On the Influence of Latency Estimation on Dynamic Group Communication using Overlays

Knut-Helge Vik, Carsten Griwodz, Pål Halvorsen

Simula Research Laboratory, Norway, and
Department of Informatics, University of Oslo, Norway

ABSTRACT

Distributed interactive applications tend to have stringent latency requirements and some may have high bandwidth demands. Many of them have also very dynamic user groups for which all-to-all communication is needed. In online multiplayer games, for example, such groups are determined through region-of-interest management in the application. We have investigated a variety of group management approaches for overlay networks in earlier work and shown that several useful tree heuristics exist. However, these heuristics require full knowledge of all overlay link latencies. Since this is not scalable, we investigate the effects that latency estimation techniques have on the quality of overlay tree constructions. We do this by evaluating one example of our group management approaches in Planetlab and examining how latency estimation techniques influence their quality. Specifically, we investigate how two well-known latency estimation techniques, *Vivaldi* and *Netvigator*, affect the quality of tree building.

1. INTRODUCTION

Many types of distributed interactive applications are popular today. Examples include audio/video conferencing, online games, virtual museums and shopping malls, etc. In these applications, the media types may range from text to continuous media, each which must be delivered with stringent latency bounds while satisfying some bandwidth requirements. Especially, the interactivity imposes demanding restrictions on network latency. Thus, to provide a satisfactory user service, efficient means for timely data delivery are important. A complication for achieving this is that not all users of such an interactive distributed application need to receive the same data, but that users form subgroups instead. Online multiplayer games, for example, tend to define such groups through regions-of-interest. All-to-all communication is required within such subgroups, and membership in the groups may change frequently.

In our work, we investigate how to construct efficient distribution paths using graph algorithms that construct overlay multicast trees. The goal is to make group communication feasible in these applications. To address latency constraints and bandwidth demands at the same time, we have previously investigated many different graph and tree algorithms. The algorithms achieve low diameters (maximum pair-wise latencies) in multicast trees while staying within degree bounds to satisfy bandwidth demands.[1,2] In our tests, we have found several suitable algorithms. However, a general challenge is that these algorithms have an underlying assumption of full network knowledge.

Achieving full, up-to-date knowledge of the network requires monitoring and is nearly impossible for a large number of nodes because the monitoring traffic grows quadratically with the number of nodes. This scalability problem is addressed by techniques that estimate link latencies and costs, but the trade-off is their accuracy. We investigate these accuracy problems in our paper. We use link estimation to perform overlay tree construction and compare this with tree construction based on full network knowledge. Specifically, we have implemented a group communication system that includes the link estimation techniques *Vivaldi*[3] and *Netvigator*[4] which estimate link latencies between participating clients, and we compare their estimated latencies with all-to-all measurements using *ping*. We build trees using our *degree-limited shortest path tree (*dl-SPT*)* approach to achieve a low tree diameter[2] and select the source of the tree using the *group center* heuristic. Overlay (multicast) trees are updated based on the latency estimates, degree constraints and group membership information. To achieve realistic, dynamically changing latency, we have evaluated the penalty of estimation techniques on our algorithms' results through experiments on PlanetLab.

Our results show that Netvigator performs better than Vivaldi in terms of the latency estimations. However, Vivaldi is easier to deploy than Netvigator and recovers from membership dynamics. The evaluation of the core heuristics and tree algorithms shows that it is possible to use the estimations from both Vivaldi and Netvigator to find appropriate core nodes

and construct trees of low diameters. Our main conclusion is that both Vivaldi and Netvigator make group communication feasible for use in dynamic interactive applications.

The rest of this paper is organized as follows. Some related work is given in section 2. In section 3, we introduce our group communication scenario including challenges and algorithms. Section 4 presents PlanetLab results from Vivaldi and Netvigator. Section 5 tests core selection heuristics, and section 6 tests tree heuristics applied to the latency estimations. In section 7, we extent our evaluation and look at the link estimation penalty on alternative tree construction algorithms. Finally, conclusions and future work are presented in section 8.

2. RELATED WORK

There exist many types of interactive applications. However, the inherent interactive nature poses strict requirements, especially with respect to latency. For example, multi-player online games allow thousands of clients to interact concurrently in a persistent virtual environment. The characteristics of game traffic have been analyzed several times before. Claypool[5] determined that first-person shooters require a latency of approximately 100 milli-seconds, role-playing games 500 milli-seconds and real-time strategy games 1000 milli-seconds. In audio conferencing and voice over IP (VoIP) with real-time delivery of voice data, clients start to become dissatisfied when the latency exceeds 150-200 milli-seconds, although 400 milli-seconds is acceptable in most situations.[6]

These investigations have determined the limits that are also valid for distributed interactive applications with larger numbers of users. The diameter of an overlay graph that is constructed for all-to-all communication among users in the same group should therefore stay within these limits. This requires firstly, knowing the communication delay between overlay nodes, and secondly, building the graph appropriately.

The round trip times (RTTs) between hosts in the Internet may be obtained by using readily available tools like Ping and Traceroute. These tools are well suited for smaller groups of hosts. However, an all-to-all measurement incurs an $O(n^2)$ traffic growth.[7] In our target applications, this poses a scalability problem because thousands of clients participate. The clients form dynamic subgroups, making it important to retrieve the link latencies among the clients, in order to organize the subgroups intelligently in an overlay graph. In such scenarios, an $O(n^2)$ approach is not sufficiently scalable. Instead, latency estimation techniques that reduce the probing overhead must be applied. Estimation techniques have been classified into three classes, 1) landmarks-based latency estimation, 2) multidimensional-scaling based latency estimation and 3) distributed network latency database.[8] In this paper, we evaluate representative latency estimation techniques for class 1 and 2. Class 3 requires $O(n^2)$ efforts to update the database and is therefore too expensive in our scenario.

