Guaranteed Greedy Routing in Overlay Networks

Dragan Milić, Torsten Braun
Institute of Computer Science and Applied Mathematics
Universität Bern, Neubrückstrasse 10, 3012 Bern, Switzerland
email: {milic|braun}@iam.unibe.ch phone: +41 31 511 2631

***Abstract*—Greedy routing can be used in mobile ad-hoc networks as geographic routing protocol. This paper proposes to use greedy routing also in overlay networks by positioning overlay nodes into a multi-dimensional Euclidean space. Greedy routing can only be applied when a routing decision makes progress towards the final destination. Our proposed overlay network is built such that there will be always progress at each forwarding node. This is achieved by constructing at each node a so-called nearest neighbor convex set (NNCS). NNCSs can be used for various applications such as multicast routing, service discovery and Quality-of-Service routing. NNCS has been compared with Pastry, another topology-aware overlay network. NNCS has superior relative path stretches indicating the optimality of a path.**

I. Introduction

Every overlay network can be visualized as a graph. In such a graph, each overlay node is represented as a vertex and each overlay link is represented as an edge. Those edges may be weighted, i.e. each edge can be assigned a weight corresponding to the round trip time (RTT) of the overlay link. In general, such a graph is a directed graph, because the fact that overlay node A has overlay node B as a direct neighbor, does not always imply that B has A as its direct neighbor.

Finding a route between two overlay nodes (usually end systems) in an overlay network can now be reduced to the problem of finding a cycle-free path between the two vertices representing those overlay nodes in the graph of the overlay network. We can quantitatively compare different paths between the same two overlay nodes by comparing their sum of weights (RTT stretch). The lower this sum is, the better is the path. Since the number of vertices and edges in such a graph is finite, the number of possible cycle-free paths between two overlay nodes is finite, too. The path with the lowest minimal RTT stretch is denoted as optimal path.

Unfortunately, we need to know the whole graph to find the optimal path. This poses a problem, since every overlay node has only a limited amount of memory to store the information of the complete overlay network. The size of the overlay network is therefore limited by the available memory for the routing protocol on each overlay node. Thus, in order to have a scalable overlay network, we must accept a trade off between optimality of found paths and memory requirements of the routing protocol. Ideally, the amount of memory required on each overlay node is constant and independent of the size of the overlay network. Another problem when complete network knowledge is needed is the overhead and delay to disseminate network and topology updates.

978-1-4799-0059-6/13/$31.00

Most of the currently existing structured overlay routing protocols [3], [17] require a constant amount of memory, but they totally disregard the RTT stretch of the found paths. Instead, they optimize the number of hops on the path. As a result, they find routing paths with only few hops, but possibly with a high RTT stretch. Different suggestions have been made to reduce the RTT stretch of the found paths and thus the approaches are made more topology-aware [14], [15]. Still, resulting paths are far from being optimal.

In this paper, we present an approach for overlay network routing, which takes advantage of embedding overlay nodes in a virtual space. Embedding overlay nodes into an Euclidean space, where RTTs between overlay nodes are represented as Euclidean distances, was originally introduced [2], [6], [8]–[10], [16], [18] as a method for scalable RTT prediction. We present applications of such an embedding in order to provide topology-aware unicast and multicast routing. We also present a way to provide locally bound flooding services to be used for service discovery.

In the following Section we present related work on RTT embedding, geographic routing and topology aware overlay networks. Section III discusses problems related to applying geographic routing protocols in overlay networks and presents nearest neighbors convex set (NNCS) overlay networks, which are easy to build and to maintain in a distributed manner. Greedy routing is always successful in NNCS overlay networks. In Section IV, we introduce applications of NNCS, Section V compares NNCS with Pastry concerning the path stretch in an overlay network. Section VI concludes the paper.

II. Related Work

A. Greedy Routing

Greedy routing allows us to keep routing decisions simple. It compares distances of each known neighbor to the destination. The neighbor nearest to the destination is chosen as the next hop of the message to be sent. This procedure is continued until the destination has been reached.

