TreeOpt: Self-Organizing, Evolving P2P Overlay Topologies Based On Spanning Trees

Peter Merz and Steffen Wolf*

Department of Computer Science
University of Kaiserslautern, Germany
{pmerz,wolf}@informatik.uni-kl.de

Abstract. We present a novel approach for self-organizing peer-to-peer overlays which enables the self-optimization of spanning tree topologies. We consider the minimum routing cost spanning tree problem which is known to be NP-hard and demonstrate that our proposed algorithm approximates a close lower bound even with minimal cooperation among the peers. This is achieved by evolving shortest path trees in which the root is allowed to move. We present results of simulations based on networks derived from Internet ping measurements. Moreover, we show that the algorithm can handle unannounced leaving as well as joining of nodes.

1 Introduction

Peer-to-peer (P2P) systems are self-organizing networks in the sense that they are capable of dealing with concurrent joining and leaving of peers without relying on central components. In this respect, P2P systems can be considered as complex dynamic systems comprised of autonomous interacting entities. A complex system may not only be self-organizing but also self-optimizing with respect to a given design goal such as communication performance. Today's P2P protocols like CAN [1], Chord [2], Pastry [3] or Gnutella [4] are self-organizing since they deal with the dynamics of joining and leaving of peers, but they lack the ability to optimize end-to-end message delays. Although systems like Gnutella or Pastry respect the underlying network by preferring communication links with low ping (round-trip) times, they do not optimize overall communication costs explicitly. Finding the best topology is a combinatorial optimization problem. Therefore, we focus on the optimization of P2P topologies by utilizing techniques from combinatorial optimization. As required by P2P systems, we focus on algorithms for self-optimization without relying on central components.

Viewing a P2P topology as a graph with the nodes being peers and the edges being communication links, the idea is to optimize the graph by minimizing the (expected) time for performing node-to-node communication. Since the graph should be easy to maintain as well as to repair and routing algorithms should be simple, we focus on spanning trees. The aim is to reduce the overall communication time between any pair of nodes in the network when all communication has to be routed over the edges of the spanning tree.

* Partially supported by the Rhineland-Palatinate Cluster of Excellence DASMOD.

We propose a distributed algorithm (TreeOpt) for optimizing spanning trees and show in simulation experiments based on Internet data, how much can be gained from this optimization. In particular, we demonstrate that TreeOpt is capable of improving spanning trees very fast, leading to a high gain in communication cost within few optimization rounds.

The paper is organized as follows. In Sect. 2, an overview is given for network optimization problems with focus on the minimum routing cost spanning tree (MRT) problem. A new distributed algorithm for tree optimization (TreeOpt) is presented in Sect. 3. In Sect. 4, results from experiments based on measured Internet data are shown. Conclusions and an outline for future research are provided in Sect. 5.

2 Network Optimization

A P2P overlay topology can be represented by a weighted graph $G = (V, E, d)$ where the nodes are the peers and the edges are communication links used for communicating in the P2P system. The weight $d(i, j)$ of an edge (i, j) denotes either the message delay on the communication link or the average round-trip-time (RTT) between peer i and peer j. A spanning tree is a subgraph $T \subseteq G$ with $|V| - 1$ edges connecting all nodes in V. Given a spanning tree, multicast and broadcast can be implemented simply by flooding. In this work, we focus on spanning trees as topologies for P2P overlays since they require a minimum number of edges (connections) and are easily maintained.

2.1 Problem Definition

In order to minimize the time for node-to-node communication, a tree with optimum routing cost is sought. The problem is defined as follows. Given a graph $G = (V, E, d)$, finding a spanning tree T of G such that

$$C(T) = \sum_{u,v \in V} d_T(u, v) \tag{1}$$

is minimal is called the *Minimum Routing Cost Spanning Tree (MRT) Problem*, where $d_T(u, v)$ denotes the distance/length of the path from u to v in T (message delay from u to v). The problem is a special case of the optimum communication spanning tree (OCST) problem [5], which is known to be NP-hard [6], but also known to admit a polynomial-time approximation scheme (PTAS) [7].

