Constructing Low-Latency Overlay Networks: Tree vs. Mesh Algorithms

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

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

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

***Abstract*—Distributed interactive applications may have stringent latency requirements and dynamic user groups. These applications may benefit from a group communication system, and to improve the system support for such applications, we investigate graph algorithms that construct low-latency overlay networks for application-layer multicast. In particular, we focus on reducing the diameter and the pair-wise latencies in the overlay. The overlay construction time is also considered, as it is often time-dependent in our dynamic target applications.**

Here, we have implemented and experimentally analyzed spanning-tree heuristics and mesh construction heuristics, and compared their performance and applicability to distributed interactive applications. We found that trees are faster to construct and save considerable amounts of resources in the network. Meshes, on the other hand, yield lower pair-wise latencies and increases the fault tolerance, but at the expense of increased resource consumption.

I. Introduction

In recent years, many new types of distributed applications have appeared. The media types in these applications may range from text to continuous media such as video streams. Distributed interactive applications, like interactive 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. For example, groups of users (but not necessary all users) in the same application must be able to interact, and this interactivity imposes restrictions on network latency, especially in highly interactive virtual environments. Further, because (potentially large) groups of users in virtual environments interact, there is a need for many-to-many communication mechanisms. Many-to-many communication, combined with restrictions on network latency, results in requirements on the latency between any pair of users (pairwise latency).

Furthermore, in many distributed interactive applications, the user groups are highly dynamic. 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.

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, i.e., increase the users' perceived quality. 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 (frequently used also for time dependent applications like games), 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 overlays while conforming to some optimization goal. Both mesh and tree algorithms are considered, and due to our interactive time-dependent application scenario, we concentrate particularly on heuristics that minimize the pairwise latency in overlays. The maximum pairwise latency in an overlay is known as the diameter. All the algorithms are implemented, and the performance is analyzed using experiments. Our results show that trees can compete with meshes when it comes to computing low diameter overlays, however, the average pair-wise distance in meshes is lower. Furthermore, we show that both mesh and tree heuristics yield vital properties for use in distributed interactive applications.

II. Overlay algorithms and applications

In this paper, the targeted applications for our overlay algorithms are distributed interactive applications. These applications achieve live interaction between multiple participants over the Internet.

A. Applications and their requirements

Concrete examples of distributed interactive applications are multi-player online games, audio/video conferences, and virtual reality applications linked to education and entertainment. The characteristics of multi-player online game traffic have been analyzed several times before, and in [3], 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 [**?**]. 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.

B. Group communication in the Internet

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 relatively thin [7].

It is possible to use overlay 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 cyclic graph (*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. A mesh increases the node failure tolerance of the graph, because multiple paths to a node exist. Unless some path routing is applied to a mesh it introduces data redundancy because some nodes receive two copies of the same data. Multiple paths are also valuable in cases of fluctuating link costs and to reduce the pair-wise latencies. Generally, there are two main directions of mesh and tree (*overlay*) construction related to group communication in networks. Overlays that are constructed for a *single source* and for *multiple sources*. However, unless data redundancy is necessary because of unstable network nodes and/or links, a mesh is mostly used in a multiple source scenario.

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 II-A). 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 overlay should be constructed as a *shared-overlay*. For such a shared-overlay scenario, it is not enough to only consider an overlay viewed from a single source or a set of sources. Every node is both a source and a receiver, thus the worst case eccentricity in a shared-overlay equals the diameter. Therefore, in a shared-overlay scenario, the diameter is a very important metric.

C. Overlay and algorithm requirements

We have previously investigated a range of spanning tree problems [19] and Steiner-minimum tree problems [18] that reduce the diameter of trees. These investigations revealed several tree algorithms suitable for distributed interactive applications. In this paper, the investigation includes a subset of these tree algorithms and also a range of mesh algorithms. The goal is still to reduce the diameter of the overlay, but, in addition, the mesh algorithms should yield reduced eccentricities, and possess a configurable level of redundancy to allow for fault and congestion tolerance. Moreover, the complexity and consequently the execution time of the overlay algorithms should be low in cases of time-dependant construction requirements. Finally, there should be a bound on the *stress*, i.e., work-load, of each node in the overlay. 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.

III. Tree construction algorithms

The tree construction algorithms presented here are spanning tree algorithms [6] and construct trees that cover all the vertices in a given input graph. Formally, a spanning tree algorithm $\mathcal{A}_T$ takes as input 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. The spanning tree algorithm $\mathcal{A}_T$ then constructs a connected acyclic graph (tree) $T = (V_T, E_T)$ on G, where $V_T = V$.

A. Tree heuristics considering the diameter

We focus on spanning tree heuristics 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. In addition, we study heuristics that optimize for the *total cost*, i.e., the sum of the edge weights in T. Table I provides an overview of the tree construction heuristics.

