

Dynamic Group Membership Management for Distributed Interactive Applications

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 have become increasingly popular, making it important to address their communication needs, where one of the needs is group communication. In this paper, we consider the applications in which it is at any given time possible to divide its users into groups. The group membership changes over time, and the group division is unrelated to the physical proximity. As a way of enabling the group communication in distributed interactive applications, we choose application layer multicast. We use simulation to evaluate several dynamic algorithms for managing overlay multicast trees. They are compared with respect to four metrics that can be relevant for a distributed interactive application. These are total tree cost, diameter, reconfiguration time and stability. We demonstrate algorithms that perform well for these metrics although they do not consider all users during reconfiguration.

I. INTRODUCTION

Distributed interactive applications that make use of virtual environments are increasingly popular. Example applications are on-line virtual meeting rooms, shops, museums and games that host several concurrent participants. In the case of virtual environments, users are represented by so-called avatars. The users interact through the avatars and perform actions based on what they see and hear in the distributed virtual environment. Whenever a user acts through an avatar, the action and its effects must be presented to all others who can observe them. The central topic of this paper is the efficient management of user groups who can observe each other in the virtual world. In applications with thousands of users, it is typical that only small groups of avatars interact with each other at any given time. This allows us to divide the users into groups and investigate the communication among them as a *multi-party communication problem*.

The way in which participation in groups is determined differs between applications. Virtual environments are usually modeled after a semblance of the real world where most of the interaction takes place among avatars that are virtually close to each other in the virtual environment. Thus, location in the virtual environment is the most important factor to determine whether users interact. It defines implicitly groups of avatars who observe the events occurring within a region of the virtual environment called their *area-of-interest*. We can use areas-of-interest to define group membership, but we have to take into account that avatars move frequently

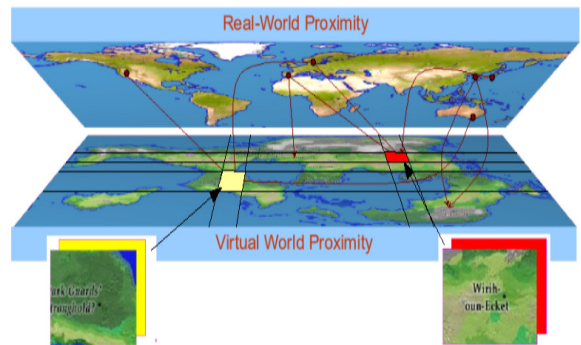


Fig. 1. Location in the real world, and area of interest in the virtual world.

and quickly in the virtual environment, making group membership highly dynamic. When group membership has been determined, efficient communication between the users in the group needs to be established. Here, the main challenge is to design algorithms that create efficient event distribution paths, by taking the physical location of the users who virtually interact into account. Figure 1 depicts the disparity between users' physical location and their areas-of-interest in the virtual world.

We use application layer multicast to achieve group communication although many distributed interactive applications support IPv4 multicast. The reasons to operate at the application layer are that IPv4 multicast 1) is not supported by all Internet service providers, 2) can not be efficiently used with TCP, 3) does not easily support frequent membership change, 4) can not prevent snooping, and 5) has a rather limited address space available. To avoid these problems, we build overlay networks. Such overlay networks are fully meshed and composed of nodes that include users' computers and a limited number of additional machines operated by the application provider. Our dynamic group management chooses overlay nodes and links from the full mesh which is an optimization problem with goals that differ between applications.

The rest of the paper is structured as follows: We explain the investigated metrics in section II, present the algorithms for group management in section III and discuss their comparison in section IV. We present related work in section V and make concluding remarks in section VI.

II. METRICS FOR GROUP MANAGEMENT

In this paper, we address the group management issues related to group membership changes. There are a number of contradictory goals in a group membership change. Efficient algorithms are vital for our application scenario, where the changes are very frequent, and where there are time critical events that need a short end-to-end latency. We observe that the algorithms executing the membership changes must be fast and produce distribution paths with low latencies. Furthermore, it is desirable to replace as few links as possible in a group reorganization because extensive changes in overlay networks lead to packet reordering and loss. Finally, it is always desirable to keep the total amount of network resources consumed low. Here, we consider only tree structures and assume full knowledge of the network. It is situation dependant which of these desirable goals is most important.

The algorithm that is chosen to execute the membership change is dependant on the current desired metric(s), which determines the quality of the resulting tree. In our experiments, we consider 4 metrics to address the above requirements. The *total cost* of the tree is the sum of all the edge weights and acts as a metric for the total bandwidth consumed. Reducing the *total cost* may help the scalability of distributed interactive applications [14]. The *diameter* expresses the worst case latency between any pair of group members. It is especially important in group communication when the event distribution is many-to-many within a group. A high *diameter* may give users an intolerable (and unfair) experience. The *reconfiguration time* is the time that is spent on executing a group membership change, which must be put in perspective with the frequency of membership changes. Finally, the *edge-change* is the number of link changes in a single group membership change, and is an estimate for the stability of a multicast tree. An algorithm with high *edge-change* leads to unstable traffic patterns and loads in the network.

