)$, where V is the set of vertices, E is the set of edges, and $c : E \rightarrow R$ is the edge cost function. Find a connected undirected subgraph (tree) $T = (V_T, E_T)$, without cycles, where $V_T = V$.*

Many tree algorithms are spanning tree algorithms and try to build a tree that covers all the vertices. From this basic spanning tree problem numerous others have been derived. Here, we focus on spanning tree problems that (we believe) may reduce the diameter of T. The *diameter* of T is defined as the longest of the (shortest) paths in T among all the pairs of nodes in V. In addition, we study algorithms that optimize for the *total cost*, i.e., the sum of the edge weights in T. Next, we present problem definitions of relevant spanning tree problems that exist in the literature.

C. Spanning tree problems and diameter

The spanning tree problems that reduce the diameter are often NP-complete. 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. We here mention heuristics that all have a worst-case time complexity of $O(n^3)$.

PROBLEM 2: ***Minimum diameter spanning tree problem (MDST):*** *Given G, find a spanning tree T of G such that the maximum weight shortest path $p \in T$, $\sum_{e \in p} W(e)$ is minimized.*

An MDST-algorithm builds a tree of minimum *diameter*, and is solvable in polynomial time. Ho, Lee, Chang and Wong [8] 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* MDST problem. They prove that there is an optimal tree in which either one or two vertices in V are connected to the remaining vertices. The result extends to any complete graph whose edge lengths satisfy the triangle inequality, which holds for overlay networks that are built from shortest path links from the network layer.

Hence, for a complete graph, finding a simple heuristic for building a close-to-optimal MDST reduces to finding a single node located close to the center of the graph that connects to the remaining nodes through shortest paths (direct links). 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 3: ***Bounded diameter minimum-spanning tree problem (BDMST):*** *Given G, and a bound $D > 0$. Find a minimum weight spanning tree T on G, where $\sum_{e \in E_T} c(e)$ is minimized and the diameter of which does not exeed D.*

A BDMST-algorithm builds a tree of minimum *total cost* within a diameter bound. BDMST is an NP-complete problem and examples of heuristics are *one-time tree construction* (OTTC) [1] and *randomized greedy heuristic* (RGH) [17]. An advantage of BDMST over MDST is that it is possible to tune the tree diameter while minimizing the *total cost*. However, one problem with BDMST remains the potentially high node degree of central nodes in the tree when the diameter bound D is stringent.

PROBLEM 4: ***Minimum diameter degree-limited spanning tree problem (MDDL):*** *Given G, a degree bound $d(v) \in N$ for each vertex $v \in V$; find a spanning tree T of G of minimum diameter, subject to the constraint that $d_T(v) \leq d(v)$.*

An MDDL-algorithm builds a tree of minimum diameter while obeying the degree limits. MDDL is NP-Hard, nevertheless, it is relevant because it avoids the problems with stress that beset spanning tree problems that do not have limitations

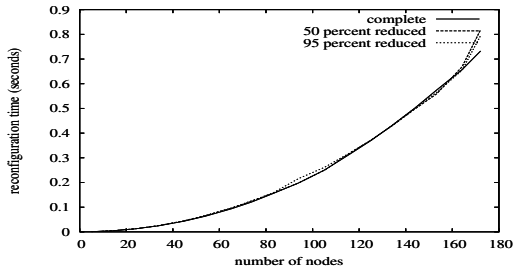

Fig. 1. *Reconfiguration time* (ms) of CT using input graph with different sizes of the edge set.

on degree. One issue with the MDDL problem is that it is a minimization of the maximum diameter within a degree limit, which increases the complexity of a heuristic.

The MDDL-heuristic *compact-tree* (CT) [18] (described in section III-D), is a greedy algorithm based on Prim's minimum-spanning tree algorithm [7]. Figure 1 compares the execution time of CT for a complete graph to some graphs with reduced (pruned) edge sets, running our implementation on an Intel Core 2 Duo machine. We observe that the execution time of CT is independent of the edge-set size, but dependant on the node-set size.