One-time tree construction (OTTC) [1] is a $O(n^3)$ heuristic of the bounded diameter minimum spanning tree (BDMST)

Algorithm	*Meaning*	*Optimization*	*Constraints*	*Complexity*	*Problem*	*Reference*
MST	Prim's minimum-spanning tree	total cost	-	$O(n^2)$	MST	[6]
SPT	Dijkstra's shortest-path tree	core/destination cost	-	$O(n^2)$	SPT	[6]
md-OTTC	Minimum diameter one-time tree construction	diameter	-	$O(n^3)$	MDST	[19]
OTTC	One-time tree construction	total cost	diameter	$O(n^3)$	BDMST	[1]
RGH	Randomized greedy heuristic	total cost	diameter	$O(n^2)$	BDMST	[15]
mddl-OTTC	Minimum diameter degree-limited one-time tree construction	diameter	degree	$O(n^3)$	MDDL	[19]
dl-OTTC	Degree-limited one-time tree construction	total cost	diameter and degree	$O(n^3)$	BDDLMST	[19]
dl-RGH	Degree-limited randomized greedy heuristic	total cost	diameter and degree	$O(n^2)$	BDDLMST	[19]
dl-SPT	Degree-limited Dijkstra's shortest-path tree	core/destination cost	degree	$O(n^2)$	d-SPT	[13]
dl-MST	Degree-limited Prim's minimum-spanning tree	total cost	degree	$O(n^2)$	d-MST	[13]

TABLE I
TREE CONSTRUCTION ALGORITHMS ($\mathcal{A}_T$).

problem [19]. The NP-complete BDMST-problem optimizes for the *total cost* while obeying an upper bound diameter constraint. Formally, a BDMST-algorithm takes as input G and a bound $D > 0$. Then, it constructs a minimum weight spanning tree T on G, where $\sum_{e \in E_T} c(e)$ (total cost) is minimized and the diameter of which does not exeed D. OTTC is a modification of Prim's minimum spanning tree (MST) algorithm to accommodate the diameter bound. It maintains the node eccentricities as the tree is built and (if possible) adds the minimum weight edges that result in node eccentricities below the diameter bound.

Randomized greedy heuristic (RGH) [15] is a fast $O(n^2)$ heuristic of the BDMST problem for complete graphs. 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.

Degree-limited one-time tree construction (dl-OTTC) [19] is a $O(n^3)$ heuristic of the bounded diameter degree limited (BDDLMST) problem [8]. The NP-complete BDDLMST-problem is identical to the BDMST-problem only it adds degree limits on each node. Our dl-OTTC heuristic builds the tree the same way as OTTC, while obeying the degree limits.

Degree-limited randomized greedy heuristic (dl-RGH) [19] is a $O(n^2)$ heuristic of the BDDLMST problem. Our dl-RGH heuristic builds the tree the same way as RGH, while obeying the degree limits.

Minimum diameter one-time tree construction (md-OTTC) [19] is a $O(n^3)$ heuristic of the minimum diameter spanning tree (MDST) problem [8]. The MDST-problem is to find a spanning tree of a graph such that the diameter is minimized. Formally, an MDST-algorithm constructs a spanning tree T on G such that the maximum weight shortest path (diameter) $p \in T$, $\sum_{e \in p} W(e)$ is minimized. MDST-algorithms often construct a star-shaped tree, where the work-load (stress) of the central nodes is high. md-OTTC is our alteration of the BDMST-heuristic OTTC [1]. Instead of minimizing the total cost within a *diameter* bound, md-OTTC always adds the vertex that minimizes the diameter.

Minimum diameter degree-limited one-time tree construction (mddl-OTTC) [19] is a $O(n^3)$ heuristic of the minimum diameter spanning tree (MDDL) problem [16]. The NP-complete MDDL-problem introduces degree limits to solve the stress issues of the MDST. mddl-OTTC works as md-OTTC while obeying the degree limits.

B. A tree heuristic considering the shortest path

There exist related spanning tree problems that do not explicitly consider the diameter, but are cheaper in terms of the execution 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 [6] and has a worst-case time complexity of $O(n^2)$.

Degree-limited shortest-path tree (dl-SPT) [13] is a fast $O(n^2)$ heuristic of the degree-limited shortest path tree (d-SPT) problem. The NP-complete d-SPT problem is to find a spanning tree from a given source such that the source destination distance is minimized. Formally, a d-SPT algorithm is given G and a degree bound $d(v) \in N$ for each vertex $v \in V$. Then it constructs 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 $d_T(v) \leq d(v)$. It is an alteration of Dijkstra's original SPT algorithm. At each step it includes the cheapest eligible path connecting a vertex currently in the (partial) shortest-path tree with one not yet connected that does not violate the degree constraint. We showed in [19] that a d-SPT heuristic combined with a carefully selected center (source) node may be able to compete with heuristics that consider the diameter.