Landmark-based latency estimation techniques include Netvigator, NetForecast, Global Network Positioning (GNP) and Practical Internet Coordinates.[9] Among these, Netvigator is often considered the most accurate.[4] Netvigator probes a set of landmark nodes L asynchronously from N clients using Traceroute, i.e., $L \times N$ probes in every round. Each node reports its measurements to a repository (typically a server node), which estimates a global graph with latencies. Another approach is to use multidimensional-scaling based techniques which do not need any infrastructure and works in any peer-to-peer setting. Such techniques include Vivaldi[3] and the Big Bang Simulation technique.[9] Vivaldi is based on spring embedding, which models network nodes as masses connected by springs (links) and then relaxes the spring length (energy) in an iterative manner to reach the minimum energy state for the system. All clients joining the system are placed at the origin of a virtual 2D coordinate system, and start sharing Vivaldi information with other clients. The Vivaldi information includes a client's coordinates in the virtual coordinate system, confidence estimations and latency measurements. Estimations are reported to a repository (typically a server) that does some calculations and updates clients' coordination in the virtual coordinate system in such a way that measured latency is represented by Euclidean distance.

Several others latency estimation techniques have been proposed and evaluated.[8] Netvigator and Vivaldi were found to be among the top ranked techniques in their respective latency estimation technique classes. Vivaldi has the advantage that it recovers from client churn (clients joining and leaving), and does not depend on any infrastructure. Netvigator, on the other hand, needs Landmark nodes that it controls and does not (easily) recover from churn. Table 1 summarizes the comparisons.

As this section shows, there is a considerable body of work on latency estimation techniques. However, we have not found any work that investigates the performance penalties that latency estimation techniques impose on group communication approaches that frequently re-configure multicast trees based on an assumption of full knowledge of the network. In this paper, we investigates these penalties.

Technique	*Measurement Overhead*	*Requires*	*Churn recovery*	Infrastructure dependability
Vivaldi	-	Inter-nodes traffic	yes	no
Netvigator	$O(L * N)$	Traceroute	no	yes

Table 1. Properties of the latency estimation techniques.

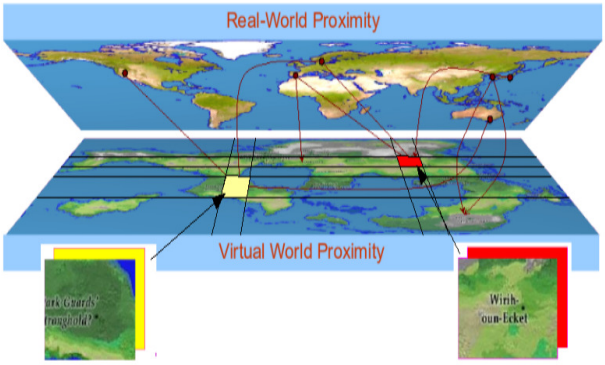


Figure 1. Real location versus virtual area of interest

3. GROUP COMMUNICATION

The group management mechanisms in applications with high membership dynamics must address particular challenges. For example, in an online game, the the gamers is not interested in all events happening in the entire game, but would like to limit the amount of data received to the local region or their area of interest (figure 1). However, as the players move, they also change the region of interest. The need to support this dynamics and client subgroups leads to contradictory group management goals. In that context, we have investigated graph algorithms that build trees (tree algorithms). These algorithms compute trees from connected input graphs while satisfying certain criteria for optimization. Rapid membership change requires that the algorithm itself executes quickly. Latency-critical data requires that graphs are built with a small diameter and bandwidth limitations impose degree constraints on the nodes in the graph. Our algorithms require first the identification of a fully meshed global graph that contains properties of the nodes and links in the overlay network. Many tree algorithms are spanning tree algorithms that try to build a tree covering all the group members (vertices).[2] We have enhanced the performance of the tree algorithms by using core-node selection heuristics. We used these heuristics to identify nodes that are centrally located and that may help tree-building heuristics to achieve a low diameter (maximum pair-wise latency) of the tree at limited computing cost. A distribution tree is thus built such that the data flows directly between the nodes in a group.

We have determined in earlier work[1,2] that the run-time of tree-building algorithms can be improved with little penalty to graph quality by manipulating the input graph before constructing the target graph. This means that our graph management process consists of three steps or phases: *identification*, *manipulation* and *construction*. In the next subsections, we briefly look closer at each of these before we in the following sections investigate for selected algorithms how the use of latency estimation influences the quality achieved in each of these three steps.

3.1 Identification

In the identification phase, we identify network characteristics among clients or nodes. Thus, we need to collect information about the current state of the network, and it is here that we employ latency estimation techniques. In section 4, we compare the quality of Netvigator's and Vivaldi's estimation with the **close-to-real** all-to-all measurements performed with ping.

3.2 Manipulation

The manipulation phase has optional techniques to reduce the time required for membership updates requiring new tree constructions (e.g., pruning the input graph for the construction phase) and to optimize the distribution paths (e.g., finding a suitable core node which the construction phase can use as a root node). We have earlier looked at both types of manipulation mechanisms, but in this investigation, the manipulation phase identifies a core-node that is well-suited as

the root node for the construction phase (using *dl-SPT*). Several core selection heuristics have been proposed, and a comprehensive study is given by and Karaman and Hassanein.[10] An overall goal is to select cores on the basis of certain node properties, such as, latency, bandwidth and computational power. We base our decision primarily on latency. The core selection depends on the location and degree limitations of the available core nodes. By applying core selection heuristics that identify well-placed nodes in a group, we design algorithms that exploit their degree capacity and reduce the diameter of the group tree. For example, degree-limited heuristics often exhaust the degree limits on centrally located nodes in the input graph. The core selection heuristic used here is the **group center** heuristic:[10] *Given a graph $G = (V, E)$; find the core node $c \in V$ that is the graph median, i.e., the node for which the sum of lengths of shortest paths to all other vertices is the smallest.* In section, 5, we investigate the latency estimation mechanisms influence on the results of the core selection heuristic.