PROBLEM 5: ***Bounded diameter degree-limited minimum-spanning tree problem (BDDLMST):*** *Given G, a diameter bound $D > 0$, and a degree bound $d(v) \in N$ for each vertex $v \in V$; Find a minimum weight spanning tree T on G, where $\sum_{e \in E_T} c(e)$ is minimized, subject to the constraint that the diameter does not exceed D, and $d_T(v) \leq d(v)$.*

The NP-hard BDDLMST problem is identical to BDMST, but with degree limits for each vertex. A BDDLMST-heuristic is able to produce trees with a diameter that is in accordance with the diameter bound it received. This property is vital in cases of lighter application requirements, because the time complexity of the BDDLMST-heuristic decreases with looser diameter bounds. We have implemented degree limited versions of OTTC and RGH (described in section III-D) that are both heuristics of BDDLMST.

PROBLEM 6: ***Limited diameter residual balanced tree problem (LDRB):*** *Given an undirected weighted graph $G = (V, E, c)$, a degree bound $d(v) \in N$ for each vertex $v \in V$, and a diameter bound $D \in R$; Find a spanning tree T of G with diameter $\leq D$ that maximizes $res_T(v) = d(v) - d_T(v)$, subject to the constraint that $d_T(v) \leq d(v)$.*

An LDRB-algorithm builds the "most balanced" tree, that satisfies an upper bound on the diameter. The most balanced tree is any tree that maximizes the smallest *residual degree*. However, a balanced tree does not have an optimal diameter, instead, all nodes suffer. One heuristic of the LDRB-problem is *balanced compact-tree* (BCT) [18] (described in section III-D).

D. Related spanning tree problems

There exist related spanning tree problems that do not explicitly consider the diameter, but are cheaper in terms of the reconfiguration time. For example, a shortest-path tree is a source specific tree in which all nodes have shortest paths to the source. It was solved by Dijkstra and has a worst-case time complexity of $O(n^2)$. A shortest-path tree is actually a simple MDST heuristic if the source vertex is selected on the basis of its location in relation to the other nodes. Remember that in a complete graph, the topology of a close-to-optimal MDST is a star, where the issue was the degree (stress) on the center vertex. Hence, a degree limit is needed when using a shortest-path tree. We here introduce two degree limited spanning problems, with example heuristics which are $O(n^2)$.

PROBLEM 7: ***Degree-limited shortest-path tree problem (d-SPT):*** *Given G, a degree bound $d(v) \in N$ for each vertex $v \in V$; find a spanning tree T, starting from a root node $s \in V$, where, for each $v \in V$ the path $p = (v, \ldots, s)$ minimizes $\sum_{p_i \in p} c(p_i)$. subject to the constraint that $d_T(v) \leq d(v)$.*

A d-SPT algorithm builds a shortest-path tree in which each vertex is subject to a degree limit. We believe that a d-SPT heuristic combined with a carefully selected center (source) node may be able to compete with MDDL heuristics. For example, we found that the MDDL-heuristic CT has a high reconfiguration time, due to a worst-case time complexity of $O(n^3)$. We have implemented a d-SPT heuristic called dl-SPT [15] (described in section III-D).

PROBLEM 8: ***Degree-limited minimum-spanning tree problem (d-MST):*** *Given G, a degree bound $d(v) \in N$ for each vertex $v \in V$; find a spanning tree T, where the $\sum_{e \in E_T} c(e)$ is minimized, subject to the constraint that $d_T(v) \leq d(v)$.*

In addition to studying the d-SPT problem, we investigate the d-MST problem. A d-MST algorithm builds a minimum-spanning tree T in which each vertex is subject to a degree limit. d-MST heuristics found in the literature often employ Prim's MST algorithm. We have implemented a d-MST heuristic that we call dl-MST [15] (described in section III-D).

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

III. CENTRALIZED GROUP MANAGEMENT

As noted above, our focus is on centralized graph algorithms, 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. The users are divided into groups 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. Membership management process

We identify three vital processes for centralized membership management, and, in particular, group updates, that is, when

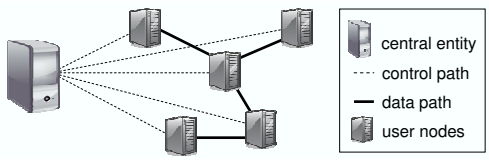

Fig. 2. Centralized group management with control and data paths.

a central entity receives a join or leave request.