IV. MESH CONSTRUCTION ALGORITHMS

The mesh construction algorithms presented here construct a mesh that covers all the vertices in a given input graph. Formally, a mesh construction algorithm $\mathcal{A}_\mathcal{M}$ takes as input a connected undirected weighted graph G, and constructs a connected undirected cyclic graph (mesh) $M = (V_M, E_M)$ on G, where $V_M = V$. Observe that a mesh can be constructed like $M = G$. However, in our scenario, G is a fully meshed application layer graph, therefore, this is not a viable solution because the *total cost* of the mesh would be tremendous. The mesh construction algorithms should rather construct the mesh using our application requirements, without being bound to the requirement of constructing a tree.

The mesh construction algorithms that we investigate can be divided into the three categories: A) Interleaved-trees, B) Enhanced tree, and C) Edge pruning (removal and addition) algorithms. For each category, we present algorithms found in the literature, but also propose new mesh construction variations based on ideas from these (see table II).

Algorithm	*Meaning*	*Optimization*	*Input to $\mathcal{A}_{\mathcal{M}}$*	*Complexity*	*Category*	*Reference*
kIT	k-Iterative Tree constructions	$\mathcal{A}_{\mathcal{T}}$ optimization goal	G, k, $\mathcal{A}_{\mathcal{T}}$	$O(k * O(\mathcal{A}_{\mathcal{T}}))$	Interleaved trees	[21]
kDL	k-Diameter Links	latency diameter	G, k, $\mathcal{A}_{\mathcal{T}}$	$O(n^3 + O(\mathcal{A}_{\mathcal{T}}))$	Enhanced-trees	-
kLL	k-Long Links	pair-wise latency	G, k, $\mathcal{A}_{\mathcal{T}}$	$O(n^2 + O(\mathcal{A}_{\mathcal{T}}))$	Enhanced-trees	[20]
kBL	k-Best Links	minimum cost	G, k	$O(n^2)$	Edge pruning	[21]
dl-BL	degree limited Best Links	pair-wise latency	G	$O(n^2)$	Edge pruning	-
aCLO	add Core Links Optimized	pair-wise latency	G, k, $O \subset V$	$O(n^2)$	Edge pruning	[18]
dl-aCLO	degree limited add Core Links Optimized	pair-wise latency	G, k, $O \subset V$	$O(n^2)$	Edge pruning	[18]

TABLE II
MESH CONSTRUCTION ALGORITHMS ($\mathcal{A}_{\mathcal{M}}$).

A. *Interleaved-trees algorithms*

Interleaved-trees algorithms compute multiple connected trees and merge them into one mesh. The interleaved-trees algorithms may compute trees *sequentially* or in *parallel*. The sequential approach is round-based, where one tree is computed and merged with the previous trees in each round, and the previously chosen tree edges are excluded from the input graph. In the parallel approach, the trees are computed concurrently and possibly independantly of each other and then merged at the end.

The interleaved-trees algorithms may also be referred to as k-trees algorithms. Formally, a sequential k-trees algorithm comprises a tree algorithm $\mathcal{A}_{\mathcal{T}}$ that constructs k trees, where T_i is the ith tree. For each round $i \geq 1$ an input graph $G_i = G - M_i$ is created, where $M_i = T_1 \cup \ldots \cup T_i$, and $i \leq k$. The graph G_i is then input to the tree algorithm $\mathcal{A}_{\mathcal{T}}$ to produce T_{i+1}. Young et al. [21] described such an algorithm called k-MST, which computes k minimum spanning trees that are merged into one mesh.

The k-trees algorithms construct meshes that include the "best" edges given the optimization goal of the tree algorithm. For example, the k-MST algorithm ensures that the k minimum weight links of every node are included in the graph [21]. A k-trees algorithm does also produce an approximate k-connected graph. A k-connected graph is a graph in which the removal of any $k-1$ nodes does not disconnect the graph (a 1-connected graph is a tree). Informally, there are at least k independent paths from any vertex to any other vertex. A related graph-theoretic problem is a k-connected minimum weight subgraph, for which several approximation algorithms have been proposed [9]. However, these heuristics do not take degree constraints into account, and do not opt for a reduced diameter. In this paper, we use the tree-heuristics presented in section III as the basis for our interleaved-trees algorithms.

k-Iterative Tree-construction($\mathcal{A}_{\mathcal{T}}$,k) (kIT) takes as input a tree algorithm $\mathcal{A}_{\mathcal{T}}$ and computes k trees sequentially from an input graph and merges them to a mesh. In each round, the current input graph is updated such that the previously chosen tree-edges are excluded. For the degree-limited algorithms the current available degree on each node is reduced according to the previously chosen tree-edges.

B. *Enhanced tree algorithms*

Enhanced tree algorithms apply a tree algorithm to an input graph, and then add single edges to the tree based on some criteria. Wang et al. [20] described such strategies, and proposed an overlay protocol called Tmesh, which adds "short-cut" edges to a pre-constructed tree.