A lot of work exists on the reduction of the total cost of group communication in networks. Heuristics for the Steiner minimum tree problem [15], [18] provide solutions that trade execution time for total cost. Other variations exist that produce graphs with minimum diameter while constrained to a given degree limit [11] (degree is also known as stress). However, these algorithms do not consider dynamic group membership, and stability is usually not a goal. We consider execution time and stability to be highly relevant for our application area, and we here evaluate algorithms that update trees dynamically whenever nodes are inserted or removed from a group.

We have tested combinations of 13 insert and 13 remove strategies and compared them with the Steiner minimum tree heuristic *shortest path heuristic* (SPH) and the minimum diameter degree limited heuristic *compact tree* (CT) that we have investigated in earlier work [14]¹. In this paper, we present only the evaluation for the most promising and interesting insert and remove combinations. All of the approaches

¹The CT heuristic was called MDDBST in [14].

investigated here are centralized where a well-known central server receives group change requests, which it then executes and distributes new overlay routing information.

III. ALGORITHMS

We implemented and tested combinations of 13 insert and 13 remove strategies for building dynamic algorithms to maintain overlay multicast trees. Such overlay multicast trees are used to distribute updates to the *group of members* (nodes of one area-of-interest, whereas other areas-of-interest define other groups and other trees). The nodes in a group are called its *member nodes*. The insert and remove strategies may include *non-member nodes* into an overlay multicast group (as well) to improve the resulting tree with respect to a chosen metric. Furthermore, we assume the availability of more powerful nodes for inclusion into the trees. We call these *well-connected nodes*, and they represent what would be called supernodes in interactive peer-to-peer applications or proxies in centrally managed applications [13].

In the following, we present results for representative strategies (5 insert and 3 remove) that perform well under the metrics presented in section II. We also compare the combinations to heuristics that are known to produce good results when the membership is static. The insert and remove strategies are invoked whenever a node is inserted into or removed from a group, respectively, while the heuristics must perform complete reconfigurations.

We use the following symbols, when we introduce the insert and remove strategies. We have an undirected weighted graph $G = (V, E, c)$, where V is the set of vertices ($n = |V|$), E is the set of edges, $c : E \rightarrow \mathbb{R}$ is the edge cost function. Informally, G comprises all the members of the application, the well connected nodes, and all the links that interconnect them. Each *group of members* is connected by an undirected group tree $T = (V_T, E_T)$, where $V_T \subseteq V$, and $E_T \subseteq E$. The group of members itself is the set $Z_T \subseteq V_T$, where $p = |Z_T|$. The *diameter* of T is defined as the longest of the paths in T among all the pairs of nodes in V_T . Moreover, the *total cost* of T is the sum of the edge weights in T . The degree of $v \in V_T$ is the number of incident edges $e \in E_T$ it has, and is expressed by $d_T(v) \in \mathbb{N}$. Each strategy obeys a given degree limit $d(v) \in \mathbb{N}$ that exists for each $v \in V$.

A. Dynamic algorithms

Each dynamic algorithm is comprised of one insert and one remove strategy. Table I provides a quick overview of abbreviations and features. We define an algorithm to be non-member node aware if it is able to insert or remove non-member nodes from a group tree. Furthermore, we here have tested algorithms that optimize for *total cost* or the *diameter*. Moreover, the reconfiguration set R is the worst-case number of edges that is changed between reconfigurations of a group tree. Finally, the time complexity (in big-oh notation) is the worst-case number of computational steps of an algorithm. Following is an introduction of the insert and remove strategies both formally and informally.

Algorithm	Meaning	Non-member node aware ¹	Optimization	Reconfiguration set R	Time complexity
SPH	Shortest Path Heuristic	yes	<i>total cost</i>	$ R \leq E_T $	$O(pn^2)$
CT	Compact Tree heuristic	no	<i>diameter</i>	$ R \leq E_T $	$O(n^3)$
I-MC	Insert minimum cost edge	no	<i>total cost</i>	$ R = 1$	$O(n)$
I-WCN	Insert well-connected node	yes	<i>total cost</i>	$ R \leq d(m) * 2$	$O(n^2)$
I-MDDL	Insert minimum diameter degree limited edge	no	<i>diameter</i>	$ R = 1$	$O(n^2)$
I-CN	Insert center node	no	<i>diameter</i>	$ R = 1$	$O(n)$
I-CW	Insert center well-connected node	yes	<i>diameter</i>	$ R \leq 2$	$O(n)$
RK	Remove keep as non-member node	no	<i>total cost</i>	$ R = 3 \rightarrow d_T(m) \leq 2$ $ R = 0 \rightarrow d_T(m) > 2$	$O(n)$
RTR-MC	Remove try replace and minimum cost	yes	<i>total cost</i>	$ R \leq d(m) * 2$	$O(n^2)$
RTR-P	Remove try replace and prune	yes	<i>total cost</i>	$ R \leq E_T $	$O(n^2)$

[1] The algorithm is able to add and remove non-member nodes.

TABLE I
ALGORITHM DESCRIPTIONS AND A SET OF PROPERTIES.