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 to change. Here, the techniques include core selection heuristics and prune algorithms (sections III-B and III-C). In the *construction* process, the latest group graph information is used as input to a tree algorithm. The tree algorithm creates/updates the group distribution tree (section III-D).

We here present algorithms related to manipulation and construction. Extensive research that addresses identification may be found in [4]. Identification is independent of manipulation and construction, and its discussion therefore lies outside the scope of this paper.

B. Core selection heuristics

Core-based multicast protocols offer efficient group management capabilities. Efficient (fast) group management is especially important when the groups are dynamic. Core-based multicast protocols select one or more cores as group management and forwarding nodes.

Several core selection heuristics have been proposed, a comprehensive study of which is given by and Karaman and Hassanein [10]. An overall goal is to select cores on the basis of certain node properties, such as, bandwidth and computational power. The number of cores that are selected depends on the group size and the degree limitations in the available core nodes.

In our scenario, core nodes are found among the nodes in a group. By applying core selection heuristics that identify well-placed nodes in a group, we aim to design algorithms that exploit their degree capacity to the fullest 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.

The best location for a core is often related to the location of other group members. The core selection heuristics presented here search for a set of vertices beginning with the *graph median*. 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 [10]:

Topological 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.

C. Prune algorithms

The complexity of an algorithm has a great influence on the execution time, which is important for our target applications. However, it is 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. 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 *reconfiguration time* compared to a graph with a pruned edge set [19].

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 (users) are connected strongly to the remaining nodes. The core nodes are identified using a core selection heuristic. In this paper we present two promising edge-pruning algorithms:

add Core Links (*a*CL) (see algorithm 1) takes as input a fully meshed group graph G and a set $O \subset V$ that contains 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 (kBL) as defined in [21]. 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. *a*CL produces a graph with $|E| = k*|V-O|+|O|*|V|$. After applying *a*CL, the new pruned graph forms, conceptually, a two-layer graph, where the core nodes are fully meshed and the remaining nodes have a degree-limited by $k + |O|$.

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 $v \in V$.

add Core Links Optimized (*a*CLO) (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 *a*CLO.

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 $l \in L$.

D. Tree algorithms

We tested the following 12 tree algorithms that are also listed in table I. All the tree algorithms start building the tree

Algorithm	*Meaning*	*Optimization*	*Constraints*	*Complexity*	*Problem*	*Reference*
MST	Prim's minimum-spanning tree	total cost	-	$O(n^2)$	1) MST	[7]
SPT	Dijkstra's shortest-path tree	core/destination cost	-	$O(n^2)$	1) SPT	[7]
md-OTTC	Minimum diameter one-time tree construction	diameter	-	$O(n^3)$	2) MDST	-
OTTC	One-time tree construction	total cost	diameter	$O(n^3)$	3) BDMST	[1]
RGH	Randomized greedy heuristic	total cost	diameter	$O(n^2)$	3) BDMST	[17]
CT	Compact-tree	diameter	degree	$O(n^3)$	4) MDDL	[18]
mddl-OTTC	Minimum diameter degree-limited one-time tree construction	diameter	degree	$O(n^3)$	4) MDDL	-
dl-OTTC	Degree-limited one-time tree construction	total cost	diameter and degree	$O(n^3)$	5) BDDLMST	-
dl-RGH	Degree-limited randomized greedy heuristic	total cost	diameter and degree	$O(n^2)$	5) BDDLMST	-
BCT	Bounded compact-tree	diameter	diameter and degree	$O(n^3)$	6) LDRB	[18]
dl-SPT	Degree-limited Dijkstra's shortest-path tree	core/destination cost	degree	$O(n^2)$	7) d-SPT	[15]
dl-MST	Degree-limited Prim's minimum-spanning tree	total cost	degree	$O(n^2)$	8) d-MST	[15]

TABLE I
TREE ALGORITHMS.

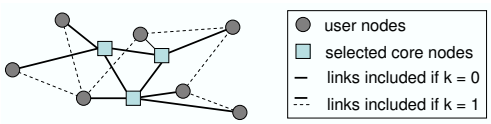

Fig. 3. Pruned graph using *add* Core Links Optimized.