An enhanced tree algorithm comprises a tree algorithm $\mathcal{A}_{\mathcal{T}}$ that produces a tree T, and an edge-selection algorithm that adds edges to the graph T such that it is transformed into a mesh M. The number of edges that are added may be a predefined integer k, or based on some optimization goal, for example, a mesh-diameter below a given bound D. The most common edge-selection strategies add edges that reduce a node's eccentricity or the mesh's diameter. Notice that in a shared-overlay the maximum eccentricity equals the diameter, such that reducing the maximum eccentricity does in effect reduce the diameter of the mesh. Therefore, a strategy for reducing both is to always pick the nodes with the maximum eccentricity (= diameter) and try to reduce their eccentricity.

There are also edge-selection strategies that aim at reducing the pair-wise distances in an overlay. The average pair-wise distance of a node is the sum of the shortest path distances between it and every other node, divided by the number of nodes. Both Narada [4] and Tmesh [20] focus on reducing a node's pair-wise distance. Reducing the pair-wise distance or the diameter are very much similar goals. If the diameter is reduced, the pair-wise distance also reduces, however, if the pair-wise distance is reduced it does not automatically reduce the diameter. In a shared-overlay it is more important to reduce the diameter.

k-Diameter-Links($\mathcal{A}_{\mathcal{T}}$,k) (kDL) adds k links to an input tree T and constructs a mesh M. In each round, it tries to reduce the overlay diameter by adding a shortest-path edge between two nodes in the current diameter path. Ties are broken arbitrarily.

k-Long-Links($\mathcal{A}_{\mathcal{T}}$,k) (kLL) adds k links to an input tree T and constructs a mesh M. In each round, it chooses the node with the lowest degree and adds the longest shortest-path edge to an overlay-node. Ties are broken arbitrarily.

C. *Edge pruning algorithms*

Edge pruning algorithms include strategies that remove edges from an input graph G based on some goal, and also algorithms that pick single edges from an input graph and constructs a mesh M. These two approaches are essentially different, however, the algorithms share the same goal. Consequently, we call all of them edge pruning algorithms.

The simplest edge pruning algorithm is to add a number of edges randomly to the mesh. Yoid [5] applies this method, with the added step of applying a routing protocol atop of the mesh. In the Narada [4] protocol a node joins a random peer and then slowly moves to more favorable peers as they are discovered. Although random edge pruning algorithms are in use, they are the most naive edge pruning algorithms and we disregard them from this investigation.

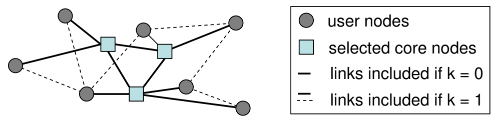


Fig. 1. A mesh produced by *add* Core Links Optimized.

k-Best-Links(k) (kBL) is an edge pruning algorithm that lets each node autonomously add its k minimum weight (best) edges to the mesh. In kBL, pairs of nodes will independently include each other into the mesh, such that the resulting mesh may have only $(k * n)/2$ edges (at most $k * n$). Because of this, kBL has a higher chance of producing a disconnected graph [21]. This is the main reason that few if any protocols use kBL as described. We have also tested a *degree-limited-Best-Links* (dlBL) edge pruning algorithm, which is a special case of kBL. In dlBL each node autonomously adds its minimum-weight edges to the mesh until its degree-limit is reached. We propose dlBL such that the node-degree is limited but also exploited to the maximum on each node.

We have also introduced an edge pruning algorithm [18] that produce meshes that include low weight links and higher weight longer links to a set of pre-selected core nodes, selected using a core selection heuristic [19]. Young et al. [21] have described similar randomized approaches as short-long strategies. Our algorithm combine kBL with shortest path links to connect the core nodes to the remaining nodes.

add Core Links Optimized(k) (*a*CLO) takes as input a graph G, an integer $k \geq 0$, and a set $O \subset V$ containing nodes that is identified by a core selection heuristic. In aCLO each node in $V - O$ includes its k minimum-weight edges into the mesh, where $k \geq 0$ (exactly like kBL). Then, it builds a full mesh of shortest paths of the nodes in O. Further, aCLO constructs $s = |V|/|O|$ disjoint sets $S_s = S_1 \cup \ldots \cup S_i$, where $i \leq s$, from the nodes in V. Finally, for each node $o \in O$ it adds edges to all nodes in set S_i. After applying *a*CL, the mesh forms, conceptually, a two-layer graph. Figure 1 illustrates a mesh after using *a*CLO. We also tested a degree-limited version of *a*CLO, that we labelled dl-*a*CLO.

V. Experiments and results

The previously introduced problems and observations were used when we designed the experiments for the overlay construction algorithms, and chose which metrics to target.