More formally, the procedure of greedy routing is the following: When an overlay node R with neighbors $\{N_1 \dots N_c\}$ with corresponding positions in the virtual space $\{\mathcal{C}_{N_1} \dots \mathcal{C}_{N_c}\}$ receives a message for destination D with destination position $\mathcal{C}_D$, it calculates the distances between its neighbors and the destination: $\{d_n | d_n := d(\mathcal{C}_{N_n}, \mathcal{C}_D)\}$. The neighbor N_m with d_m, the minimal distance to the destination, is chosen for the next hop.

What makes greedy routing problematic is the requirement that for each routing decision we have to choose a neighbor

that is closer to the destination. If we are unable to find such a neighbor, our routing protocol can end up in a loop and is never able to deliver the message to its destination. If we would be able to build an overlay network that guarantees that at each overlay node there is at least one neighbor closer to the destination than the current overlay node, greedy routing would always be successful. If we use just any overlay network such as [12] or [14], there is no guarantee that we will always be able to make "progress" toward the destination. In Section III, we will show that if overlay nodes have positions in a virtual Euclidean space, we can construct an overlay network in which the success of greedy routing is guaranteed.

Figure 1 illustrates an example of a message delivery path, where each node forwards a message for a destination to a neighbor node closer to the destination.

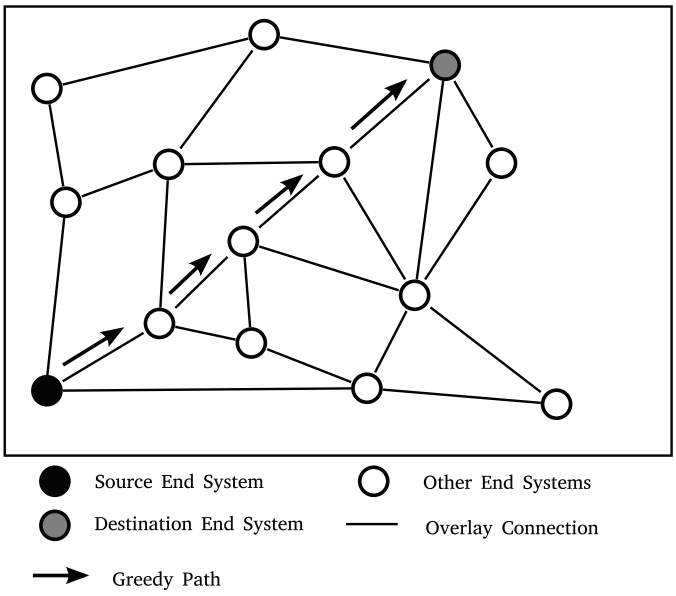

Fig. 1. Example of Greedy Routing in a Virtual Space

B. Greedy Routing in Mobile Ad-Hoc Networks

Mobile ad-hoc networks (MANETs) often use geographic node positions, which means that the number of dimensions is limited to two or three. In an overlay network, we assume that each overlay node can communicate directly with every other overlay node via the underlying network. In a MANET, each node's radio transceiver has a limited transmission range, which restricts its communication range to direct neighbors. Another difference is node mobility. In MANETs, every node can change its position at any time. In overlay networks, overlay nodes change their position in the virtual space only if there has been a significant change in the underlying network. Still, the basic problem remains the same: Each overlay node has limited memory, limited available bandwidth and knows only its own position as well as that of the destination node.

C. Topology Aware Overlay Networks

There are already different existing topology aware overlay networks. Most of them are designed to limit the number of routing hops and topology awareness is then added. Pastry [15] orders all overlay nodes in an one dimensional torus key space (a ring). Each overlay node chooses a random ID from the key space as a unique node address within the Pastry ring. Routing in Pastry is done similarly as in Plaxton routing [13]. Topology awareness of routing is ensured by preferring neighbors with smaller measured RTTs for the routing table.