The MRT problem is closely related to the Shortest-Path-Tree (SPT) Problem. The SPT consists of shortest paths from a special root vertex to all other vertices. In fact, there is a vertex (the median vertex) such that any SPT rooted at this vertex is a 2-approximation of the MRT [8]. By minimizing the path from any vertex to the root, the cost function of the MRT can be also reduced. The following cost function

$$C_{\mathrm{SPT}}(T) = \max_{v \in V} d_T(v, r) \tag{2}$$

denotes the time or cost required for a multicast from the root r. The cost of a multicast from any other node is equal to or smaller than $2 \cdot C_{\mathrm{SPT}}(T)$.

The SPT problem can be solved in polynomial time with Dijkstra's algorithm [9] or the algorithm of Bellman and Ford [10,11]. However, resulting trees may degenerate to stars. Hence, in the context of P2P overlays, degree-constrained SPTs are desired where all nodes have at most k children. In P2P systems, k should be kept small since on failure of a peer, each of the k children has to find a new parent. However, if $k = 1$, we get a Hamiltonian path problem which is known to be NP-hard. Hence, we propose a distributed evolutionary heuristic for the degree-constrained SPT problem with $k > 1$.

2.2 Related Work

Only few research has been done on the optimization of network topologies, in particular overlay topologies, focusing on combinatorial optimization techniques. In [12], two combinatorial problems are addressed, a minimum spanning tree problem and a traveling salesman problem. In both cases, centralized algorithms are proposed. In [13], evolutionary algorithms for self-organized networks are proposed. However, the authors do not consider the MRT objective and it is unclear how their algorithms can work in a fully decentralized distributed system. A centralized approximation algorithm for application-layer multicast trees is presented in [14]. The approach concentrates on an NP-hard optimization problem in which each node has a nonnegative processing delay. The model also differs from our problem regarding the objective. The MeshTree approach [15] is a decentralized approach differentiating between backbone and delivery trees. Each node has a maximum neighbor degree. It is unclear how effective the optimization is in terms of approaching the optimum or a lower bound. Moreover, the benefits from using a minimum spanning tree (MST) as a backbone tree are not obvious. Since finding a degree-constrained MST is an NP-hard problem it can be expected to be harder for heuristic search than degree-constrained SPT.

2.3 Local Search for the MRT Problem

Local search is an effective heuristics in combinatorial optimization. It works by iteratively improving a current solution by searching the neighborhood of the current solution. If a better solution is found, it is accepted as the new current solution, otherwise a local optimum is found and the local search terminates. A neighborhood of a solution is defined by the set of solutions that can be obtained by slightly modifying the current solution. Such a modification, also called move, modifies the tree in a simple non-destructive way. Two types of moves considered in our algorithms are shown in Fig. 1. In the first move (a), a node i is connected to a new parent q. The second move (b) is actually a combination of two moves of type (a). Here, the nodes i and j exchange their parents.

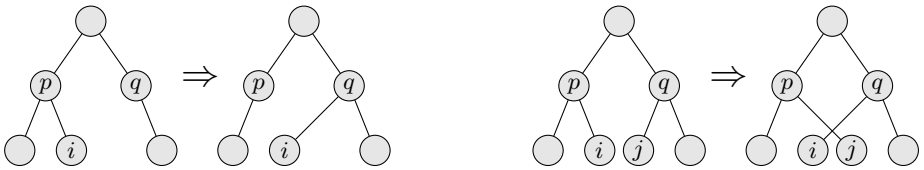

Fig. 1. Two types of tree moves

In order to compute the gain associated with a move, we make use of the following formula. The objective can be rewritten as:

$$C(T) = \sum_{v \in V} C_v(T) \quad \text{with} \tag{3}$$

$$C_v(T) = 2\, c(v) \cdot (N - c(v)) \cdot d(v, p^v) \tag{4}$$

Here, $N = |V|$, p^v denotes the parent of node v and $c(v)$ denotes the size of the subtree rooted at v (number of children +1). We assume that the tree has an arbitrary root. With (3) the cost of a tree can be computed in linear time. Also, (3) can be used to calculate the gain of a move efficiently. For both moves, only those C_v have to be recalculated that do change. These are those nodes for which the number of children or parent edges change and hence all nodes on the path P on the tree from i to parent q (or node j):

$$\Delta C(T \to T') = C(T) - C(T') = \sum_{v \in P} (C_v(T) - C_v(T')) \tag{5}$$

Hence, the number of updates is expected to be in $O(\log N)$ if the tree is balanced but in the worst case linear in N (if the tree degenerates to a single path).

The local search can be implemented in a distributed self-organizing way. However, calculating the gain of a move requires cooperation of many nodes in the network. More precisely, the number of nodes to cooperate is determined by the length of path P. Limiting the allowed path length and hence the number of cooperating nodes may reduce the effectiveness of the local search.

3 The Distributed TreeOpt Algorithm

The basic idea behind the TreeOpt algorithm is as follows. Rather than performing a local search on the MRT problem directly, we perform a local search to find shortest path trees with bounded degree. These trees are approximations to the MRT problem and can be found more easily in a distributed system. Compared to the local search for the MRT, the system requires a minimum amount of cooperation of the peers. Each peer tries to improve its situation in a greedy, selfish fashion. In order to reduce the time to repair the network in case of failures we bound the maximum number of children (degree) to a small constant value.

```
every delta time units do:
// Epidemic:
   p  := randomPeer(P);
   sendPeerList(p, P);
   P  := updatePeerList( receivePeerList(p) );
// Network repair:
   if parentLost and reconnectFails then root := true;
   if root then p := randomPeer(P);
                if Root(p) < me then connectTo(p); root := false;
// Tree improvement:
   updateAverageRTT();
   if root then p= childWithBestAvgRTT();
                if AvgRTT(p) < AvgRTT(me) then MoveRootTo(p);
           else dRoot(me) := dRoot(parent) + RTT(me,parent);
                p := randomPeer(P);
                probeForMove(dRoot, p);
```

Fig. 2. TreeOpt Algorithm. The three parts are shown: Epidemic, Tree improvement and Network repair

3.1 General Outline

After joining the network, each peer builds and maintains a peer list by using an epidemic algorithm [16]. The epidemic algorithm is extended such that in every round, data for the optimization is updated and an improvement step is executed in which the peer tries to improve its tree connections. The general outline of the algorithm is shown in Fig. 2. The purpose of the epidemic part is twofold. First, it increases robustness since it is unlikely that the overlay will be partitioned in case of failures. Second, for the improvement step it is essential that a peer can probe any other peer from the overlay (`randomPeer`). The peer list held by each peer does not need to contain all peers. In fact, only a relatively small number of peers with respect to the network size have to be known in order for the epidemic part to work [16].

3.2 Joining and Leaving

A node joins the overlay by connecting to a peer already in the network. The peer either accepts the joining peer as a child in the overlay tree or redirects it to one of its children randomly if its maximum number of children is reached.

The leaving of a node is handled by rejoining the children of the leaving peer. During joining each peer receives a list of other peers as usual in an epidemic algorithm as well as the estimated distance of the parent to the root (`dRoot`).

3.3 The Tree Improvement Step

In the TreeOpt algorithm the same moves are considered as in Fig. 1. In terms of overlay tree construction the first move (a) can also be seen as a node i trying to

connect to a new parent q. If node q accepts more children, and if the distance from node i to the root is reduced, the move is accepted. Denoting the distance from node x to the root in terms of the sum of the weights of the path by $d_r(x)$, the gain in cost of choosing node q as the new parent for i is

$$g(i,p,q) = d(i,p) + d_r(p) - d(i,q) - d_r(q) \tag{6}$$

If the gain is greater than zero, the move is accepted.