3.3 Construction

In the construction process, the latest group graph information is used as input to a graph algorithm that creates (or updates) the group distribution graph (the used overlay network). In earlier work, we have investigated several tree heuristics for group communication.[2] We have performed our investigation of the performance degradation with several of our group management strategies. In this paper, we primarily look at *dl-SPT*[11] which is one of the best-performing heuristics among 12 different tree heuristics for optimizing the diameter while minimizing the reconfiguration time. *dl-SPT* is a modification of Prim's minimum spanning tree algorithm that approximates the **degree-limited shortest-path tree problem:** *Given a graph $G = (V, E)$, a degree bound $d(v) \in N\ \forall v \in V$, a cost function $c(e) \in N\ \forall e \in E$; find a spanning tree $T \in G$, starting from a root node $s \in V$, where, $\forall v \in V$ the path $p = (v, \ldots, s)$ minimizes $\sum_{p_i \in p} c(p_i)$, subject to the constraint that $\forall v \in V$, $degree_T(v) \leq d(v)$.* In section 6, we look at how latency estimations affect the results of *dl-SPT*, and in section 7, we add results for several other construction (tree generation) algorithms.

4. IDENTIFICATION PERFORMANCE

Up-to-date information about the nodes and networks is collected in the identification phase of our group communication system. Since we build overlay networks, the nodes are basically forming an fully meshed network, which implies that the collection of link costs scales quadratically with the number of nodes. This is not feasible and the reason for investigating estimation techniques. We use three metrics to compare the quality of Netvigator's and Vivaldi's estimations with all-to-all measurements using ping:

- *Relative error* expresses the relative difference between the latency between pairs of nodes that is actually measured using *ping* and the estimated latency using Netvigator and Vivaldi.
- *Relative rank loss (RRL)*[12] expresses how well the relative closeness of nodes is maintained when estimations are used instead of all-to-all measurements. It is computed for every node. For a node A, it is checked for every pair of other nodes B and C that when B is closer to A than C according to measurements, it is also closer according to the estimation. If the answer is no, the relative rank loss is increased by one. The resulting sum is divided by the number of all pairs.
- *Closest neighbor loss significance (CNLS)*[8] weighs, for every node A, the difference in measured latency between the closest neighbor according to ping measurements and the closest according to the estimations. The closest neighbor according to measurement, B, and the closest neighbor according to estimation, C, are determined, and their measured latencies to A is computed. The CNLS is then computed as the absolute difference of their distances divided by the absolute distance between A and B.

4.1 Experiment configurations

To verify the performance of the two chosen latency estimation techniques, we performed estimation experiments repeatedly over a 10 day period and included 215 PlanetLab nodes (the total number of nodes we were allowed to access). The plots represent the results from Netvigator, Vivaldi, ping and tcpinfo using the same nodes over the same time interval. The reference for our latency estimations was established by measuring latencies between all pairs of nodes using ping once per minute.

Descriptions	*Configurations*
Vivaldi group sizes	4, 8, 12 clients
RTT measures	*tcpinfo*, *ping*
Packet rates	high (100 packets/sec.), low (2 packets/sec.)
Log times	4, 8, 12, 16, 20 minutes

Table 2. Vivaldi experiment configurations.

For the Netvigator experiments, we used publicly available estimates done on PlanetLab. Netvigator is currently a running PlanetLab service that estimates the link latencies between nearly every PlanetLab node. The results of the measurements are available from the S^3 website*, which is updated every four hours with the most recent estimations. We used these measurements in our experiments. However, the Netvigator configuration is currently a black box. Furthermore, we tested Vivaldi on PlanetLab using a wide variety of configurations, looking at all combinations of parameters that are shown in table 2. Vivaldi group sizes determine the number of neighbors to which a node actually performs *RTT* measurements, where a higher number means more measurements and an improved estimation. Groups were formed randomly. The measurements themselves where performed once by looking at the *tcpinfo* structure that is updated for each open TCP connection and that can be used easily in passive measurements, and once by active measurements using *ping*. The *packet rate* was varied because a higher rate follows the actual latency development more closely, while it is also consuming more bandwidth itself, at least in the active measurements. The log times parameter determined for how long the Vivaldi information was collected until its estimations were used for identification decisions.

4.2 Experiment results

Figure 2 visualizes the discrepancy between *ping*-measured latency and the estimated latencies in greater detail. It is a scatterplot of all pairs of nodes that compares the absolute measured RTT with the absolute estimated RTT†. Measurements are sorted by *ping*-measured RTTs on the X axis, the estimations for the same pairs of nodes are Y-values, i.e., the ideal line is $y = x$. Vivaldi results are shown for two configuration to see the effects of the packet rate. The figures show that Netvigator is very accurate in its estimations, closely following the ideal line whereas Vivaldi has a bit more variation. They show also that Vivaldi, and to some degree also Netvigator, overestimates RTTs for the smaller actual RTTs, while it underestimates longer distances. When the actual RTTs are very small, the overestimations are relatively high, but the absolute deviation may still be acceptable for many applications.