III. Nearest Neighbors Convex Set (NNCS) Overlay Networks

A. Problems of Greedy Routing

The main problem of greedy routing in any kind of network is posed by non-convex "holes" in the network. The presence of non-convex holes leads to the failure of greedy routing. This problem is illustrated in Figure 2, where greedy routing decisions lead to a loop. The message is sent back and forth between two overlay nodes and never reaches its destination.

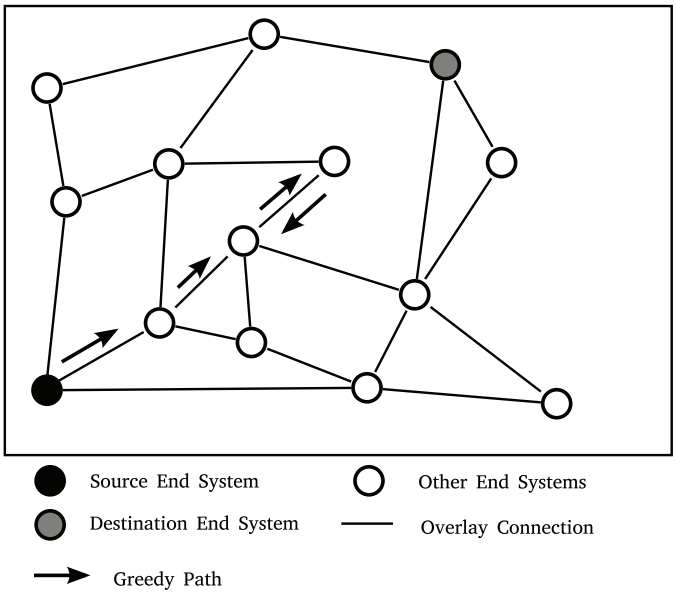

Fig. 2. Example Greedy Routing Failure Due to a Non-Convex Network Hole

To solve this problem, greedy routing schemes usually have a backup mode which is used when greedy routing fails, e.g., the Right-Hand rule in GPRS. The Right-Hand rule can only be used in two-dimensional graphs, but not in general Euclidean graphs with more than two dimensions. Fortunately, it is possible to construct an overlay network in such a way that the creation of non-convex holes is prevented. If we can guarantee that each overlay node can choose a relay closer to the destination, greedy routing will always work.

B. Requirements for Overlay Networks

Unicast routing in an overlay network is equal to finding a cycle-free path between two vertices. Each time an overlay node receives a unicast message that is not addressed to it, it has to forward it to one of its neighbors. In this Section, we assume that every overlay node is part of an overlay network and is embedded into a k-dimensional Euclidean space. In our approach, we will use the overlay nodes' position for the routing decision.

We also assume that each overlay node knows the location and underlying network address of its direct neighbors. This information is provided by the overlay network protocol implementation. One possibility for such an overlay network is NETICE9 [8]. We require that the overlay network protocol provides each overlay node with the knowledge of its direct neighbors and makes sure that this information is up to date. The overlay network protocol has to ensure that the overlay network is a connected graph. Otherwise, no routing protocol can ever find a path to every destination within the overlay network.

C. Nearest Neighbors Convex Set

In order for greedy routing to succeed, we must make sure that on each "hop" within the overlay network we are able to progress towards the destination overlay node. There are different methods to achieve this. One of them is to create a Delaunay triangulation of the overlay network, where greedy routing is always successful [1]. Constructing a Delaunay triangulation of an overlay network is non-trivial. Building a Delaunay triangulation in a distributed manner is possible, but overlay nodes need to cooperate and share states. The communication protocol and the algorithm required to achieve this may be complicated. We propose a far simpler approach involving the minimal neighborhood convex hull. If we can guarantee that for any possible destination position within the virtual space each overlay node has at least one neighbor that guarantees progress towards the destination, then greedy routing is always successful. We ensure that this property is fulfilled by requiring that each overlay node knows its nearest neighbors convex set (NNCS).