The second move (b) occurs only when node q already has the maximum number of children (2 in the figure). Node q may decide to accept the new child i and redirect its child j with maximum distance to itself if $d(i,q) < d(j,q)$. Hence, the distance to the root of node i is reduced, possibly at the cost of the other node j.

3.4 TreeOpt Messages

In each optimization step, a peer pings its current parent and retrieves its distance to the root. Moreover, the peer tries to improve its situation by looking for a new parent. If choosing the new parent leads to a shorter distance to the root, it tries to connect to the new parent. A peer only accepts a new child if the maximum number of children is not reached or if the candidate peer is closer to itself than some other child. In the latter case the child with the maximum distance is redirected to the (old) parent of the new child. Hence, each peer sends a ping message to a potential parent and receives the distance from the parent to the root as part of its pong response. The round-trip time of the request is measured and the peer can decide to connect to the new parent. If so, it sends a connect message and receives either an accept or a reject message from the new parent depending on the willingness to accept the new child. The peer may need to send a disconnect message to its old parent. If a parent accepts the new child it may send a redirect message to its child with the maximum distance. This child has to send a connect message to its new parent in turn. Hence, each move can be realized with six (if the peer is rejected) to nine messages (if the peer is accepted and another is redirected) per move.

In this algorithm each peer acts greedily to improve itself, either to reduce the distance to the root or the distance to its children. Hence, no further incentive mechanisms are required for the optimization to work in a P2P environment.

3.5 Moving the Root

In the algorithm above, the root node of the tree is the first node joining the overlay. However, if the root is not a central node within the overlay, the resulting SPT will not approximate the MRT very well. Therefore, we allowed the root to move to the center of the graph. This is accomplished as follows. Each node estimates its average distance to the other nodes by summing up all ping (RTT) times and counting their numbers. Every m time steps (rounds), the root node considers to pass the root role to one of its children. If there is a child with

an average distance estimate smaller than the current root, this child becomes the new root. Both nodes (old and new root) update the 'distance to the root' estimate. This way the root gradually moves to the center of the graph.

In order for this mechanism to work, peers are required to cooperate. If they do not, the root may not move at all. In this case the overlay still evolves over time – however, to a suboptimal configuration.

3.6 Self–repair and Network Partitioning

If a peer chooses a new parent, the resulting graph may no longer be a tree. This is the case if the new parent is in a subtree of the peer. Then, a cycle is introduced and the graph has disconnected components. To prevent this scenario, the peers may gather information about the path to the root and check it for cycles. This information might be outdated, so the cycle could remain unnoticed for a short period of time. An alternative is to tolerate such a move. TreeOpt is able to repair the graph by itself since a cycle leads to a permanent increase of the distance to the root of all nodes within the cycle. Hence, these nodes tend to choose alternative parents closer to the root in subsequent moves, breaking the cycle and reconnecting the graph.

Still, the unannounced leaving of a node which is not a leaf of the tree will result in a network split. When a peer notices that the connection to its parent is lost, it should assume such a split and start a repair mechanism. We propose a simple mechanism to repair these kinds of network partitions which works on the data available in each node and does not need a global view.

The network repair mechanism is included in Fig. 2. As a first step after the partitioning is recognized, the children of the missing node try to reconnect to nodes they have been recently connected to. In most cases, this helps keeping the tree connected. However, if too many nodes fail, these orphans become roots of their subtrees. This also means that their estimated distance to the root is set to zero. The resulting network will then consist of a forest. In order to reconnect the trees, all root nodes try to find roots of other trees. Each root r randomly selects a node i and asks for its root node (denoted by r^i). If this gives another root node ($r \neq r^i$), the node r connects to node i. To ensure that this does not result in a cycle, an arbitrary order has to be used, so only one direction of connection is allowed. If all nodes have unique ids, the node with a higher id can connect to a node with a lower root node's id ($r > r^i$). The result is the merging of two trees, but it will probably not represent a good overlay tree in respect to the routing cost.