Description	*Parameter*
Placement grid	$1000x1000$ units
Number of nodes in the network	1000
Degree limits	3,5 and 10
Diameter bound	0.25
Core node set size	group size/degree limit $*2$

TABLE II
EXPERIMENT CONFIGURATION.

from a source node. If not otherwise noted, we use the group center heuristic to select a source node.

Minimum-spanning tree (MST) [7] is Prim's minimum-spanning tree algorithm. Prim's MST has been empirically shown to be the fastest MST algorithm for large dense-graphs [14].

Shortest-path tree (SPT) [7] is Dijkstra's shortest-path tree algorithm, which is almost exactly the same as Prim's MST. SPT optimizes to find the shortest-path between a source and any target.

Minimum diameter one-time tree construction (md-OTTC) is a heuristic of the MDST problem. It is an alteration of OTTC. md-OTTC adds the vertex that minimizes the diameter.

One-time tree construction (OTTC) [1] is a heuristic of the BDMST problem. OTTC is a modification of Prim's MST algorithm to accommodate the diameter bound.

Randomized greedy heuristic (RGH) [17] is a heuristic of the BDMST problem for complete graphs, and is based on Prim's MST algorithm.

Compact-tree (CT) [18] is a heuristic of the MDDL problem, and is based on Prim's MST algorithm.

Minimum diameter degree-limited one-time tree construction (mddl-OTTC) is a heuristic of the MDDL problem. It is an alteration of OTTC. mddl-OTTC adds a vertex that minimizes the diameter while obeying the degree limits.

Degree-limited one-time tree construction (dl-OTTC) is a heuristic of the BDDLMST problem. It builds the tree the same way as OTTC, while obeying the degree limits.

Degree-limited randomized greedy heuristic (dl-RGH) is a heuristic of the BDDLMST problem. It builds the tree the same way as RGH, while obeying the degree limits.

Bounded compact-tree (BCT) [18] is a generalization of the CT algorithm. It uses a balancing factor M, and the authors found that $M = 4$ fits best.

Degree-limited shortest-path tree (dl-SPT) [15] is a heuristic of the d-SPT problem. dl-SPT is the same as dl-MST, except that it is optimized for shortest-paths.

Degree-limited minimum-spanning tree (dl-MST) [15] is a heuristic of the d-MST problem. It is a modification of Prim's MST algorithm.

IV. EXPERIMENTS

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

A. Simulator

We implemented all algorithms presented in section III 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 core selection heuristic topological center.

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 [13] topology generator to generate Internet-like router networks. We simulated an overlay network, so the network graph was translated into an undirected fully-meshed shortest-path graph. Furthermore, the central entity 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 optional *manipulation* techniques (core-selection heuristics and edge-pruning algorithms) use the fully meshed

group graph as input and create a new pruned 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, which is given the latest available group graph as input, and 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 was distributed according to a Zipf distribution. We assumed that the central entity has the latest member view, and that it has full knowledge of the network. Further experiment parameters are listed in table II.

B. Metrics and constraints

We considered two metrics to address the application requirements: *diameter* and *reconfiguration time*. The diameter 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. In addition, a degree-unlimited algorithm is not desirable if the constructed tree has a very high maximum degree. Many of the tree algorithms in our investigation use a constraint on the degree-limit.

In general, adding constraints to an algorithm increases the algorithm complexity if an optimal solution is targeted. Many 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 the algorithms 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. We here relax the degree limits whenever a tree heuristic cannot continue the tree construction.

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 still able to build a tree.

C. Fully meshed results

We here present results from using a fully meshed input graph to every tree algorithm. The *diameter* achieved, with degree limit 10, is plotted in figure 4. As expected, MST constructs trees with high diameter, because it optimizes for the *total cost*. dl-MST performs similarly to MST but is not plotted. Hence, we can safely disregard MST and dl-MST. SPT performs best, and constructs trees with a diameter close to 0.3 regardless of group size. The source of SPT is chosen by the group center heuristic, hence, the combination results in an MDST-heuristic (problem 2). We do not plot all the algorithms because many of them construct trees with very similar diameter. In fact, all remaining algorithms construct trees with a diameter between 0.3 and 0.4 seconds.