Insert strategies insert a new member m into the group tree and assure that m can communicate with the remaining member nodes. Formally, an insert strategy works like this:

Given $G = (V, E, c)$, a group tree $T = (V_T, E_T)$, and a new member $m \in V$. Create a new $\hat{T} = (\hat{V}_T, \hat{E}_T)$, where $\hat{V}_T \subseteq V$, $\hat{E}_T \subseteq E$. The set $\hat{Z}_T \subseteq \hat{V}_T$ is the updated group of members, where $\hat{Z}_T = Z_T + m$, and $\hat{p} = |\hat{Z}_T|$.

I-MC (minimum cost) finds the minimum cost edge, in terms of latency, for adding m to the tree. It optimizes for *total cost*, and create trees that resemble spanning trees, with a higher *diameter*. Formally, I-MC connects $m \in V$ to T through the minimum cost edge $e \in E$.

I-WCN (well-connected nodes) tries to include a well-connected node along with m , but only if it reduces the *total cost* of T . Formally, I-WCN finds the minimum cost edge $e \in E$ between m and a member node $v \in Z_T$ in the group tree T . Then, it chooses a well-connected node and checks whether an inclusion will reduce the *total cost* when it is used as an intersection for v , m , and the neighbor nodes of v . Otherwise, the minimum cost edge $e \in E$ is used.

I-MDDL (minimum diameter degree limited) finds an edge for adding m that minimizes the *diameter* of the new tree while obeying the degree limits. This strategy creates trees that have clearer centers than I-MC, where the *total cost* is generally higher. Formally, I-MC connects $m \in V$ to T through the edge $e \in E$ that minimizes the *diameter* of T .

I-CN (center node) optimizes for the *diameter*. Each group elects a center node from the the current group tree. Upon election, the center node has the smallest worst-case latency to all the other member nodes, and has not reached the degree limit. A new node m is (always) connected to it. Whenever the current center node has exhausted its degree limitation, a new one is elected. Formally, I-CN connects $m \in V$ to the group center node $v \in V_T$ using the edge $e = (m, v)$, where $e \in E$.

I-CW (well-connected center nodes) works like I-CN, but the center node is elected (if possible) from the well-connected nodes. Formally, I-CW connects $m \in V$ to the group center node $v \in V_T$ using the edge $e = (m, v)$, where $e \in E$.

Remove strategies remove a member m from the multicast tree while assuring that the group members stay connected. Formally, a remove strategy works like this:

Given $G = (V, E, c)$, a group tree $T = (V_T, E_T)$, and a member $m \in V$. Create a new $\hat{T} = (\hat{V}_T, \hat{E}_T)$, where $\hat{V}_T \subseteq V$, $\hat{E}_T \subseteq E$. The set $\hat{Z}_T \subseteq \hat{V}_T$ is the updated group of members, where $\hat{Z}_T = Z_T - m$, and $\hat{p} = |\hat{Z}_T|$.

The number of direct neighbor nodes of m (its degree) in T influences the necessary actions. If the degree $d_T(m) = 1$, m is a leaf that is simply removed along with the edge to its only neighbor. If it is greater than 1, a removal of m would partition the group if no additional steps are taken, and $d_T(m)$ unconnected subtrees would be the result. The neighbors of the leaving node are connected directly in the simple case $d_T(m) = 2$. The basic goal of remove strategies is the reconnection of subtrees into a single tree when $d_T(m) > 2$. (Use figure 2 while reading the (next) remove strategies.)

RK (keep) does not actually remove the leaving node m from the tree but requires it to forward traffic (as a non-member node), until its degree has dropped to 2. *RK* is most commonly used in the literature today. The authors have previously shown that it can degrade for larger groups [8]. Formally, $\hat{Z}_T = Z_T - m$ but m remains in $\hat{V}_T = V_T$.

RTR-MC (try remove, minimum cost edge reconnects) creates two sets of nodes, S_n , nodes that are necessary in the new tree, and S_o nodes that are optional to the new tree. S_n contains all direct neighbours of m , and S_o contains m and one well-connected node that has the shortest average distance to all nodes in S_n . Then, it removes all the edges that interconnect $S_n + S_o$ in the group tree, and finds the least cost tree that connects the nodes in S_n with each other. The nodes in the optional set S_o may be used if the *total cost* is reduced. The operation is basically an SMT heuristic. Formally, *RTR-MC* works by first identifying a subtree $T_- = (V_-, E_-)$ of T , where $V_- = S_n + S_o$, and $E_- \subseteq E_T$. Then it updates $\hat{T} = T - T_-$, such that \hat{T} is disconnected. Further, it reconnects \hat{T} using $T_+ = (V_+, E_+)$, where $V_+ \supseteq S_n$, $V_+ \subset S_o$, and $E_+ \subseteq E_T$. Finally, $\hat{Z}_T = Z_T - m$, and $\hat{T} = \hat{T} + T_+$ is reconnected.