The mechanism can take more than one round to repair the network, simply because it is a stochastic algorithm. So in some cases the network will remain partitioned for more than one round of the algorithm. TreeOpt will still optimize the two trees, but the routing cost for such a partitioned network is undefined, keeping in mind that it is the sum of all communication costs between all node pairs using only the edges of the tree(s). Therefore, the algorithm aims to reduce the time to repair the network.

4 Evaluation

We performed several experiments to study the effectiveness of tree-based optimization in P2P overlays. In the experiments we used real world data from two sources. The first is PlanetLab [17], a world-wide platform for performing Internet measurements. Here we used the round-trip-time (RTT) of messages from any host to any other host as the communication cost for the edges in the overlay graph. From the daily measurements reported in [17] we used the first measurement for each month of the year 2005 (denoted by `mm-2005`). Those networks consist of 70 to 419 nodes. A larger network with 2500 nodes was taken from the Meridian project [18]. All results given are averaged over 64 runs.

Since we are interested in self-organizing overlays, it is important that the local search can be implemented in a distributed algorithm. Therefore, we investigated the importance of cooperation in the local search for the MRT by restricting the maximum path length of the moves. Preliminary experiments showed that for some PlanetLab networks this path length has to be as high as 10 in order to arrive at near optimum solutions. These results motivated to find alternative distributed algorithms for the MRT based on shortest path trees.

4.1 Local Search for the MRT vs. SPTs

In a first set of experiments, we focused on finding good solutions for the MRT based on global view optimization. Table 1 shows for selected networks their size (Size), the best solution (Best) found by an evolutionary metaheuristics [19], the all pairs shortest path lower bound (APSP), and an upper bound (SPT). The lower bound measures the overall communication cost if we consider the shortest path tree for each node instead of a single shared tree as in the MRT. The upper bound is the best of all possible shortest path trees in terms of our cost function. This tree is a 2-approximation to the optimum solution [8]. The table shows that there are shared trees which are very close to the lower bound, all deviations are less than 14 %. Hence, maintaining a multicast tree for each sender does not pay off compared to a single shared tree, since the overhead for maintaining n trees compared to a single tree is too high. The table also indicates that single shortest path trees can be used to find good approximate solutions since the gap to the best MRT solutions are between 4 % and 14.5 %.

4.2 SPT based TreeOpt

In the next set of experiments, we focused on approximating the MRT problem by using shortest path spanning trees. Since these trees may degenerate to stars, we limited the number of children in the tree to a predefined degree k. Experiments not discussed here showed that for degree $k = 5$ the performance is almost as good as for degree $k = 21$ or higher. Hence, the MRT can be effectively approached using SPTs with a maximum degree of $k = 5$.

To study the influence of the moving root as described above, we allowed the root to move every $m = 8$ time units. In Table 2, the percentage excess above the

Instance	Size	Best	Lower Bound (APSP)	Upper Bound (SPT)
01-2005	127	2 743 556	2 447 260 (12.1 %)	3 025 120 (10.3 %)
02-2005	321	17 567 152	15 868 464 (10.7 %)	19 607 936 (11.6 %)
03-2005	324	19 288 320	17 164 734 (12.4 %)	20 842 606 (8.1 %)
04-2005	70	751 404	663 016 (13.3 %)	787 856 (4.9 %)
05-2005	374	19 175 890	17 324 082 (10.7 %)	19 949 036 (4.0 %)
06-2005	365	20 312 884	18 262 984 (11.2 %)	22 217 312 (9.4 %)
08-2005	402	30 540 984	27 640 136 (10.5 %)	32 209 226 (5.5 %)
09-2005	419	25 712 960	23 195 568 (10.9 %)	27 223 336 (5.9 %)
11-2005	407	26 797 284	23 694 130 (13.1 %)	29 322 480 (9.4 %)

Table 1. Overview of some of the considered instances. For each instance, the network size, the best solution found, the All-Pairs-Shortest-Path lower bound and an upper bound based on the Best Shortest Path Tree are shown.