A. Group communication simulator

We implemented all algorithms presented in sections III and IV in a simulator for application layer multicast. It mimics group communication in a distributed interactive application using a central entity to handle the membership management.

We used the BRITE [12] topology generator to generate Internet-like router networks. The network graph was translated into an undirected fully-meshed shortest-path graph, where each router had one client attached to it. All the clients 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, it chooses an overlay construction algorithm, and with the latest available group graph given as input, it constructs a new group overlay. If not otherwise noted, we use the group center heuristic [19] to select a source node when needed. The group popularity is distributed according to a Zipf distribution. 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. Further experiment parameters are listed in table III.

Description	*Parameter*
Placement grid	$100x100$ milli-seconds
Number of nodes in the network	1000
Degree limit	5
Diameter bound	250 milli-seconds

TABLE III
EXPERIMENT CONFIGURATION.

B. Evaluated target metrics

In this paper, an overlay construction algorithm is considered good if it can produce overlays with a low diameter, a low average pair-wise distance, within a reasonable time that does not add unreasonable cost to the network. For our evaluation of the overlays and algorithms, we therefore consider four metrics to be very important: overlay diameter, average pair-wise distance, algorithm execution time, and total network cost. In addition, the algorithm should obey degree-limitations such that the stress on each node in the overlay is bounded.

C. Results from one group size range

In the following, we evaluate the results from our simulations. Our main plot is figure 2, which includes a complete comparison of the tree and mesh construction algorithms evaluated towards our target metrics. It includes statistics from overlays of sizes between 100 and 120. The tree-heuristics are plotted as interleaved tree algorithms with k = 1.

We observe that the best tree-heuristics achieve a *diameter* of around 0.3 seconds, while the degree-limited tree-heuristics achieve 0.5 seconds. For the degree-unlimited algorithms there is almost no gain in going from a tree to a mesh. For the degree-limited algorithms we can see a larger reduction in the diameter, but even here it is not significant. Among the enhanced-tree algorithms we see that kDL reduces the diameter more than kLL. Comparatively, we see that it is only the mesh algorithms that use MST or dl-MST as input that reduce the diameter significantly in a mesh (compared to a tree). The edge pruning algorithms all produce low diameter meshes, with the exception of kBL(k $<$ 3). We observe similar trends for the average *pair-wise* distance (seconds).

When we analyze the *total cost* of the overlays it is clear that the mesh algorithms build more costly overlays compared to the tree construction algorithms. The enhanced-tree algorithms only slightly increases the total cost of the overlays, due to the rather small k we used. The edge pruning algorithms kBL and dl-aCLO also yield a reasonable total cost. We see that the interleaved tree algorithms kIT(dl-MST,k) and kIT(dl-SPT,k) construct overlays with a low total cost. However, this is due to the fact that dl-MST and dl-SPT often fail to construct degree-limited trees in sparse graphs [18].