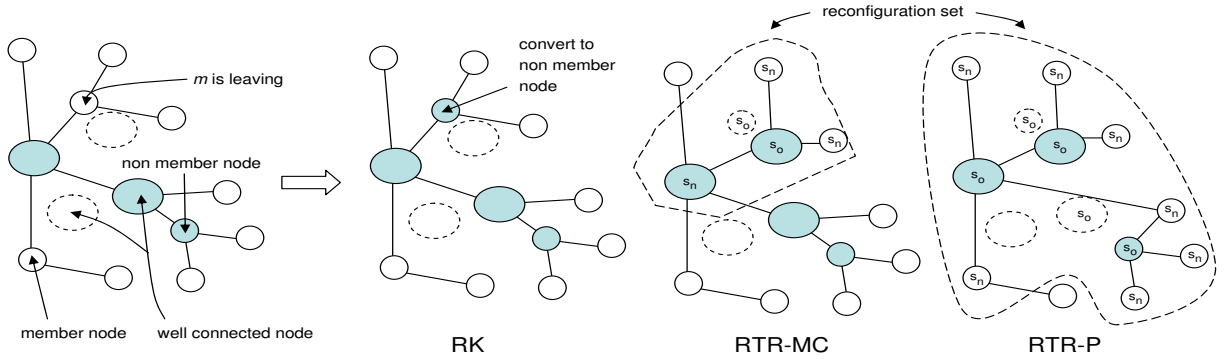


Fig. 2. Remove reconfiguration example.

RTR-P (try remove, prune well-connected nodes) is similar to *RTR-MC*. The difference is that S_n is not formed from the direct neighbors, but rather from all group member nodes that are either directly connected to m or that are only separated by non-members. All well-connected nodes and non-member nodes that are traversed are added to S_o . Then, it removes all the edges in the group tree that interconnect $S_n + S_o$, and finds the least cost tree that connects the nodes in S_n with each other. Optionally, nodes from S_o are used. *RTR-P* tries to minimize the number of nodes it uses from S_o , since it is only comprised of non-member nodes. Formally, *RTR-P* is the same as *RTR-MC*, but *RTR-P* has a greater influence on the tree. It is less stable because a large number of nodes can be candidates for reconfiguration. The advantage of *RTR-P* is that it reduces the number of non-member nodes in a tree, and thereby the *total cost*.

B. Tree heuristics

We have studied two related tree problems that we find relevant for the sake of comparison. They are the *Steiner minimum tree problem* and the *minimum diameter degree-limited spanning tree problem*, both of which are proven to be *NP-hard*. For each problem we introduce a tree heuristic that we have implemented and experimentally analyzed. The steiner minimum tree (SMT) problem is defined as [15]:

Given a graph $G = (V, E, c)$, where there is a subset $Z \subseteq V$ of group members. Find a connected undirected subgraph (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 members, and $\sum_{e \in E_Z} c(e)$ is minimized.

An SMT-algorithm builds a minimum-cost tree that may contain a set $V - Z$ of non-member nodes (aka. steiner points) that reduces the cost of the tree. The *shortest path heuristic* (SPH) [12] is a heuristic of the SMT-problem and has a time complexity of $O(pn^2)$. SPH is based on Prim's minimum spanning tree algorithm [7]. The tree is built from a source, and for each iteration, SPH connects the next member node through the minimum cost path to the tree. Waxman [16] showed that SPH is only performing 5% worse than the optimum SMT on average.

Description	Parameter
Placement grid	1000x1000 units
Number of nodes in the network	$n = 500$
Number of well-connected nodes	100
Degree constraint well-connected nodes	8
Degree constraint all other nodes	4

TABLE II
EXPERIMENT CONFIGURATION.

Next, we introduce the minimum diameter degree limited spanning tree (MDDL) problem, which is defined as [11]:

Given $G = (V, E, c)$, a degree limit $d(v) \in \mathbb{N}$ 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. *Compact tree* (CT) [11] is a heuristic of the MDDL-problem. CT has a time complexity of $O(n^3)$ and is also based on Prim's minimum spanning tree algorithm. It creates a tree of minimum diameter where each node in the tree obeys its degree limit.

IV. EXPERIMENTS

We have implemented a simulator (please see [14] for details), that mimics group communication in distributed interactive applications. It includes algorithms to create and configure groups. We generated network topologies with the BRITE Internet topology generator [10], and in our experiments, we used flat undirected Waxman topologies [16]. From this network, we created a fully connected undirected mesh, representing application layer communication that is shortest-path routed. We have tested different network sizes, and the results in this paper are from tests in a network of 500 nodes as a representative example. 100 of these nodes were well-connected nodes and could never be group members, while the other nodes could change their group membership over time. Group popularity was distributed according to a Zipf distribution. Further experiment parameters are listed in table II.

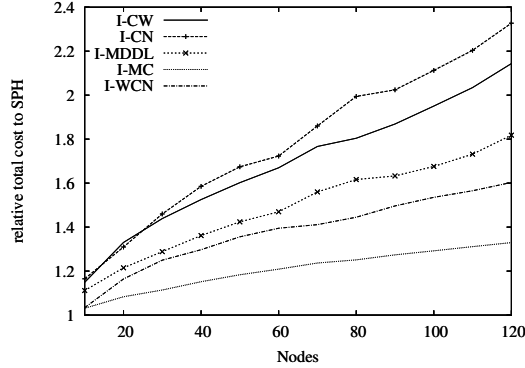


Fig. 3. Total cost relative to SPH for insert strategies using RTR-P.