	Static root		Moving root	
Instance	Gap (%)	G	Gap (%)	G
01-2005	36.09	5.3	20.97	6.2
02-2005	59.42	4.4	18.18	5.6
03-2005	68.81	4.0	15.77	6.2
04-2005	66.78	2.4	8.02	3.8
05-2005	40.08	6.4	12.85	8.4
06-2005	65.86	4.9	16.02	7.1
08-2005	61.36	4.0	13.74	5.9
09-2005	45.45	5.9	15.97	7.8
11-2005	50.55	4.9	17.02	6.3

Table 2. Influence of the moving root: Excess and gains for some PlanetLab networks.

best known solution (Gap) as well as the factor G as the quotient of a random tree and the evolved tree with degree 5 after $10\,n$ rounds (n denotes the network size) are displayed for the case with a random static root and the case where the root is allowed to move. The results demonstrate the importance of adapting the root of the tree during the evolution. In case of `04-2005`, moving the root reduced the gap to the best known solution from 66.78 % to 8.02 %. Moreover, the results indicate that the cost of the tree can be reduced by a factor of up to 8.4 for the considered networks using the TreeOpt algorithm. The moving root scenario clearly dominates the static case.

4.3 Dynamic Networks

In P2P environments peers can be expected to join and leave at all times. To show that TreeOpt can cope with joining and leaving nodes we present results from experiments where a certain percentage of nodes suddenly leaves or joins the network. In these experiments we used the Meridian network with 2500 nodes provided by [18]. To make the effect of the leaving nodes visible, we started with

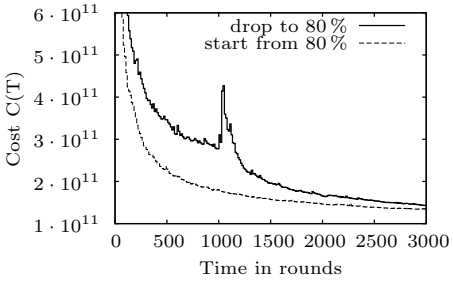

Fig. 3. Meridian network, 20 % nodes leaving at round 1000, compared to the network with only 80 % of the nodes

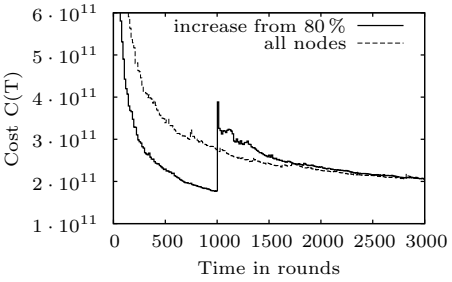

Fig. 4. Meridian network, 20 % nodes missing until round 1000, compared to the network with all nodes

a static network for an initial phase and scheduled the leaving of the nodes after TreeOpt found a good overlay tree. This way we can filter out other influences like moving roots and focus on the leaving and rejoining.

In a first set of experiments we were interested in long term effects of suddenly leaving nodes. We compare the Meridian network where 20 % of the nodes are scheduled to fail after 1000 time rounds to the same network where 20 % of the nodes are already left out and TreeOpt can optimize the routing cost from the beginning. Figure 3 shows the behavior of TreeOpt for both cases. After the loss in round 1000 the tree is repaired quickly, but the routing cost increases significantly. It can be seen though that both networks, then containing the same number of nodes, converge to the same routing costs. It has to be noted that the loss happened early while TreeOpt had not yet finished optimizing the networks.