We define the nearest neighbors convex set (NNCS) as follows. For any given overlay node R with position $\mathcal{C}_R$ in the virtual space, its NNCS contains c overlay nodes $\{H_1, \ldots, H_c\}$ with positions $\{\mathcal{C}_{H_1}, \ldots, \mathcal{C}_{H_c}\}$ as neighbors of R. NNCS is the convex set defined by the intersection of half spaces S_j, $j \in \{1, \ldots, c\}$. Each half space S_j is bound by the hyperplane orthogonal to the line containing both $\mathcal{C}_R$ and $\mathcal{C}_{H_j}$ and containing $\mathcal{C}_R$ as depicted in Figure 3.

NNCS has following properties:

- NNCS of R contains R and its direct neighbors $\{H_1, \ldots, H_c\}$.
- Every other overlay node H_x that is not a neighbor of R ($H_x \notin \{H_1, \ldots, H_c, R\}$), is outside of R's NNCS.

Figure 4 presents an example of NNCS in two dimensions.

D. Greedy Routing in NNCS

NNCS guarantees that greedy routing makes progress (reduces the distance to the destination) at each hop. The mathematical proof for this statement can be outlined as follows.

Let us assume that some overlay node R receives a message M_D for overlay node D. Furthermore, greedy routing chooses its neighbor H_n as the next hop. If distance $d(\mathcal{C}_{H_n}, \mathcal{C}_D)$ is greater or equal than distance $d(\mathcal{C}_R, \mathcal{C}_D)$ greedy routing would not make any progress and hence it will not be successful. Now, if we closely inspect the properties of half spaces, we can see, that if a half space S_n defined by $\mathcal{C}_{H_n}$ and $\mathcal{C}_R$ with $d(\mathcal{C}_{H_n}, \mathcal{C}_D) < d(\mathcal{C}_R, \mathcal{C}_D)$ must contain $\mathcal{C}_D$, too. If we then revise the properties of an NNCS, we can see that in that case $\mathcal{C}_D$ is within R's NNCS. This also means that D is one of the neighbors of R. This contradicts to the assumption, that greedy algorithm would choose another neighbor, since $d(\mathcal{C}_D, \mathcal{C}_D) := 0$. Hence, our assumption is not correct. This means that every "hop" in the overlay network reduces distance to the destination and greedy routing is always successful in NNCS.