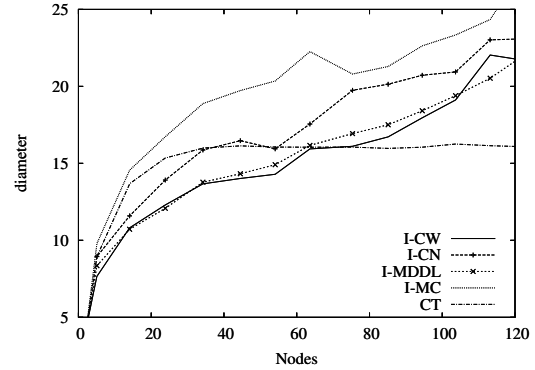
A. Individual results

The **reconfiguration time** of dynamic algorithms is very low compared to that of the SPH and CT heuristics [14]. The **reconfiguration time** of the dynamic algorithms is on average below 10 milliseconds for group sizes up to 120 member nodes (using a Pentium4 at 1.3 GHz), while, SPH and CT are 10 and 50 times slower, respectively. The main reason for this is the size of the reconfiguration set, e.g., figure 2. SPH and CT reconfigure (rebuild) the entire tree for each insert and remove, while, the dynamic algorithms reconfigure smaller parts of the tree. Hence, the time complexity of the dynamic algorithms does not greatly influence the **reconfiguration time** because the reconfiguration set size is small. Graphs of the **reconfiguration time** are not shown in this paper, but it puts the results for **diameter** and **total cost** into perspective.

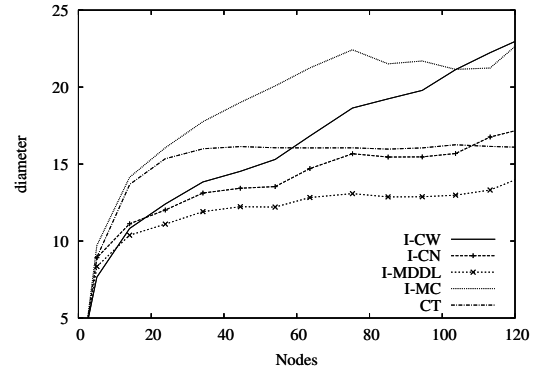
The **total cost** of the insert strategies combined with RTR-P, as RTR-P minimizes **total cost**, is compared to the close to optimal SPH in figure 3. All the dynamic algorithms have a higher total cost than SPH which does not have a **degree** limit. The cheapest trees are produced by I-MC, which is in worst case about 36% (group size 120) more expensive than SPH. Moreover, we observe that I-WCN performs badly although it is meant to insert well-connected nodes to reduce **total cost**. I-MDDL, which optimizes **diameter**, performs only slightly worse than I-WCN. The center node algorithms I-CN and I-CW have the highest **total cost**.

The **diameter** achieved is plotted in figure 4, where figure 4(c) shows that RTR-MC performs well with all insert strategies when $n = 100$. CT is plotted as a reference point in figures 4(a) and 4(b). SPH does not optimize for **diameter** and performs considerably worse.

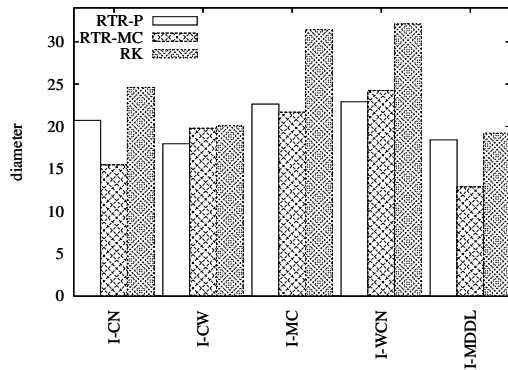
CT produces a tree with close to optimal minimum **diameter** using given **degree** limits. The algorithms that perform better are able to include (non member) well connected nodes with higher **degree** capacity. They are included if and only if it helps to optimize the tree towards the target metric. The increased **degree** capacity keeps the **diameter** small despite the simplicity of the algorithms and local reconfigurations. CT can not differentiate between non member and member nodes,



(a) Insert with RTR-P



(b) Insert with RTR-MC



(c) All combinations (100 nodes)

Fig. 4. Diameter of insert and remove strategy combinations.

and thus, can not include non member well connected nodes to reduce the **diameter**.

The insert strategies I-MDDL, I-CN and I-CW exhaust the **degree** of nodes that reduce the **diameter** of the tree. Commonly, such nodes are centrally located well connected nodes

(with higher *degree* capacity). The result is that many member nodes are leaves with a well connected node as a neighbor. Remember, well connected nodes are kept in the tree as non member nodes. Now, when a node m tries to leave, RTR-P will include many non member nodes to the reconfiguration set and reduce the number of non members (including well-connected nodes). Thus, a lower *degree* capacity is the result, which increases the *diameter*. RTR-MC does not prune non-member nodes (other than m), but may include well connected nodes. Thus, RTR-MC produce trees with higher *degree* capacity, which reduces the *diameter*.