In the opposite set of experiments we scheduled 20 % of the nodes to join at a later time (after 1000 time rounds) and compare this to the complete Meridian network which TreeOpt can optimize from the beginning. Figure 4 shows the results. Again a significant increase in the routing cost occurs when the new nodes connect to a random node. This time TreeOpt can optimize the tree faster than in the previous experiment. It can now start from an already good tree as a core, instead of a number of subtrees remaining from a formerly good tree.

In the last set of experiments we combine joining and leaving. Here, the network is scheduled to lose a certain amount of its nodes and at the same time receive the same number of new nodes. These nodes can be seen as new nodes or as the same nodes after deleting all knowledge about the network. Since the network size and all distances remain the same the routing costs before and after this event can be compared directly. Figure 5 shows the behavior of TreeOpt. A loss of only 1 % of the nodes can be repaired quickly. Even with heavy changes the network is reconnected after at most 40 rounds. However, the resulting tree is still bad in terms of the routing cost. The figure shows how long it takes to recover to a routing cost that is only slightly larger than the routing cost before the change (given as the excess in percent over this value). The figure shows that TreeOpt can recover to useful trees (about 40 % above the routing cost before

Drop/Join rate	Repair time
1 %	5 rounds
5 %	33 rounds
10 %	28 rounds
20 %	39 rounds

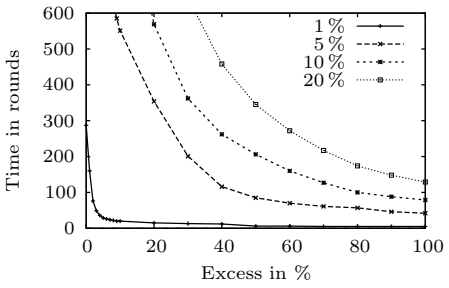

Fig. 5. Analysis of combined leaving and joining of 1, 5, 10 or 20 % of the nodes in the Meridian network. The time needed to repair the tree is given in the table, the times needed to re-optimize the tree to certain degrees are displayed in the plot.

the change) in a reasonable time. However, the larger the change the longer it takes to re-optimize the tree.

Summarizing, the TreeOpt algorithm based on dynamic degree constraint SPTs appears to be effective in optimizing the MRT objective as long as the network does not undergo heavy changes. Compared to a local search for the MRT it can be realized with minimum cooperation of the peers in a P2P overlay.

5 Conclusions

We have presented a distributed algorithm (TreeOpt) to optimize the routing cost in P2P topologies based on degree-constrained spanning trees. In experiments we demonstrated that TreeOpt is capable of approaching the theoretical lower bound provided by the sum of all pairs shortest paths. Here, a percentage excess of approx. 30 % is typical. In respect to the routing cost, we showed that compared to unoptimized trees, the communication cost/time can be reduced by a factor of up to 8.4 for a PlanetLab scenario with 374 nodes. Moreover, we demonstrated that in the presence of churn, TreeOpt is capable of repairing the tree(s) in short time while simultaneously (re-)optimizing the tree(s).

There are several issues for future research. First, the TreeOpt algorithm has to be evaluated in real PlanetLab experiments. Second, the scalability of the approach has to be evaluated for P2P overlays with millions of nodes. Since PlanetLab is relatively small, and real world data for such large networks is not available, the data has to be extrapolated preserving all relevant properties.

Also, TreeOpt makes heavy use of ping measurements. To achieve optimum performance, each peer has to perform $N = |V|$ pings. To reduce this message overhead drastically, a coordinate system may be introduced on the overlay as described in [20,21]. With this mechanism, peers may compare their coordinates instead of measuring the distances via pings. The coordinates may be propagated via the epidemic information exchange and hence peers may learn much faster about their neighborhood. We are currently working on a network coordinate version of TreeOpt as well as on generation of network coordinates.