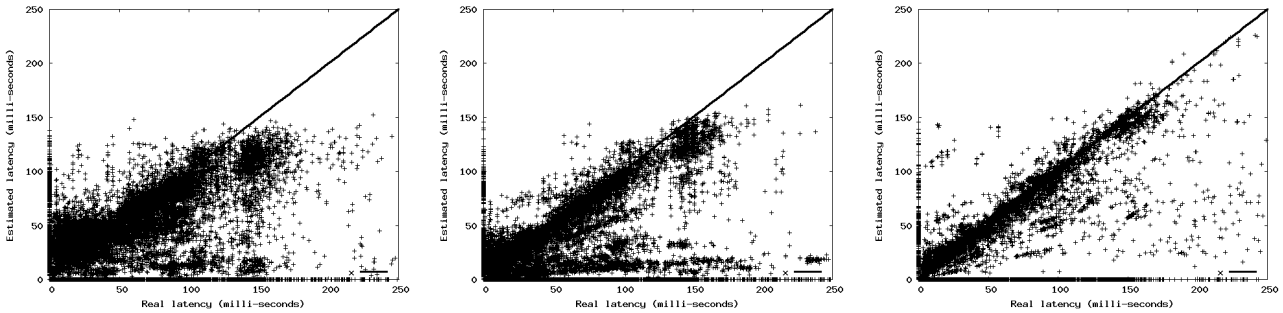
Furthermore, figure 3 shows how Netvigator and Vivaldi perform with respect to the three metrics that we introduced for identification performance. Figure 3(a) plots the CDF of the relative error between estimated and ping-measured latencies. Netvigator yields 80 % of the estimations within a 15 % relative error of the ping measurements and is clearly best. Vivaldi estimations are best in configurations with a high packet rate, and yields 80 % of the estimations within a 50 % relative error for a 4-minute log time. This is due to more probes and statistics for Vivaldi to measure from in a shorter time-span. Lower packet rates require a considerably higher log time to approach the relative error satisfyingly. Figure 3(b) plots the CDF of the RRL in the estimates. Netvigator has the lowest RRL, where 80 % of the estimates have a relative rank loss of 15 %. The Vivaldi configurations perform similarly with respect to RRL, and performs best with group size 8. Intuitively, larger group sizes should produce a lower relative rank loss. This phenomenon is apparently due to load on PlanetLab nodes, such that we restrict ourselves to group size 8. Figure 3(c) plots the CDF of the CNLS. Netvigator outperforms Vivaldi on this metric. The authors of Netvigator listed proximity estimation (i.e., rank nodes according to proximity to any given node) as the prime focus of Netvigator, and the results from CNLS shows that it works very well. Finally, we noticed that *tcpinfo* measurements always led to slightly worse results than *ping* measurements and ignore this parameter in the following experiments. We also saw that after 8 minutes of logging, the Vivaldi results had stabilized, so we do not plot the results for the longer intervals.

5. MANIPULATION ALGORITHMS

This step of the group management process simplifies the problem of building overlay routes that is solved in the construction step. For the SPT-based construction algorithm of our choice, the critical parameter is the choice of the root node.

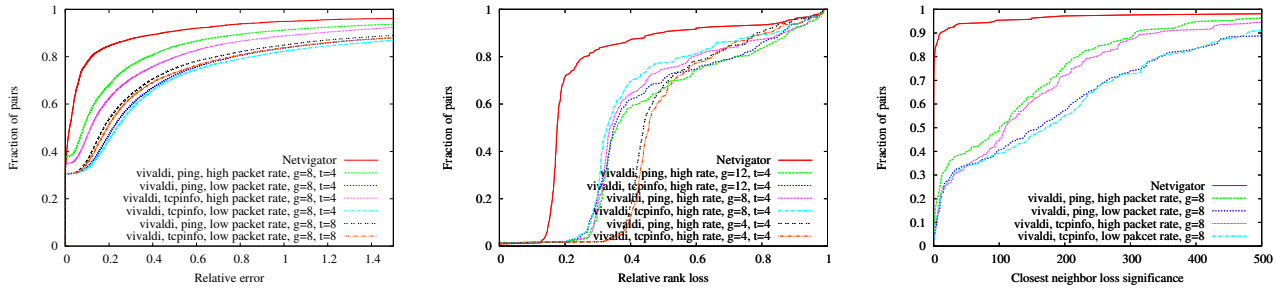
*http://networking.hpl.hp.com/cgi-bin/scubePL.cgi

†Note that there are several points in the plots that have 0-values. This is due to the fact that not all of the 215 nodes could talk to each other, and the reason why we use only those 100 nodes in the remainder of the paper that could communicate in a full mesh.



(a) Vivaldi, low packet rate, g=8, t=4 (b) Vivaldi, high packet rate, g=8, t=4 (c) Netvigator

Figure 2. Real versus estimated latency, where g is Vivaldi group size and t is log time



(a) Relative Error (b) RRL (c) CNLS

Figure 3. CDFs of Netvigator and Vivaldi performance. Vivaldi plots with different configurations, where is g is group size, t is log time

However, as the results from manipulation step are heavily dependent on the underlying network knowledge, we compare the quality of our core selection heuristic for latency estimations with *ping*-measured latencies. The heuristic that we use to select a core node is based on an evaluation of core nodes' *eccentricity*. Here, the eccentricity of a node is the maximum distance to all members of a set of nodes, where the distances to each member of the set is determined by the shortest-path latency between it and the core. The set of nodes in our evaluation was chosen by simulating regions-of-interest as described in section 1.

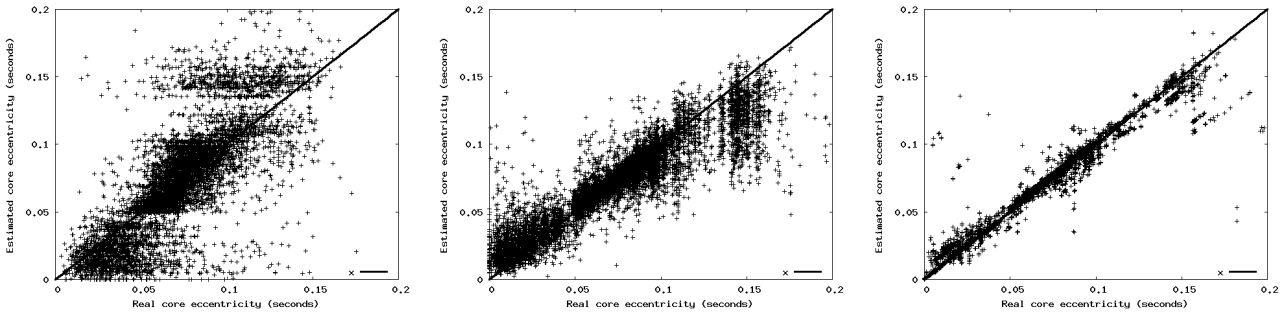
5.1 Experiment configuration

For the manipulation experiment on PlanetLab, we used 100 nodes. This limited number was due to many unstable PlanetLab nodes and end-to-end reachability issues. The tests were run for a 10 day period. The 100 nodes dynamically joined and left groups, i.e., simulating the users frequently changing their region-of-interest, throughout our experiments by sending join and leave requests to a central entity. The group popularity was distributed according to a Zipf distribution.