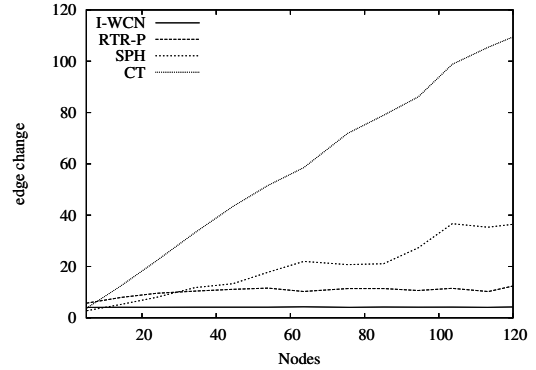
To summarize, the insert strategies I-MDDL, I-CN and I-CW in combination with RTR-P have higher *diameter* than the same insert strategies in combination with RTR-MC. RTR-P has a powerful prune mechanism (larger reconfiguration set) that makes the insert strategies less influential. RTR-MC is less powerful (smaller reconfiguration set) and thus cooperates better. I-MC quickly degrade and perform worse than CT, even for small groups. I-WCN is not plotted, but performs similarly to I-MC. RK only performs well combined with insert strategies that optimize for the *diameter*.

The **stability** of the dynamic algorithms is compared with that of SPH and CT in figure 5. A higher stability is indicated by a lower number of edges that are affected by the dynamics. Figure 5(a) compares the *edge-change* for the worst performing insert and remove strategies, I-WCN and RTR-P, with the heuristics SPH and CT. Clearly, the basic tree heuristics produce very unstable graphs. They reconstruct the entire tree by considering the fully meshed graph for every insert and remove operation. With the exception of I-WCN, all insert strategies change only one edge. I-WCN tries to insert a well-connected node for each member, which leads to five changed edges on average. Figure 5(c) shows that the stability of the insert/remove combinations are mainly determined by the remove strategies. Figure 5(b) adds the observation that the stability of RK and RTR-MC is unaffected by group size, while the stability of RTR-P drops. It is lowest when it is combined with insert strategies that optimize for *diameter* (I-CN, I-CW and I-MDDL). The reason for this is that all of these insert strategies are biased towards making connections between well-connected nodes, which RTR-P collapse to reduce their number. Conversely, the stability is higher in combination with I-MC and I-WCN because they optimize for *total cost* and make few connections between well-connected nodes.

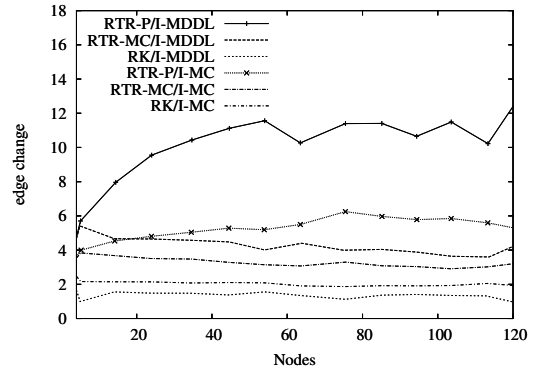
B. Combined results

We have tested several combinations of insert and remove strategies for efficiency, in terms of the tree metrics introduced in section II. We have also compared the dynamic algorithms with an SMT- and an MDDL-heuristic. Those heuristics outperform our strategies in many situations, but their practical use is limited by their performance with respect to the reconfiguration time in a dynamic scenario [14].

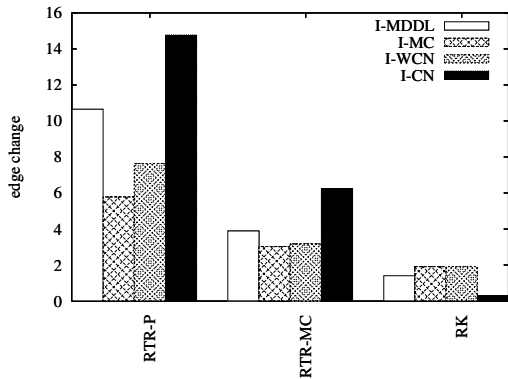
We observe that there exist insert and leave strategies that do not fit together. For example, an insert strategy that tries



(a) Worst case insert and remove, SPH and CT.



(b) Leave algorithms using I-MC and I-MDDL.



(c) All combinations (100 nodes).

Fig. 5. *Edge-change* of insert and remove strategy combinations.

to minimize the *diameter* does not fit best with a remove algorithm that reconfigures with the aim of minimizing the *total cost*. This effect can be seen clearly in the figures. RTR-P optimizes for *total cost* and we can see that the consequence is that the combination of RTR-P with I-MDDL or I-CN

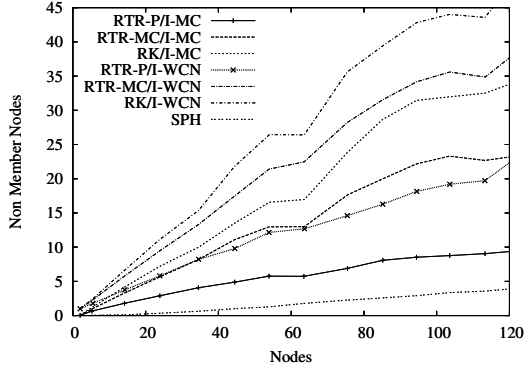


Fig. 6. Number of non-members for remove strategies using I-MC and I-WCN.

produces trees with lower *total cost* (figure 3) but higher *diameter* (figure 4(b)). Additionally, RK does not include well-connected nodes but leaves members with *degree* > 2 in the tree. We see that RK performs better with respect to the *diameter* in combination with the *diameter*-optimizing strategies I-MDDL, I-CN and I-CW. The algorithms oriented towards *total cost*, I-MC and I-WCN, are clearly worse.