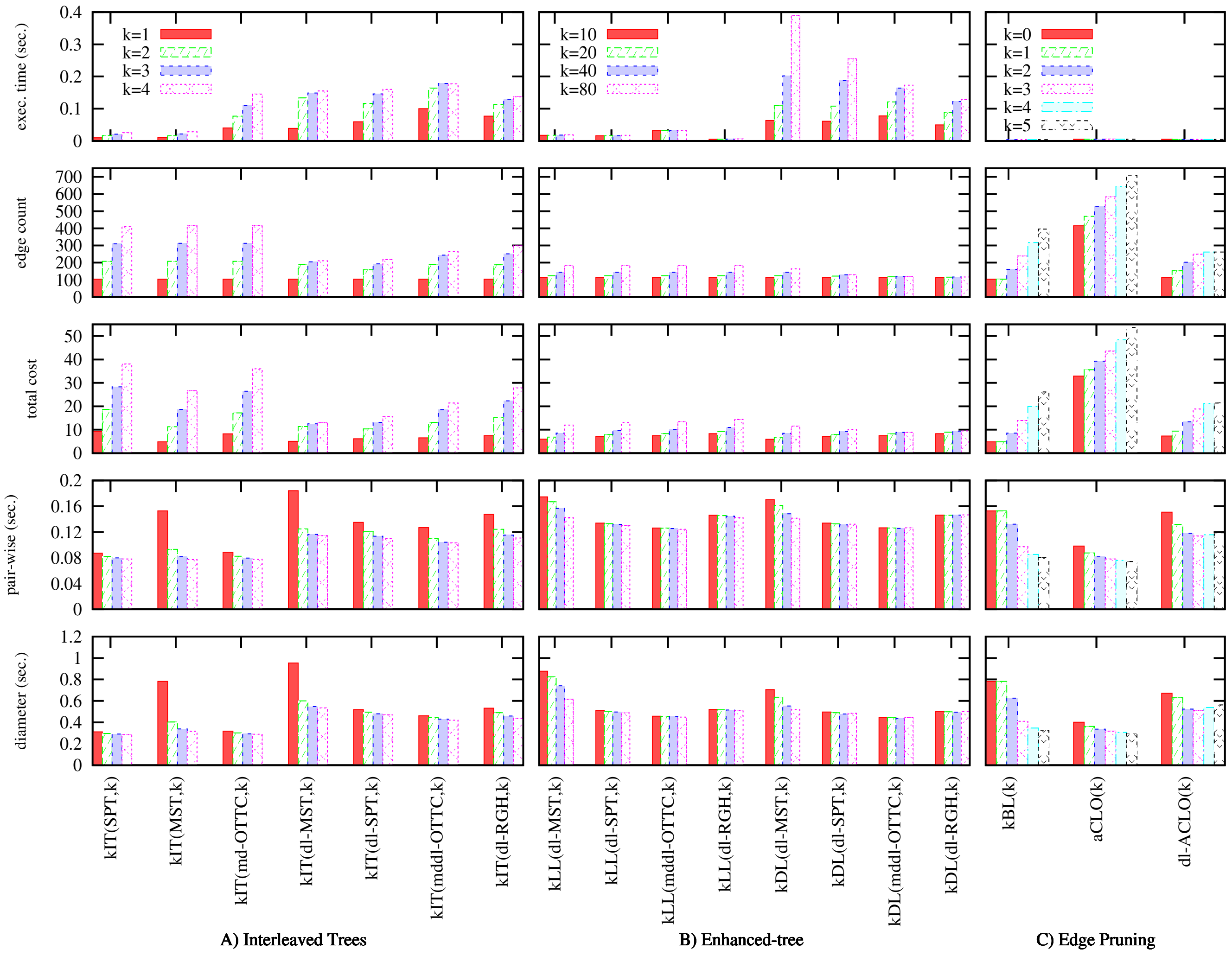


Fig. 2. Comparison of all tree and mesh algorithms.

The *edge count* is the number of links in an overlay, and the plots confirm this observation. We see that the edge-count for the kIT(dl-MST,2) and kIT(dl-SPT,2) overlays are much lower than the remaining interleaved trees algorithms. The edge pruning algorithm aCLO clearly adds too many edges to the overlay. For the interleaved-trees algorithms we can see that the edge-count increases in certain intervals for each k increase. The degree-unlimited algorithms merge a new full tree for each round, whereas, the degree-limited algorithms are not able to construct a full spanning tree for each round, in particular when $k > 2$.

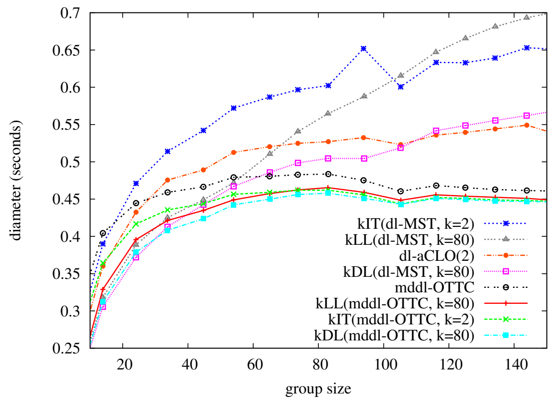
The *execution time* shows that the degree-limited interleaved-tree algorithms struggle to find trees when the k increases. The kDL algorithm is quite time-consuming when $k > 40$, however, our implementation may be sub-optimal for kDL. The fastest algorithms are the edge-pruning algorithms, which are barely visible in the graphs.

D. Results with varying group sizes

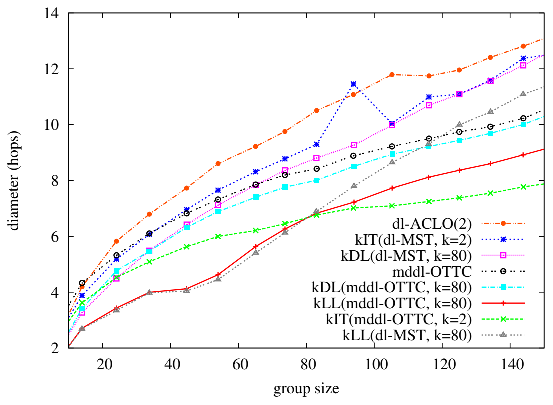
The group size may influence the performance of an overlay construction algorithm. In figure 3, we plot the diameter, total cost and overlay edge count for selected algorithms with group sizes between 10 and 150. We include results from the interleaved- and enhanced-trees algorithms using the tree algorithms dl-MST and mddl-OTTC as input.