In figure 5, the diameters with degree limits 3, 5 and 10 are plotted. When the degree limit is 3, the degree-limited algorithms struggle to find trees with a diameter below 0.6 seconds. A degree limit of 5 reduces the diameter to 0.5, while a degree limit of 10 further lowers it to 0.4. We deduce that the degree limit can not be stringent if a low diameter is the desired goal. Thus, henceforth, we use a degree limit of 10 in all our plots.

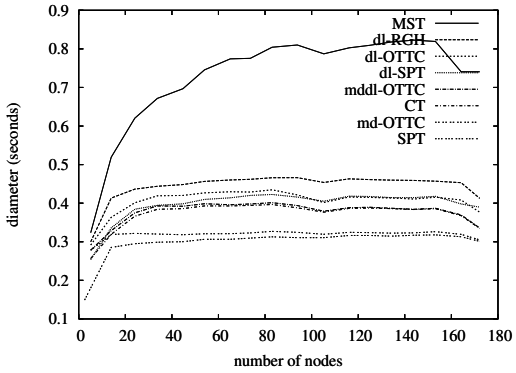

Fig. 4. Diameter of fully meshed graph (degree limit=10).

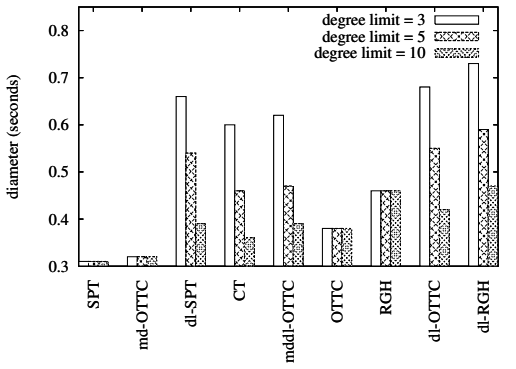

Fig. 5. Diameter of trees with 100 nodes (degree limits 3,5 and 10).

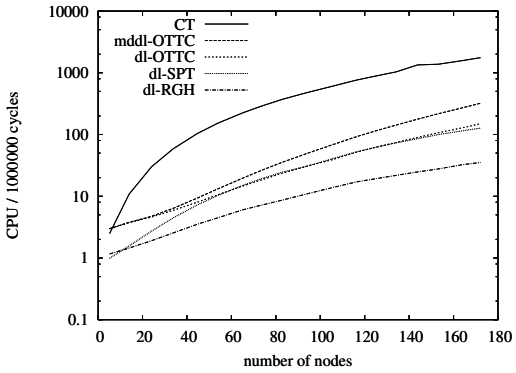

Fig. 6. Reconfiguration time using full mesh.

The *reconfiguration time* of selected algorithms is plotted in figure 6. CT and BCT clearly perform worst (only CT is plotted). In fact, during frequent group tree updates they are almost useless for larger group sizes. The remaining algorithms are considerably faster, with RGH/dl-RGH being the fastest. An SPT algorithm on a fully meshed application layer graph reduces the complexity to $O(1)$, because the input mesh contains all the shortest paths from the source to any destination.

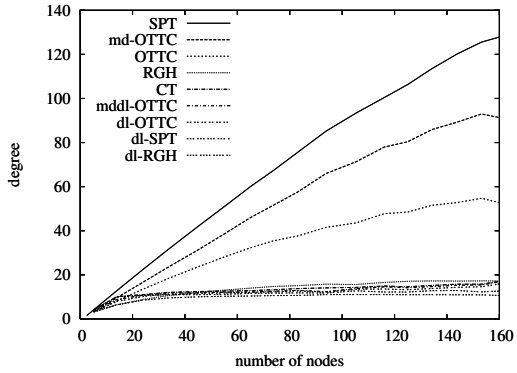

Fig. 7. Maximum degree using full mesh (degree limit=10).

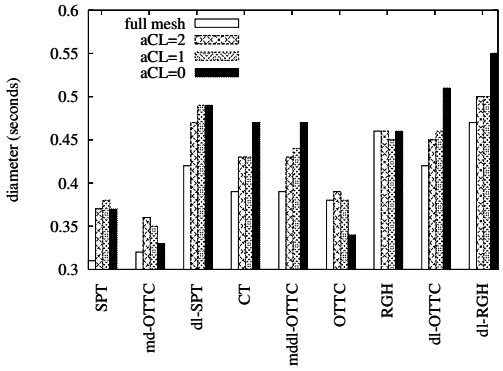

Fig. 8. Diameter for full mesh and *a*CL.