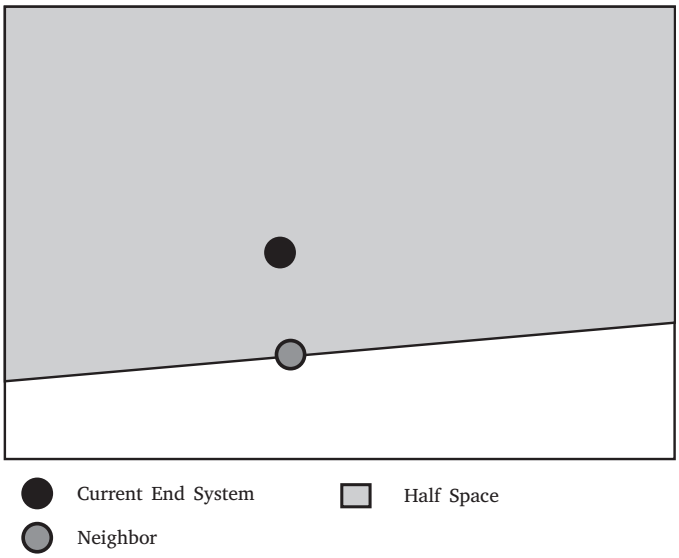

Fig. 3. Half Space Defined By One Neighbor

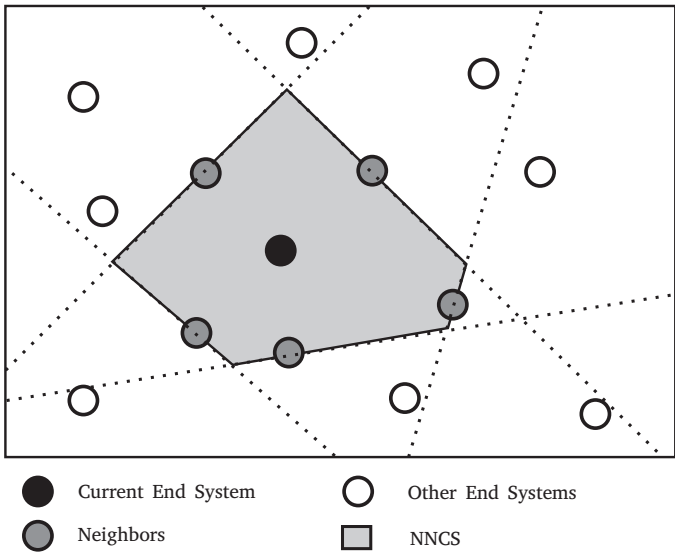

Fig. 4. Example of a Nearest Neighbors Convex Set (NNCS) in two Dimensions

1) Building a NNCS Overlay Network: Since overlay nodes do not have a global view of the overlay network, they must have means to discover and maintain their *NNCS*s. Since each overlay node R has its own *NNCS*, which is independent of *NNCS*s of other overlay nodes, maintenance can be done rather easily. As in any other overlay network, each overlay

node joining the network must know at least one overlay node already participating in the overlay network. At the beginning, a new overlay node N_n knows one overlay node B that is already participating in the *NNCS* overlay network. Since B is the only overlay node known to N_n, it puts B in its *NNCS*. After that N_n queries B about its *NNCS*. After receiving the *NNCS* of B, N_n uses Algorithm (1) to update its *NNCS*. This procedure is repeated until *NNCS* of H_n becomes stable (no new neighbors are taken into the *NNCS*). As soon as *NNCS* of H_n becomes stable, H_n periodically queries its neighbors about their *NNCS*s. If some of the neighbors repeatedly do not answer, H_n assumes that the neighbor has left the overlay network and removes it from its *NNCS*.

Algorithm 1 Algorithm for maintaining NNCS of an overlay node

Require: R_{NNCS}: NNCS set of R
  for $N \in R_{\text{NNCS}}$ **do**
    $N_{\text{NNCS}} \leftarrow query\_neighbor(N)$ {Query neighbor to get its current NNCS}
    for $N_i \in N_{\text{NNCS}}$ **do**
      if N_i is inside convex set defined by R_{NNCS} **then**
        $R_{\text{NNCS}} \leftarrow R_{\text{NNCS}} \cup \{N_i\}$ {add N_i to R_{NNCS}}
      end if
      $R_{\text{NNCS}} \leftarrow minimize\_nncs(R_{\text{NNCS}})$ {Remove all neighbors from R_{NNCS} that are not in the NNCS}
    end for
  end for
  return R_{NNCS}

Algorithm 2 Algorithm for finding minimal NNCS of an overlay node ($minimize\_nncs$)

Require: R_{NNCS}: NNCS set of R
Require: $\mathcal{C}_R$: Position of R in the virtual space
  $S \leftarrow construct\_halfspaces(\mathcal{C}_R, R_{\text{NNCS}})$ {Construct all halfspaces of every neighbor}
  for $N \in R_{\text{NNCS}}$ **do**
    $\mathcal{C}_N \leftarrow get\_position(N)$ {Get position of neighbor in the virtual space}
    for $H \in S$ **do**
      if $\mathcal{C}_N \notin H$ **then**
        $R_{\text{NNCS}} \leftarrow R_{\text{NNCS}} \setminus \{N\}$ {remove N from R_{NNCS} if coordinates of N are not contained in one of the half spaces}
      end if
    end for
  end for
  return R_{NNCS}