The *diameter* achieved is plotted in figure 3(a). We observe that the reduction in the diameter between mddl-OTTC and kIT(mddl-OTTC,2), is very small. The other mddl-OTTC mesh algorithm variations perform very similar. dl-MST is not plotted but produces a diameter which is 30 % higher than kIT(dl-MST,2), and for k=3 the reduction is only about 5 % (not plotted). kLL(dl-MST,80) yields overlays with a lower diameter than kIT(dl-MST,2) for group sizes < 110, while kDL(dl-MST,80) achieves about 25 % better throughout our group size range. We observed very similar trends for the average *pair-wise distance* (not plotted).

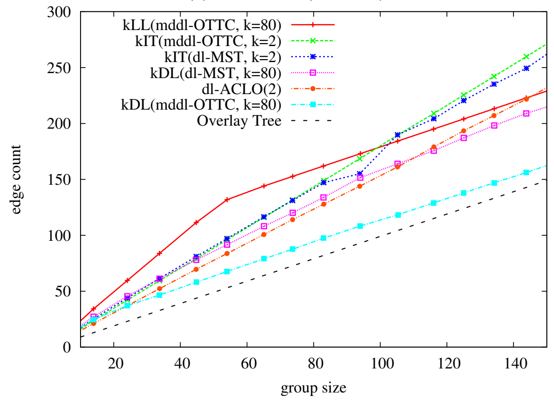
The *hop-diameter* achieved by the algorithms is plotted in figure 3(b). The hop-diameter reduction between mddl-OTTC and kIT(mddl-OTTC,2) is about 25 %. kDL(mddl-OTTC,80) does not reduce the hop-diameter much, although it reduces the diameter quite significantly. kLL(mddl-OTTC,80) and kLL(dl-MST,80) yield similar hop-diameter with group sizes < 80, and are both better than kIT(mddl-OTTC,2) in that group range. For group sizes > 80, kIT(mddl-OTTC,2) is best, with kLL(mddl-OTTC,80) only slightly higher.



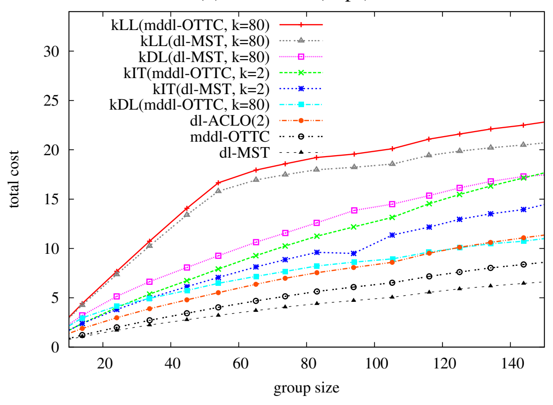
(a) Diameter (seconds)



(b) Diameter (hops)



(c) Edge count



(d) Total cost

Fig. 3. Comparison of selected overlay construction algorithms.

For the *total cost* plot in figure 3(d) wee see that the enhanced-trees algorithm KLL constructs the most expensive overlays. kIT(mddl-OTTC,2) build overlays that are slightly more expensive than kIT(dl-MST,2). The kDL(mddl-OTTC,80) is not able to add 80 links to each overlay, therefore its total cost is lower than expected. Not surprisingly, the trees constructed by dl-MST and mddl-OTTC are the cheapest.

The *edge count* in figure 3(c) confirms that kDL(mddl-OTTC,80) fails to add 80 links, due to the degree-limitation, but rather only manages to add about 15 links before it gives up. kDL(dl-MST, 80), on the other hand, has a much higher success rate in adding links. Close-to-minimum diameter trees are more star shaped (leafy trees), where inner nodes have a high degree. Therefore, kDL struggles to add edges to the diameter path because the degree capacity quickly gets exhausted.

E. The fine lines of the results

From our results we see that a tree is able to compete with a mesh when it comes to the latency diameter in an overlay. However, a mesh does have advantages in the hop-diameter. The main drawback of meshes is the added cost they incur, whereas the upside is the added failure tolerance. A tree is much cheaper, but is more failure prone.

Broadcasting shared events in a tree is cheaper than broadcasting events in a mesh. The extra bandwidth consumption in a mesh is not desirable, and may force some packet-routing on top of the mesh. In such cases it may be just as well to use a source-based tree for each client.

Among the mesh algorithms that use tree algorithms as input it is the dl-MST algorithm that has the largest diameter reduction. Furthermore, the edge pruning algorithm dl-aCLO produces meshes with a low diameter and total cost, and is very fast, however, it cannot be configured into a tree structure.

Table IV illustrates our subjective opinions on how to configure some selected mesh algorithms with the optimal k value. For each target metric an optimal k value is listed. These are averaged into the proposed k, which is what we consider the better way to configure the mesh algorithm. The mesh algorithm combinations with mddl-OTTC and dl-SPT use very low k-values, whereas the dl-MST combinations use higher k-values. The difference between dl-MST, dl-SPT and mddl-OTTC mesh algorithm combinations is reduced significantly with these configurations. For example, in the kIT algorithms, a tree structure is what we consider the most optimal for dl-SPT and mddl-OTTC.