A central entity applied the *group center heuristic*[10] to find cores for groups that changed due to membership dynamics. In this process, it used a global graph of the member network, which was built using the latency estimation techniques introduced in section 4.

5.2 Experiment results

First, we investigate how the absolute eccentricity deviates between core selection based on all-to-all measurements and estimations. We call the eccentricity of cores nodes that are chosen from a subgroup using the group center heuristic based on all-to-all ping measurements the *real eccentricity*. The eccentricity of core nodes that are chosen from the same subgroup based on estimations is called the *reported eccentricity*. The actual eccentricity computation uses ping-measured data in both cases. We visualize the discrepancy between the reported eccentricities and the real eccentricities in figure 4. We showed in section 4 that Netvigator provided the best estimates, and this is also reflected in the results from the core selection heuristics. Using the latency estimations from Netvigator, the manipulation algorithms return more accurate results close to the optimal. Vivaldi provides poorer estimates at a low packet rate than a high packet rate, and this also gives inferior approximated core nodes. Finally, we can also see from the figures that the absolute amount of mis-estimations in both estimation techniques is mostly independent of the distances of nodes.



(a) Vivaldi, low packet rate, g=8, t=4 (b) Vivaldi, high packet rate, g=8, t=4 (c) Netvigator.

Figure 4. Reported eccentricity vs. real eccentricity from cores to group members.

6. CONSTRUCTION ALGORITHMS

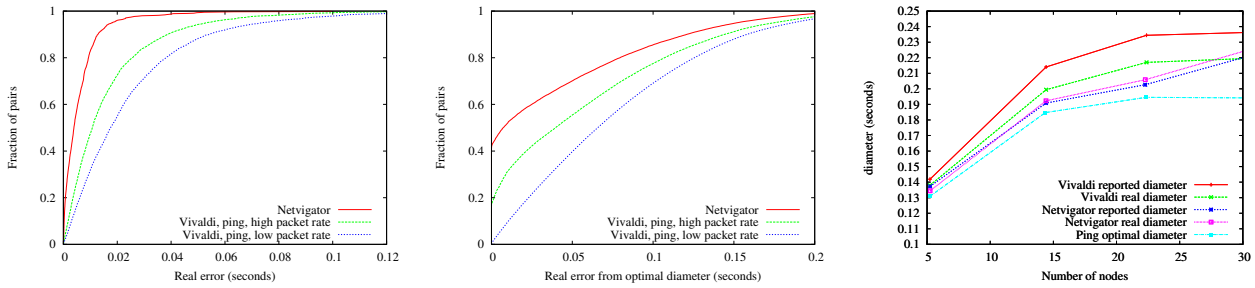
The construction step is the third and last step and the one that actually builds the trees that group nodes for all-to-all communication. The construction algorithm used for this exemplary investigation of the effects of latency estimation relies on a central entity. The central entity takes as input the graph that is created in the identification step in section 4 and a root node that is chosen in the manipulation step in section 5. The construction process does then use the tree algorithm *dl-SPT* to a tree for each group whenever group membership changes. We evaluated the quality of the trees that were created based on estimates from Vivaldi and Netvigator by examing the resulting *diameters*. The *diameter* expresses the worst case latency between any pair of group members, and is a particularly important metric when all-to-all group communication is required by an interactive application. Since applications make decisions based on the assumed or measured diameter, it is not only important that it is low but also that the application can know it with a high degree of trust. We look at and compare diameters from three different angles in our evaluation:

- The *reported diameter* is the tree diameter that is obtained by using the estimated latencies in all steps. That is, by running manipulation and construction step based on the results of the identification phase.The value of the reported diameter is an estimated value.
- The *real diameter* is the tree diameter that is obtained by applying the path of the reported diameter on the close-to-real all-to-all ping measurements. Thus, the real diameter is obtained by running the manipulation and construction steps based on estimation. The value of the real diameter, however, is calculated using the close-to-real all-to-all ping measurements on the reported diameter path.
- The *optimal diameter* is the tree diameter that is obtained by using the close-to-real all-to-all ping measurements in all steps. That is, by running manipulation and construction step based on the all-to-all ping measurements. The optimal diameter is taken from these measurements.

6.1 Experiments configuration

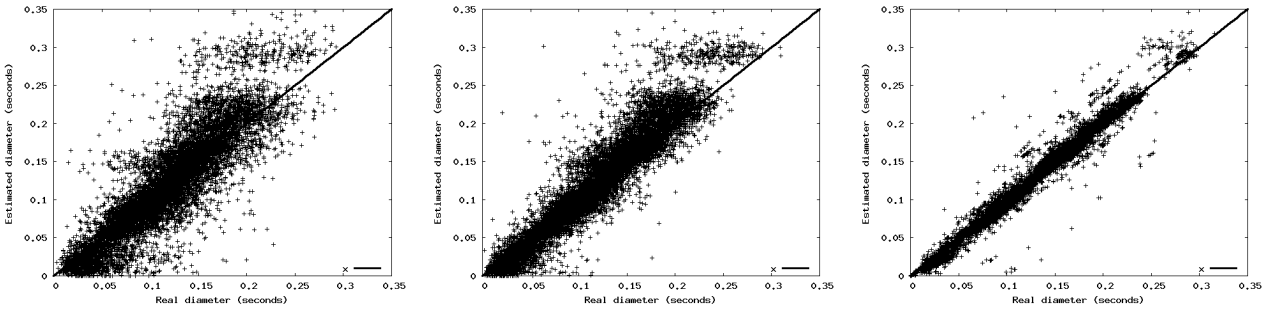
The construction experiments use *dl-SPT* and a degree limit of 5 for all nodes, which is due to somewhat limited client capacities in the Internet.[13] Furthermore, we use the same 100 PlanetLab nodes that were used for the manipulation experiments in section 5.1. The tests were run each day for a 10 day period. In general, a network of 100 PlanetLab nodes and group sizes below 40 do not give a good enough foundation for a conclusive evaluation of our tree algorithms. However, firstly we are primarily testing the applicability of latency estimation techniques and not the performance of our tree building algorithms. Secondly, we have performed the experiments on BRITE-generated graphs of size 1000, and the results correlate largely with our findings on this PlanetLab network with fewer nodes.