2) Making NNCS Overlay More Robust: Since every overlay network, where overlay nodes are involved, has a high churn each overlay node should not only know its NNCS but also have a possibility to quickly replace neighbors that left the overlay network. Also, there is a case where an overlay node, which is located on the "edge" of the virtual space, could have only one neighbor node in its NNCS. If this neighbor fails or leaves the overlay network, the overlay node will end up without any neighbors. In the worst case, this will lead to that overlay node isolated from the overlay network. To re-join the network, such an overlay node would need to re-iterate the bootstrap process. To avoid such situations, we propose each overlay node using two layers of NNCSs. The first NNCS layer is the NNCS of an overlay node. The second layer is the NNCS that is built when neighbors that are in NNCS layer 1 are ignored. Figure 5 illustrates this idea. By keeping track of two NNCS layers, each overlay node can quickly replace a neighbor from the first layer that is leaving the overlay network. At the same time both layers can be used to choose neighbors for greedy routing.

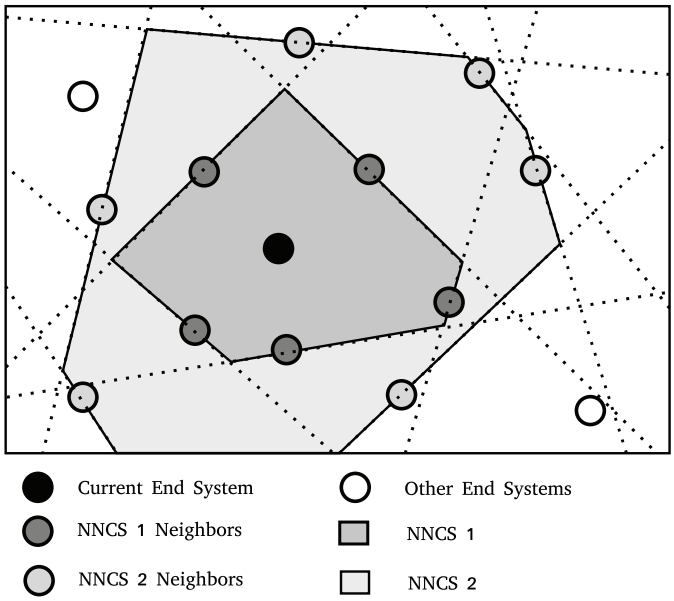

Fig. 5. Two Layers of NNCSs for one Overlay Node

3) Optimizing Routing: Using *NNCS* in an overlay network results in short routing jumps. This means that for each routing decision the message is sent to the nearest neighbor. This behavior may be satisfactory for routing on short distances. On long distances, this involves too many overlay nodes and unnecessarily delays message delivery. A much more desirable behavior would be that for distant destinations some more distant overlay node would be used for relaying the message. Since *NNCS* offers the choice of only nearest neighbors, we must broaden our choice in order to achieve this. In [12], we have described a Fisheye overlay network, which has ideal properties regarding the choice of neighbors in the sense of the choice of near and remote neighbors. Nevertheless, there is no guarantee that greedy routing is successful in the Fisheye overlay network. If we combine *NNCS* and the Fisheye overlay, we obtain an overlay network, which is both able to perform long routing jumps and where greedy routing is guaranteed to succeed. Hence, we propose maintaining both Fisheye overlay and *NNCS* overlay networks by allowing the greedy algorithm to make its choice from the set of neighbors containing both *NNCS* and Fisheye overlay neighbors.