When the *diameter*-oriented strategy I-CN is used, it is likely that most members are either leaves or have a higher *degree*. This leads to a small number of edge changes when it is combined with RK, as shown in figure 5(c). Since RK only removes a leaving group member if it has a *degree* ≤ 2 , it leaves inner nodes untouched. The negative effect that this has on the *diameter* can be seen in figure 4(c). RTR-P and RTR-MC avoid this penalty to the *diameter*, because they remove non-member nodes with higher *degree*, but, especially RTR-P, suffer the penalty in *edge-change*. The same tendency can be seen, to a lesser degree, in the combinations with the *diameter*-optimizing I-MDDL and the *total cost*-optimizing I-WCN.

Figure 6 plots the number of non-members in the trees, i.e., the *non-member node degradation*, using remove strategies combined with I-MC and I-WCN. We observe that the percentage of non-members in a group with 120 members is about 45% for RK, 30% for RTR-MC and 25% for RTR-P when I-WCN is used. I-MC does not include well-connected nodes and leads to much lower numbers. RK and RTR-MC are almost the same with about 20% non-members, RTR-P has below 10%. The SPH heuristic uses very few well-connected nodes (and zero non-members) because it builds the tree from scratch for each insert and remove, thus optimizing much better than the dynamic algorithms that only reconfigure small fractions of the tree.

C. Discussion

In the previous section, we identified that there exist insert and leave strategies that do not fit together, and some that fit especially well together. With this in mind, we have summarized all our observations in table III. In the table, every combination of insert and remove strategy as well as the tree heuristics are evaluated towards our target metrics. The

evaluation is based on the results from our experiments, but (inevitably) it is also based on subjective opinions from the authors. If an algorithm behaves positively towards a target metric it is represented with a " + ", correspondingly, if negatively with a " - ". The *performance index* is the sum of the positives (" + ") of an algorithm, where it is possible to get a maximum performance index of 4. (The optimization goals low *total cost* and small *diameter* are contradictory.)

Overall, we observe that RK performs well in combination with insert strategies that optimize for the *diameter*. I-MC and I-WCN combined with RK suffer from non-member node degradation, i.e., increasing number of non-member nodes in the trees. RTR-MC performs well with almost all insert strategies. The reason for this may be that RTR-MC keeps the reconfiguration set small, and, hence, does not greatly interfere the insert strategy. RTR-P fits well with insert strategies that optimize for the *total cost*. RTR-P also performs satisfactory combined with the remaining insert strategies, but the *diameter* does not get as good as with RTR-MC. Most importantly, the number of *edge changes* is higher with RTR-P. One approach may be to run RTR-P only periodically. SPH and CT both suffer from high *reconfiguration time* and many *edge changes*.

Algorithm		Performance metrics					Performance index
		MN ratio ¹	Total cost	Diameter	Stability	Reconfig. time	
Remove	Insert						
RK	I-MC	-	-	-	+	+	2
	I-WCN	-	-	-	+	+	2
	I-MDDL	+	-	+	+	+	4
	I-CN	+	-	+	+	+	3
	I-CW	+	-	+	+	+	4
RTR-MC	I-MC	-	+	-	+	+	3
	I-WCN	-	-	-	+	+	2
	I-MDDL	+	-	+	+	+	4
	I-CN	+	-	+	+	+	4
	I-CW	+	-	+	+	+	4
RTR-P	I-MC	+	+	-	+	+	4
	I-WCN	+	-	-	+	+	3
	I-MDDL	+	-	+	-	+	3
	I-CN	+	-	+	-	+	3
	I-CW	+	-	+	-	+	3
SPH		+	+	-	-	-	2
CT		+	-	+	-	-	2

¹The member node (MN) ratio in a tree should be high, if not it is crowded with non-member nodes and degrade.

TABLE III
ALGORITHM PERFORMANCE.

V. RELATED WORK

Distributed interactive applications may benefit from group communication, and the following overlay multicast protocols have some interesting aspects. The Yoid project [6] provides an architecture for both space- and time-based multicast, and NICE [3] arranges group members into a hierarchy of layers and proposes arrangement and data forwarding schemes. In

Narada [5], end systems self-organize into an overlay structure using a fully distributed protocol.

In our scenario, dynamic multicast algorithms support dynamic tree manipulations [19]. Typical operations include insert and remove, and some allow online rearrangement of the multicast tree. An insert operation is typically performed using the shortest path [1], [4] or the delay constrained minimum cost path [2]. This is cheaper than tearing down the tree and re-building it from scratch, but it will probably lead to a larger cost increase of the tree. A remove operation deletes a node from the multicast tree. A leaf node is trivially removed. Deleting a node with a degree of 2 results in two subtrees, these may be reconnected using the least cost path [17] or least cost delay bound path [2]. If the node has a larger degree (≥ 3), the node is often kept in the tree for routing purposes. However, this may degrade the trees [8]. Furthermore, it is critical that tree reconfiguration is fast, in particular node insert operations as new nodes should receive the data on time. It is also vital that remove operations keep the multicast tree intact for all remaining nodes. Existing interactive applications such as ACTIVE [9] perform insert operations more quickly, but the performance of the operation has not been given particular consideration.