Our distributed application mimics group communication in distributed interactive applications (e.g., online games). The 100 nodes join and leave groups dynamically throughout our experiments by sending join and leave requests to a central entity. The group popularity is distributed according to a Zipf distribution. The central entity uses the latency measurements or estimations to choose the most appropriate core node using the group center heuristic according to section 5, and recomputes the multicast tree for the group whenever membership changes.



(a) CDF of error between reported diameter and real diameter (b) CDF of error between real diameter and optimal diameter (c) Diameter of several groups comparing estimated and real latencies

Figure 5. Tree metrics for diameter



(a) Vivaldi, low packet rate, g=8, t=4 (b) Vivaldi, high packet rate, g=8, t=4 (c) Netvigator

Figure 6. Reported diameter vs. real diameter in group trees

6.2 Experiment results

Figure 5(a) shows the CDF of the error between the reported tree diameter from the estimates and the real diameter. The group sizes were larger than 20 for increased confidence, and we exhausted the group membership combinations possible. *dl-SPT* applied to Netvigator estimates yields 96 % of the estimated tree diameters within a 20 milli-second error margin from the real diameter, while the estimates from Vivaldi with high packet rates yield 85 % of the tree diameters within a 25 milli-seconds error margin. Vivaldi exhibits the poorest performance with low packet rates, which yields 80 % of the tree diameters within a 35 milli-seconds error margin. However, the discrepancy between the reported diameter from the estimates and the real diameter is on average reasonably low for Vivaldi.

Figure 5(b) shows the CDF of the error between the real diameter and the optimal diameter. We see that the error margin between them becomes quite high. This may be due to a cumulative error effect when tree algorithms build a tree from the estimates.

Figure 5(c) shows the diameter produced from the estimates compared to the real-world for various group sizes using the tree algorithm *dl-SPT*. We observe that for group size 30 the difference between the reported diameter from the estimates and the real diameter is 5 milli-seconds for Netvigator and 15 milli-seconds for Vivaldi (high packet rate). However, the difference is bigger when compared to the optimal diameter. In that case, it is 25 milli-seconds for both Netvigator and Vivaldi.

In figure 6, we complete the evaluation of the estimates and visualize the discrepancy between the reported diameter and the real diameter. Not surprisingly, the same tendency that we saw from the core eccentricities emerge. Vivaldi with high packet rates is better than with low packet rates, but Netvigator performs better than both.

7. ALTERNATIVE CONSTRUCTION ALGORITHMS

We mentioned in section 1 that we have investigated many other graph and tree algorithms that are able to connect subgroups of members of an overlay network with small diameters. Since we have already shown the tendencies of tree construction based on estimated latencies, we show only briefly how the diameters of several algorithms are influenced.

Algorithm	*Meaning*	*MN aware* [1]	*Optimization*	*Constraints*	*Reconfiguration set R*	*Time complexity*
I-MDDL	Insert miminum diameter degree limited edge	no	diameter	degree	$\|R\| = 1$	$O(n^2)$
I-CN	Insert center node	no	diameter	degree	$\|R\| = 1$	$O(n)$
ITR-MDDL	Insert try reconfiguration and MDDL-edge	yes	diameter	degree	$\|R\| \leq d(m) * 2$	$O(n^2)$
RK	Remove keep as non-member node	no	shortest path	degree	$\|R\| = 3 \rightarrow d_T(m) \leq 2$	$O(n)$
			-	-	$\|R\| = 0 \rightarrow d_T(m) > 2$	-
RTR-MDDL	Remove try reconfiguration and MDDL-edge	yes	diameter	degree	$\|R\| \leq d(m) * 2$	$O(n^2)$
RTR-P	Remove try reconfiguration and prune	yes	diameter	degree	$\|R\| \leq \|E_T\|$	$O(n^2)$
mddl-OTTC	MDDL one time tree construction	no	diameter	degree	$\|R\| \leq \|E_T\|$	$O(n^3)$
dl-SPT	Degree limited shortest path tree	no	root eccentricity	degree	$\|R\| \leq \|E_T\|$	$O(n^2)$
SPT	Shortest path tree	no	root eccentricity	-	$\|R\| \leq \|E_T\|$	$O(n^2)$

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

Table 3. Algorithm descriptions and a set of properties.

In our studies of related work we found many existing tree heuristics that produce trees with a close-to-minimum diameter while constrained to a given degree limit.[2,14] However, these algorithms do not consider dynamic group membership, and stability is therefore usually not a goal. We consider reconfiguration time and stability to be highly relevant for our application area and evaluate algorithms that update trees dynamically whenever nodes are inserted or removed from a group. Furthermore, we consider it desirable to replace as few links as possible in a group reorganization because extensive changes in overlay networks lead to packet reordering and loss. Many tree algorithms are spanning tree algorithms,[2] which try to build a tree that covers all the group members.

7.1 Other heuristics

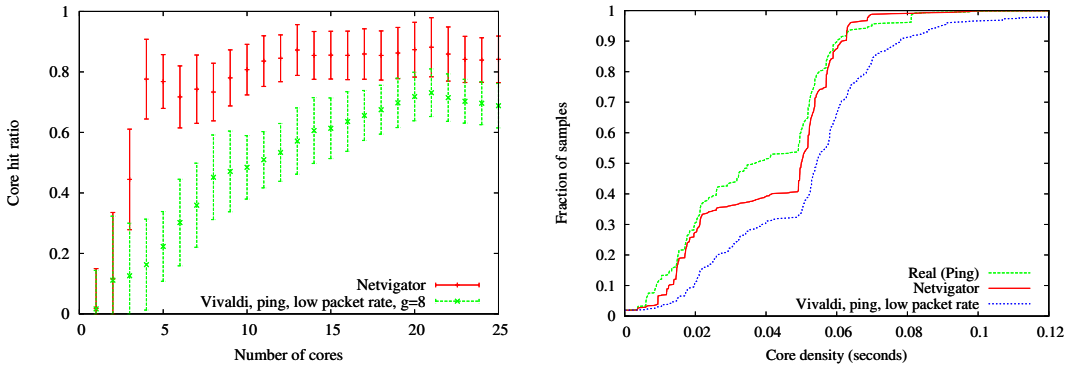
Table 3 provides a quick overview of abbreviations and features. An algorithm is non-member-node[‡] aware if it is able to insert or remove non-member nodes from a group tree. The reconfiguration set R contains the edges that are changed between reconfigurations of a group tree. Furthermore, the time complexity (in big-oh notation) is the worst-case number of computational steps of an algorithm.