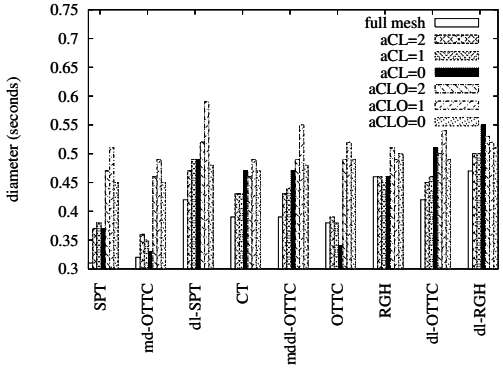

Fig. 9. Diameter for full mesh, *a*CL and *a*CLO.

Figure 7 plots the *maximum degree* in the group trees. SPT, md-OTTC, OTTC and RGH all have maximum degrees that would not be tolerated by average users of distributed interactive applications.

To summarize, MST and dl-MST produce trees with too high diameter. CT and BCT are too slow to handle frequent tree updates. SPT, md-OTTC, OTTC and RGH all have a maximum degree above acceptable. Hence, these are all *poor* alternatives in the tree construction. The *better* alternatives are dl-SPT, mddl-OTTC, dl-OTTC and dl-RGH. Table III gives an overview of some pros and cons of the algorithms. In the following, we consider only the diameter, reconfiguration time and maximum degree.

Algorithm	*Diameter*	*Time*	*Degree*	*Rank*
MST	–	+	+	–
SPT	+	+	–	–
md-OTTC	+	+	–	–
OTTC	+	+	–	–
RGH	+	+	–	–
CT	+	–	+	–
mddl-OTTC	+	+	+	+
dl-OTTC	+	+	+	+
dl-RGH	+	+	+	+
BCT	+	–	+	–
dl-SPT	+	+	+	+
dl-MST	–	+	+	–

TABLE III
TREE ALGORITHM CHARACTERISTICS USING FULL MESH.

D. Reduced Graphs

Here, we present results from combining a core selection heuristic and edge-pruning with tree algorithms. We use *a*CL and *a*CLO (k=2,1,0) to reduce the input graph size, and the group center heuristic to find the core nodes, i.e., well located group nodes. All the remaining plots are of 100 nodes with a degree limit of 10.

Figure 8 plots the *diameter* when using a fully meshed input graph and *aCL*. Overall, the algorithms produce the lowest diameter when using a fully meshed input graph. However, when *a*CL 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 9 plots the diameter for *aCLO* as well. We observe that the diameter suffers on average just below 20% when *a*CLO is used, instead of the full mesh. *a*CLO with $k = 0$ reduces the edge set by 95 %, and the construction results are still competitive.

The reconfiguration times of the construction algorithms applied to the full mesh and *a*CL/*a*CLO graphs are plotted in figure 10. The tendency is very clear. When the edge set is pruned the reconfiguration time is substantially reduced. However, CT (and BCT) continue to be very slow regardless of edge-pruning, because its execution time is dependant on the node set size.

We expect the *maximum degree* to decrease for the algorithms without degree limits when applying *a*CL and *a*CLO, see figure 11. We observe that the maximum degree is reduced to about 20 when *a*CLO is used. Hence, degree unlimited algorithms are an option for very low bandwidth streams, but, only if *a*CLO and the group center heuristic are used to manipulate the input graph. However, the degree-unlimited algorithms all have (almost) equally fast algorithm versions with degree limits. Table IV gives an overview.

E. Discussion

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 CT

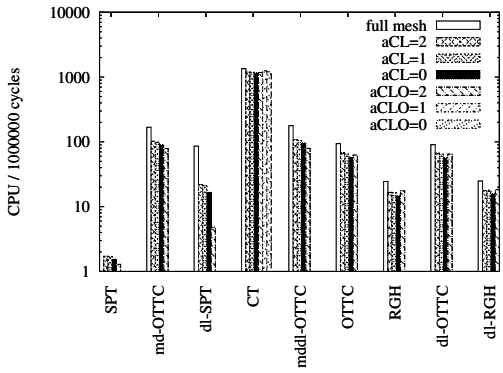

Fig. 10. Reconfiguration time for full mesh, *a*CL and *a*CLO.