References

1. S. Ratnasamy, P. Francis, M. Handley, *et al.*, A scalable content addressable network, Tech. Rep. TR-00-010, Berkeley, CA (2000).
2. I. Stoica, R. Morris, D. Karger, *et al.*, Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications, in: Proc. of SIGCOMM-01, Vol. 31, 4 of Computer Communication Review, ACM Press, New York, 2001, pp. 149–160.
3. A. Rowstron, P. Druschel, Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems, 18th IFIP/ACM International Conference on Distributed Systems Platforms (2001) 329–350.
4. M. Ripeanu, I. Foster, A. Iamnitchi, Mapping the Gnutella Network, IEEE Internet Computing Journal 6 (1) (2002) 50–57.
5. T. C. Hu, Optimum Communication Spanning Trees, SIAM Journal of Computing 3 (3) (1974) 188–195.
6. D. S. Johnson, J. K. Lenstra, A. H. G. R. Kan, The Complexity of the Network Design Problem, Networks 8 (1978) 279–285.
7. B. Y. Wu, G. Lancia, V. Bafna, *et al.*, A Polynomial-Time Approximation Scheme for Minimum Routing Cost Spanning Trees, SIAM Journal of Computing 29 (3) (1999) 761–778.
8. B. Y. Wu, K.-M. Chao, C. Y. Tang, Approximation Algorithms for Some Optimum Communication Spanning Tree Problems, Discrete Applied Mathematics 102 (3) (2000) 245–266.
9. E. W. Dijkstra, A Note on Two Problems in Connexion with Graphs, Numerische Mathematik 1 (1959) 269–271.
10. R. E. Bellman, On a Routing Problem, Quarterly of Applied Mathematics 16 (1958) 87–90.
11. L. R. Ford, Jr., D. R. Fulkerson, Flows in Networks, Princeton Univ. Press, 1962.
12. A. Sobeih, J. Wang, W. Yurcik, Performance Evaluation and Comparison of Tree and Ring Application-Layer Multicast Overlay Networks, in: Proc. of ICENCO'04, Cairo, Egypt, 2004.
13. K. A. Lehmann, M. Kaufmann, Evolutionary Algorithms for the Self-Organized Evolution of Networks, in: H.-G. Beyer, *et al.* (Eds.), Proc. of GECCO 2005, Vol. 1, ACM Press, Washington DC, USA, 2005, pp. 563–570.
14. E. Brosh, Y. Shavitt, Approximation and Heuristic Algorithms for Minimum-Delay Application Layer Multicast Trees, in: The 23rd International Conference on Computer Communications, IEEE INFOCOM, Hong Kong, 2004.
15. S.-W. Tan, A. Waters, J. Crawford, MeshTree: A Delay optimised Overlay Multicast Tree Building Protocol, Technical Report 5-05, University of Kent (2005).
16. P. T. Eugster, R. Guerraoui, A.-M. Kermarrec, L. Massoulie, From Epidemics to Distributed Computing, IEEE Computer 37 (5) (2004) 60–67.
17. S. Banerjee, T. Griffin, M. Pias, The Interdomain Connectivity of PlanetLab Nodes, in: C. Barakat, I. Pratt (Eds.), Proc. of PAM 2004, Vol. 3015 of Lecture Notes in Computer Science, Springer, Antibes Juan-les-Pins, France, 2004.
18. B. Wong, A. Slivkins, E. G. Sirer, Meridian: A lightweight network location service without virtual coordinates, in: Proceedings of the ACM SIGCOMM, 2005.
19. P. Merz, S. Wolf, Evolutionary Local Search for Designing P2P Overlay Topologies based on Minimum Routing Cost Spanning Trees, in: Proc. of PPSN X, 2006.
20. E. Ng, H. Zhang, Predicting Internet Network Distance with Coordinates-Based Approaches, in: Proceedings of IEEE INFOCOM, New York, 2002.
21. M. Costa, M. Castro, A. Rowstron, P. Key, PIC: Practical Internet Coordinates for Distance Estimation, in: Proc. of ICDCS'04, IEEE Press, 2004, pp. 178–187.