Tree heuristics are spanning tree algorithms[2] that try to build a tree that covers all the group members. Whenever a node joins or leaves a subgroup, these heuristics have to rebuild the entire tree. The tree heuristic *dl-SPT* was used through the earlier sections of the paper. For completeness, we include also *SPT*[15] into our investigation, i.e. Dijkstra's shortest path algorithm without any degree bounds. This algorithm is very likely to create star topologies. Finally, we include *minimum diameter degree-limited one-time tree construction (mddl-OTTC)*. It is a heuristic of the minimum diameter degree-limited limited problem, and an alteration of OTTC,[16] which was introduced by Abdalla, Deo and Gupta. It is based on Prim's MST algorithm. mddl-OTTC adds a vertex that minimizes the diameter while obeying the degree limits.

Dynamic algorithms are algorithms that support reconfiguration of smaller parts of an existing tree.[1] Each dynamic algorithm is comprised of one *insert* and one *remove strategy* that can be combined freely. *Insert strategies* insert a new node m into an overlay group tree and assure that m can communicate with the other nodes:

- *I-MDDL* (minimum diameter degree limited) finds an edge for adding m that minimizes its eccentricity, and consequently the diameter of the new tree while obeying the degree limits. This strategy creates trees that have a clear center, where the total cost is generally higher.
- *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 eccentricity related to the 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.
- *ITR-MDDL* (try reconfiguration, minimum diameter degree limited) finds an edge from m to a node v that is already in the tree in such a way that it minimizes the eccentricity of m. Next, two sets of nodes are created, one containing nodes that are necessary in the new tree, and one containing nodes that are optional to the new tree. The first set

[‡]A non-member-node is a node that is not member of the group itself, but that participates in the overlay network to distribute the data, e.g., proxies or super nodes.



(a) Ratio of optimal core selection hits (b) CDFs of core density. Eccentricity between cores

Figure 7. Netvigator and Vivaldi performance in core selection. Searching entire network.

contains all direct neigbours of v, v and m, the other contains a non-member node that has the shortest average distance to all nodes in the first set. Then, ITR-MDDL removes all the edges that interconnect the nodes of the two groups and finds the minimum diameter tree that connects the nodes in the first set with each other. The nodes in the optional set may be used if the diameter is reduced. The operation is basically a heuristic of the minimum diameter degree limited problem.

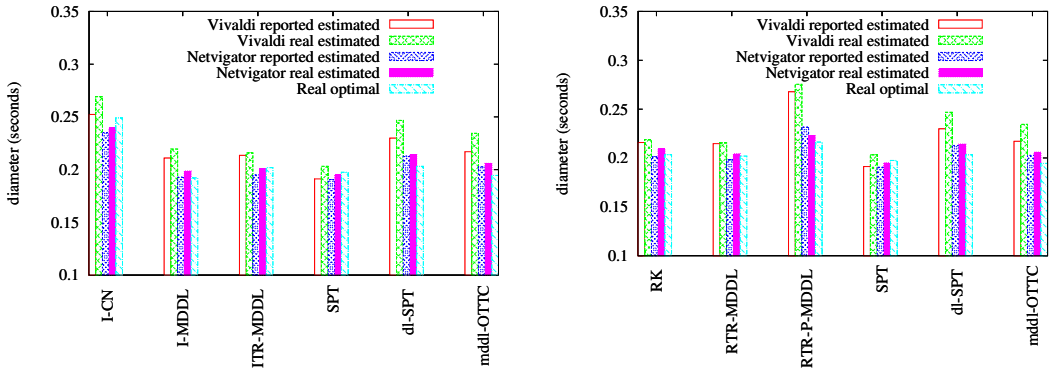
Remove strategies remove a member m from the multicast tree while assuring that the group members stay connected.

- *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. We have previously shown that it can degrade for larger groups.[17]
- *RTR-MDDL* (try reconfiguration, minimum diameter degree limited edge reconnects) is similar to ITR-MDDL. It creates a mandatory set from all direct neighbours of m in the tree, and an optional set containing m and a non-member node node that has the shortest average distance to all nodes in the mandatory set. Then, it removes all edges and reconnects the mandatory nodes as in ITR-MDDL.
- *RTR-P* (try reconfiguration, prune non-member nodes, minimum diameter degree limited edge reconnects) is similar to RTR-MDDL. However, the mandatory set does not only contain the direct neighbours of m, but also all nodes that are connected to m only through non-member nodes. This remove algorithm leads to less stable trees because a large number of nodes can be candidates for reconfiguration. Its advantage is that it reduces the number of non-member nodes in a tree.

7.2 Selecting several cores

The first major difference between several of the strategies discussed here and *dl-SPT* is that they the constructed trees can include nodes that are not members of the subgroup. The number of nodes that is eligable for acting as non-member nodes must be small, or the performance of the algorithms degrades. The selection of the appropriate nodes is therefore part of the *manipulation* step. Here, we select multiple cores instead of just one.

We have used the group center heuristic to search for a set of cores rather than just one beginning with the graph median node. Figure 7(a) plots the core selection hit ratio for Netvigator and Vivaldi (ping RTT, low packet rate, g=8) when searching for optimal cores among the 100 nodes. It is clear that Netvigator yields better estimates for use in core search and stabilizes around 80 % core selection hit ratio quickly. The group center heuristic struggles to find the optimal cores when applied to Vivaldi estimates, especially when the number of cores below 10. Furthermore, the core density, plotted in figure 7(b), compares the optimal eccentricity and the real eccentricity between the core nodes. We see that Netvigator yields the best estimates, where the eccentricities are very close to optimal. The Vivaldi estimates are not as good, but still most of the cores are within 20 milliseconds of the optimum, i.e., which may be good enough in many scenarios.