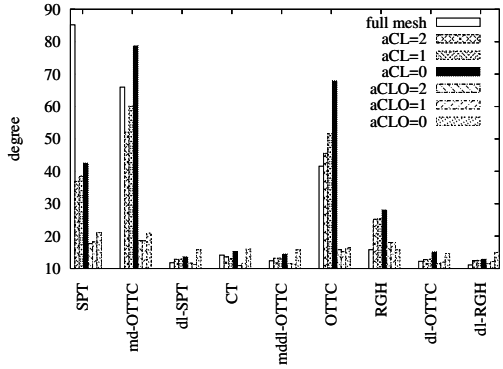

Fig. 11. Maximum degree for full mesh, *a*CL and *a*CLO.

Algorithm	*Diameter*	*Time*	*Degree*	*Rank*
SPT	+	+	+	+
md-OTTC	+	+	+	+
OTTC	+	+	+	+
RGH	+	+	+	+
dl-SPT	+	+	+	+
mddl-OTTC	+	+	+	+
dl-OTTC	+	+	+	+
dl-RGH	+	+	+	+

TABLE IV
TREE ALGORITHM CHARACTERISTICS USING *a*CLO.

Algorithm	*Diameter*	*Time*	*Degree*	*Rank*
dl-RGH	++	++++	+	++++
dl-OTTC	+++	++	+	+++
mddl-OTTC	++++	+	+	++
dl-SPT	+	+++	+	+

TABLE V
FINAL TREE ALGORITHM RANKING.

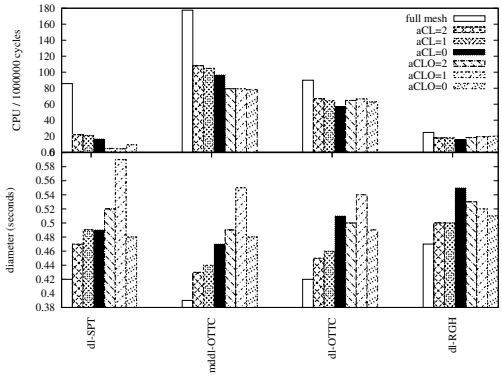

Fig. 12. Diameter and reconfiguration time for complete, *a*CL and *a*CLO.

is the best algorithm when only the diameter and maximum degree are considered. However, the reconfiguration time when using CT is very high, even with a pruned edge set. Remember, low reconfiguration time is particularly desirable during frequent tree updates, which is often the case for our target applications.

The algorithms that have all the properties that our target applications want are listed in table V. dl-RGH is the fastest algorithm, and still manages to produce low diameter trees within the degree limits. mddl-OTTC and dl-OTTC are similar to each other, but mddl-OTTC is slightly slower and does not have the flexibility of a bounded diameter algorithm. dl-SPT was a surprisingly good alternative, and is a good algorithm for a source-based tree.

Our ranking is subjective and not related to specific application needs. All the algorithms fit different needs, because they vary in performance between diameter and reconfiguration time, see figure 12. dl-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 small diameter. dl-OTTC extends the tree through the minimum weight edge that obeys the diameter bound. It is slower because it has a more time consuming maintenance of the diameter, but it produces trees with smaller diameter than dl-RGH. mddl-OTTC always minimizes the maximum diameter, and is slightly slower because of that. However, mddl-OTTC is much faster than CT, and constructs trees with almost the same small diameter. dl-SPT avoids diameter bounds, that may not be available, and minimum diameter goals, that may not be desirable in many applications. It rather optimizes for source destination cost, which is often desired by streaming applications.

V. 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. 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 [11]. Many overlay multicast protocols have been proposed, but there remains room for improvement, especially regarding the construction of overlay networks.

The Yoid project [6] provides an architecture for both space- and time-based multicast, and NICE [2] arranges group members into a hierarchy of layers and proposes arrangement and data-forwarding schemes. In Narada [5], end systems organize themselves into an overlay structure using a fully distributed protocol. ALMI [16] 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 [18], 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 [12]. 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.