VI. Related work

Considerable attention has been given to latency reduction in distributed interactive applications. Research areas such as graph theory (network layout), latency estimations, protocol optimizations (on all layers), group management (distributed and centralized), and multicast protocols are all necessary for the further enhancement of distributed interactive applications.

Algorithm			Optimal k			Proposed k
Mesh	Tree	k-range	*Total cost*	*Diameter*	*Hop-diam.*	
kIT	mddl-OTTC	$[1, 4]$	1	2	2	1
	dl-SPT	$[1, 4]$	1	2	2	1
	dl-MST	$[1, 4]$	1	3	3	2
kDL	mddl-OTTC	$[0, 80]$	0	15	15	7
	dl-SPT	$[0, 80]$	0	15	15	7
	dl-MST	$[0, 80]$	0	80	80	40
kLL	mddl-OTTC	$[0, 80]$	0	40	40	20
	dl-SPT	$[0, 80]$	0	40	40	20
	dl-MST	$[0, 80]$	0	80	80	40
kBL		$[1, 5]$	1	5	5	3
dl-aCLO		$[0, 5]$	0	4	4	2

TABLE IV
PROPOSED k-CONFIGURATIONS FOR A FEW SELECTED ALGORITHMS.

In this paper, our focus has been on comparing centralized overlay construction algorithms.

Overlay multicast can be divided into two general approaches. One is peer-to-peer networks [11] that are designed for file and information sharing in highly dynamic networks, for example BitTorrent and Gnutella. Most peer-to-peer 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 [10]. Many overlay multicast protocols have been proposed, but there remains room for improvement, especially regarding the construction of overlay networks.

The Yoid project [5] 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. ALMI [14] is a centrally managed group communication middleware, tailored towards relatively small multicast groups with many-to-many semantics. 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.

As this section describes, there is a considerable body of work on overlay construction algorithms. Studies have been performed aiming for efficient overlay construction and maintenance [17]–[21]. However, to the best of our knowledge, there does not exist a thorough evaluation and comparison of tree- and mesh-construction algorithms with the aim of constructing low-latency overlays. In this paper, we have aimed to highlight some of the unanswered questions regarding such algorithms.

VII. CONCLUSIONS AND FUTURE WORK

We have compared a range of tree and mesh construction algorithms, and evaluated their applicability to distributed interactive applications. Our investigation focused on quickly constructing overlays, that had a low diameter and pair-wise distance. Our results revealed that tree structures are cheaper than meshes and also yield a competetive diameter when diameter reducing tree heuristics are used. However, with a fairly limited number of added links, a tree may be optimized to be quite a lot better, especially for algorithms such as MST. Furthermore, our proposed edge pruning algorithm dl-aCLO proved to exhibit all of our desirable target metrics, except, that it cannot be configured to build a tree. We plan to continue our investigation with real-world experiments on PlanetLab.

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 UMIACSTR 2002-60 and CS-TR 4380, 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] G. Fox and 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)*, pages 353–359, 2002.

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

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

[7] C. Griwodz and P. Halvorsen. The fun of using TCP for an MMORPG. In *Proceedings of the International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, pages 1–7. ACM Press, May 2006.

[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] S. Khuller and U. Vishkin. Biconnectivity approximations and graph carvings. In *STOC '92: Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 759–770, New York, NY, USA, 1992. ACM.

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

[11] J. Liang and K. Nahrstedt. Dagstream: Locality aware and failure resilient peer-to-peer streaming. In *Proceedings of the SPIE/ACM Conference on Multimedia Computing and Networking (MMCN)*, Jan. 2006.

[12] A. Medina, A. Lakhina, I. Matta, and J. Byers. BRITE: Universal topology generation from a user's perspective. Technical Report BUCS-TR-2001-003, Computer Science Department, Boston University, Apr. 2001.

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

[14] D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel. ALMI: An application level multicast infrastructure. In *In Proceedings of the USENIX Symposium on Internet Technologies and Systems (USITS)*, pages 49–60, 2001.

[15] 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.

[16] S. Shi and J. Turner. Routing in overlay multicast networks. In *Proceedings of the Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, 2002.

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

[18] K.-H. Vik, P. Halvorsen, and C. Griwodz. Evaluating steiner tree heuristics and diameter variations for application layer multicast. *Accepted for publication in Computer Networks on Complex Computer and Communication Networks*, Nov. 2008.

[19] K.-H. Vik, P. Halvorsen, and C. Griwodz. Multicast tree diameter for dynamic distributed interactive applications. In *Proceedings of the Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pages 1597–1605, Apr. 2008.

[20] W. Wang, D. A. Helder, S. Jamin, and L. Zhang. Overlay optimizations for end-host multicast, Oct. 2002.

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