(a) Remove strategy is RTR-MDDL. (b) Insert strategy is ITR-MDDL.

Figure 8. Comparison of reported, real and optimal diameter (group sizes 20-30).

7.3 Comparing diameters

Figure 8 shows the tree diameter performance of the dynamic algorithms and the tree algorithms when applied to Netvigator and Vivaldi estimates, as well as the real all-to-all pings. We include SPT results mainly for the sake of comparison. Generally, we see that trees produced from Netvigator estimates have a closer to the real-world reported diameter than those from Vivaldi. The insert and remove strategies that search for well-placed nodes to use as cores (RTR-MDDL, RTR-P and ITR-MDDL) do not perform well when applied to Vivaldi estimates. In particular, RTR-P shows a clear tendency in disfavor of using the Vivaldi estimates, independent of the chosen insert strategy. RTR-P aggressively prunes non-member nodes and searches for new well-placed nodes to use as cores. We see that the lower quality latency estimates from Vivaldi have an impact on the performance of RTR-P.

It is also important and remarkable that estimates of the diameter are in all cases lower than the diameter that is actually achieved. This is supported by the observation made already in figure 2, which shows visually that more latencies are under- than over-estimated. It forces applications relying on estimation techniques to make conservative assumptions about the diameter of subgroups whenever the delay limits for the application are hard.

8. CONCLUSIONS AND FUTURE WORK

Various overlay construction algorithms assume a full knowledge of the underlying network. However, achieving full, up-to-date knowledge of the network requires monitoring traffic that grows quadratically with the number of nodes. This scalability problem is addressed by techniques that estimate link latencies and costs, but the trade-off is their accuracy, i.e., which again can severly reduce the correctness and quality of the resulting overlay tree.

In this paper, we conducted an experimental analysis of the latency estimation techniques Vivaldi and Netvigator in the context of dynamic group communication for distributed interactive applications. We evaluated the latency estimation quality and found that Netvigator yields estimations that are very close to optimal. The estimations from Vivaldi were not as good, but we found them to be usable in our application scenario. Vivaldi's advantages are that it is very easy to deploy in a peer-to-peer fashion, and it handles membership dynamics. Netvigator, on the other hand, needs an infrastructure (Landmark nodes). However, a game provider that controls a number of proxies could use Netvigator as it is the better alternative.

For future work, we plan to investigate the use of multiple central entities in a distributed proxy architecture[18] to further reduce the pair-wise latencies. For this, we investigate combining core selection heuristics to find core proxies and migration techniques to reduce the tree reconfiguration times. In addition, we are currently implementing and testing distributed tree algorithms where the tree construction is initiated from a central entity.

REFERENCES

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

[2] Vik, K.-H., Halvorsen, P., and Griwodz, C., "Multicast tree diameter for dynamic distributed interactive applications," in [*Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*], 1597–1605 (Apr. 2008).
[3] Dabek, F., Cox, R., Kaashoek, F., and Morris, R., "Vivaldi: a decentralized network coordinate system," in [*ACM International Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*], 15–26 (2004).
[4] Sharma, P., Xu, Z., Banerjee, S., and Lee, S.-J., "Estimating network proximity and latency," *SIGCOMM Comput. Commun. Rev.* **36**(3), 39–50 (2006).
[5] Claypool, M. and Claypool, K., "Latency and player actions in online games," *Communications of the ACM* **49**, 40–45 (Nov. 2005).
[6] International Telecommunication Union (ITU-T), "One-way Transmission Time, ITU-T Recommendation G.114," (2003).
[7] Andersen, D., Balakrishnan, H., Kaashoek, F., and Morris, R., "Resilient overlay networks," in [*ACM Symposium of Operating Systems Principles (SOSP)*], 131 – 145 (2001).
[8] Elmokashfi, A., kleis, M., and Popescu, A., "Netforecast: A delay prediction scheme for provider controlled networks," in [*IEEE Globecom*], (November 2007).
[9] Elmokashfi, A., *Scalable, decentral QoS verification based on prediction techniques and active measurements*, Master's thesis, Blekinge Institute of Technology, Blekinge, Sweden (Jan. 2007).
[10] Karaman, A. and Hassanein, H. S., "Core-selection algorithms in multicast routing - comparative and complexity analysis.," *Computer Communications* **29**(8), 998–1014 (2006).
[11] Narula, S. C. and Ho, C. A., "Degree-constrained minimum spanning trees," *Computers and Operations Research* **7**, 239–249 (1980).
[12] Lua, E. K., Griffin, T., Pias, M., Zheng, H., and Crowcroft, J., "On the accuracy of embeddings for Internet coordinate systems," in [*IMC '05: Proceedings of the 5th ACM SIGCOMM conference on Internet measurement*], 1–14, ACM, New York, NY, USA (2005).
[13] Vik, K.-H., Halvorsen, P., and Griwodz, C., "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).
[14] Shi, S. Y., Turner, J., and Waldvogel, M., "Dimensioning server access bandwidth and multicast routing in overlay networks," in [*International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*], 83–92 (June 2001).
[15] Goodrich, M. and Tamassia, R., [*Algorithm Design: Foundations, Analysis and Internet Examples*], John Wiley and Sons (2002).
[16] Abdalla, A., Deo, N., and Gupta, P., "Random-tree diameter and the diameter-constrained MST," Tech. Rep. CS-TR-00-02, University of Central Florida, Orlando, FL, USA (2000).
[17] Griwodz, C., Vik, K.-H., and Halvorsen, P., "Multicast tree reconfiguration in distributed interactive applications," in [*International Conference (NIME)*], 1219 – 1223 (Jan. 2006).
[18] Vik, K.-H., "Game state and event distribution using proxy technology and application layer multicast," in [*ACM International Multimedia Conference (ACM MM)*], 1041–1042 (2005).