VI. Conclusions and future work

We have investigated group communication in relation to distributed interactive applications. Our investigation involved experiments with many spanning tree problems, where we had a particular focus on reducing the diameter of a tree. We found that the fairly simple degree-limited heuristics dl-RGH, dl-OTTC, mddl-OTTC and dl-SPT all produce trees with small diameter. Moreover, the heuristics are fast, which is important in highly dynamic distributed applications.

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 *a*CL and *a*CLO are powerful means that may reduce the time a tree algorithm needs to construct a tree, while it still produces competitive results.

Currently, we are investigating tree algorithms that are able to include stronger nodes (like proxies) to a tree, that is, Steiner-tree heuristics. In addition, we study dynamic tree algorithms that insert and remove single nodes to reduce the reconfiguration time further [20]. Finally, we intend to test distributed alternatives to the centralized algorithms.

References

[1] A. Abdalla, N. Deo, and P. Gupta. Random-tree diameter and the diameter-constrained MST. Technical Report CS-TR-00-02, University of Central Florida, Orlando, FL, USA, 2000.

[2] S. Banerjee and B. Bhattacharjee. Analysis of the NICE application layer multicast protocol. Technical report, Department of Computer Science, University of Maryland, College Park, June 2002.

[3] M. Claypool and K. Claypool. Latency and player actions in online games. *Communications of the ACM*, 49(11):40–45, Nov. 2005.

[4] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: a decentralized network coordinate system. In *SIGCOMM*, pages 15–26, 2004.

[5] G. Fox and S. Pallickara. The Narada event brokering system: Overview and extensions. In *PDPTA*, pages 353–359, 2002.

[6] P. Francis, S. Ratnasamy, R. Govindan, and C. Alaettinoglu. Yoid project, 2000.

[7] M. Goodrich and R. Tamassia. *Algorithm Design: Foundations, Analysis and Internet Examples*. John Wiley and Sons, 2002.

[8] J. M. Ho, D. T. Lee, C. H. Chang, and C. K. Wong. Bounded diameter minimum spanning trees and related problems. In *SCG '89: Proceedings of the fifth annual symposium on Computational geometry*, pages 276–282, New York, NY, USA, 1989. ACM Press.

[9] International Telecommunication Union (ITU-T). One-way Transmission Time, ITU-T Recommendation G.114, 2003.

[10] A. Karaman and H. S. Hassanein. Core-selection algorithms in multicast routing - comparative and complexity analysis. *Computer Communications*, 29(8):998–1014, 2006.

[11] M. Kwon and S. Fahmy. Topology-aware overlay networks for group communication, 2002.

[12] L. S. Liu and R. Zimmermann. Immersive peer-to-peer audio streaming platform for massive online games. In *NIME*, 2006.

[13] A. Medina, A. Lakhina, I. Matta, and J. Byers. BRITE: Universal topology generation from a user's perspective. Technical report, Computer Science Department, Boston University, Apr. 2001.

[14] B. Moret and H. Shapiro. An empirical assessment of algorithms for constructing a minimum spanning tree, 1994.

[15] S. C. Narula and C. A. Ho. Degree-constrained minimum spanning trees. *Computers and Operations Research*, 7:239–249, 1980.

[16] D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel. ALMI: An application level multicast infrastructure. In *USENIX on USITS*, pages 49–60, 2001.

[17] G. R. Raidl and 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*, pages 747–752, 2003.

[18] S. Y. Shi, J. Turner, and M. Waldvogel. Dimensioning server access bandwidth and multicast routing in overlay networks. In *NOSSDAV*, pages 83–92, June 2001.

[19] K.-H. Vik, C. Griwodz, and P. Halvorsen. Applicability of group communication for increased scalability in MMOGs. In *Proceedings of the Workshop on Network and System Support for Games (NETGAMES)*, Singapore, October 2006. ACM Press.

[20] K.-H. Vik, C. Griwodz, and P. Halvorsen. Dynamic group membership management for distributed interactive applications. In *32nd IEEE Conference on Local Computer Networks (LCN)*, pages 141–148, 2007.

[21] A. Young, C. Jiang, M. Zheng, A. Krishnamurthy, L. Peterson, and R. Wang. Overlay mesh construction using interleaved spanning trees, Mar. 2004.