There is a considerable body of work on overlay multicast protocols. Studies have been performed aiming for efficient tree construction and maintenance. However, current overlay multicast protocols do not address frequent group membership changes in any particular degree. Hence, we saw a need for investigating dynamic algorithms with different optimization goals.

VI. CONCLUSIONS AND FUTURE WORK

We here investigated dynamic multicast algorithms with distributed interactive applications in mind. The tested algorithms are centralized where a given entity handles the group management. The algorithms optimized, independently, for *total cost* and *diameter*, under given degree limits.

The results show that the dynamic algorithms have very low *reconfiguration time*, and that they produce trees that can compete with SMT- and MDDL-heuristics. In addition, we were able to constrain the number of *edge changes* for each reconfiguration. However, we also showed that there is still room for improvements, for example, constraining the number of non-member nodes in a tree.

Table III highlighted that RTR-MC performed very well combined with (almost) all of the insert strategy. RK and RTR-P, on the other hand, varied in their performance based on which insert strategy they were combined with. In more details, we found that combining I-MDDL and RTR-MC produces trees with low *diameter*. I-MDDL and RTR-P was not as successful, but was marginally competitive. The combination I-MC and RTR-P produces trees with low *total cost*, even compared to the close to optimal SPH. The center based insert strategies are fast and simple, but still gave good results in terms of the *diameter*. I-CW fit especially well in combination with the simplest remove algorithm RK.

In summary, our results show that the remove strategies RK and RTR-MC that use smaller reconfiguration sets, fit better with insert strategies that optimize for the *diameter*. RTR-P uses a (potentially) larger reconfiguration set and fit best with insert strategies that optimize for the *total cost*. The results also showed that the insert and remove strategies perform very differently with respect to our target metrics. We shall further investigate how the strategies can be changed dynamically whenever the desirable target metric(s) changes for a group. Moreover, we plan to investigate reconfigure strategies that are run independently of insert and remove strategies [4]. Finally, we plan to test distributed alternatives to the centralized dynamic algorithms.

REFERENCES

- [1] E. Aharoni and R. Cohen. Restricted dynamic Steiner trees for scalable multicast in datagram networks. *IEEE/ACM Transactions on Networking*, 6(3):286–297, 1998.
- [2] T. Alrabiah and T. Znati. SELDOM: A simple and efficient low-cost, delay-bounded, online multicasting. In *Proceedings of the IFIP Conference on High Performance Networking (HPN)*, pages 95–113, 1998.
- [3] 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.
- [4] F. Bauer and A. Varma. ARIES: A rearrangeable inexpensive edge-based on-line Steiner algorithm. In *Proceedings of the Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, Mar. 1996.
- [5] G. Fox and S. Pallickara. The Narada event brokering system: Overview and extensions. In *PDPTA '02: Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, pages 353–359. CSREA Press, 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] C. Griwodz, K.-H. Vik, and P. Halvorsen. Multicast tree reconfiguration in distributed interactive applications. In *International Workshop on Networking Issues in Multimedia Entertainment (NIME)*, Jan. 2006.
- [9] L. S. Liu and R. Zimmermann. ACTIVE: A low latency P2P live streaming architecture. In *Proceedings of SPIE/ACM Conference on Multimedia Computing and Networking (MMCN)*, Jan. 2005.
- [10] A. Medina, A. Lakhina, I. Matta, and J. Byers. BRITE: Universal topology generation from a user's perspective. Technical Report 2001-003, 1 2001.
- [11] S. Y. Shi, J. Turner, and M. Waldvogel. Dimensioning server access bandwidth and multicast routing in overlay networks. In *Proceedings of NOSSDAV 2001*, pages 83–92, June 2001.
- [12] H. Takahashi and A. Matsuyama. An approximate solution for the Steiner problem in graphs. *Math. Japonica* 24, 24(6):573–577.
- [13] K.-H. Vik. Game state and event distribution using proxy technology and application layer multicast. In *Proceedings of the ACM International Multimedia Conference (ACM MM)*, pages 1041–1042. ACM Press, 2005.
- [14] K.-H. Vik, C. Griwodz, and P. Halvorsen. Applicability of group communication for increased scalability in MMOGs. In *NetGames*, Singapore, October 2006. ACM Press.
- [15] S. Voss. Steiner's problem in graphs: heuristic methods. *Discrete Appl. Math.*, 40(1):45–72, 1992.
- [16] B. M. Waxman. Routing of multipoint connections. *IEEE Journal of Selected Areas in Communications*, 6(9):1617–1622, Dec. 1988.
- [17] B. M. Waxman. Dynamic Steiner tree problem. *SIAM J. Discrete Math.*, 4:364–384, 1991.
- [18] P. Winter. Steiner problem in networks: a survey. *Netw.*, 17(2):129–167, 1987.
- [19] L.-C. Wu, L. S. Lin, and S.-C. Shiao. Constructing delay-bounded multicast trees in computer networks. *J. Inf. Sci. Eng.*, 17(3):507–524, 2001.