IV. NNCS Applications

A. Multicast Routing

Unicast routing using a geographical routing protocol can be useful, but has a few drawbacks. One of the drawbacks is that the position within the virtual space of the destination

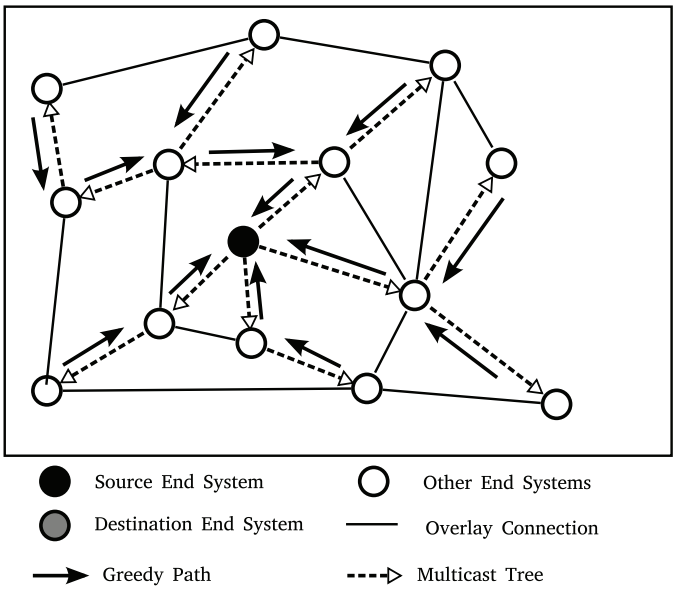

Fig. 6. Example of Building a Multicast Tree on Reverse Unicast Greedy Paths

must be known. As we have shown in [8] this position can change over time. Even if we use Global Network Positioning (GNP), the positions of overlay nodes will change as soon as there is a change in the topology of the underlying network. A more useful application of greedy routing is its use to construct multicast trees. The idea of multicast routing is the following: Sender S has a position in the virtual space $\mathcal{C}_S$. We assume that every overlay node participating in the overlay network is interested in receiving multicast data. There is one overlay network per multicast group. We also require that every overlay node participating in the overlay network not only stores and updates its own NNCS, but also keeps track of NNCSs of all other overlay nodes to which it is a neighbor. This information can be gathered, since every overlay node periodically queries its neighbors (as described in Section III-D) and with the query message it can also send its current NNCS. Every multicast message M_S from sender S contains the position $\mathcal{C}_S$ of the sender. This position is used to determine receivers of the message at each overlay node. When an overlay node R receives M_S, R delivers the message locally (since every overlay node is interested in receiving multicast traffic) and sends a copy of M_S to all overlay nodes that would route a unicast message towards S via R. In other words, multicast messages are flooded along the reverse unicast paths towards S. This approach is illustrated in Figure 6.

B. Service Discovery

One interesting application of a topology aware overlay network is to find a nearest service of some kind. This service discovery could be used to find a nearest overlay node containing a file copy. Another example is to find an overlay node with enough storage or CPU capacity to perform calculations with special requirements in a cloud computing environment. If using such services involves transferring large amounts of data, finding the nearest overlay node can have significant impact on the performance of the whole operation.

Finding such services is usually done by flooding the overlay network with request messages. Every overlay node matching the query (e.g. has the requested file) can then directly answer to the sender of the message. For example this is how Gnutella [5] is performing search queries. Of course, this way of searching in an unstructured overlay network is very inefficient, since flooding has no means of avoiding loops, meaning that there is a high probability that the same query is delivered to the same overlay node more than once.

Fortunately, there is a way to provide a similar operation in a topology aware manner. For this purpose, we use the algorithm described in Section IV-A to flood an NNCS overlay network. In analogy to multicast routing, we flood each overlay node with query messages along the reverse paths, i.e. along the path that overlay node would use to reach the sender of the query message.

Flooding begins at overlay node C, which sends one copy of query message Q_C to each of the overlay nodes that have C as its NNCS neighbor. Each of those neighbors H_n determine themselves which overlay nodes have H_n in their NNCS and would route a message with C as its destination through H_n. H_n sends one copy of Q_C to each of those neighbors. This procedure is repeated every time an overlay node receives Q_C as depicted in Figure 7. This chain of reverse forwarding terminates, when none of the overlay nodes that contain H_n in their NNCSs would route a message towards C through H_n.

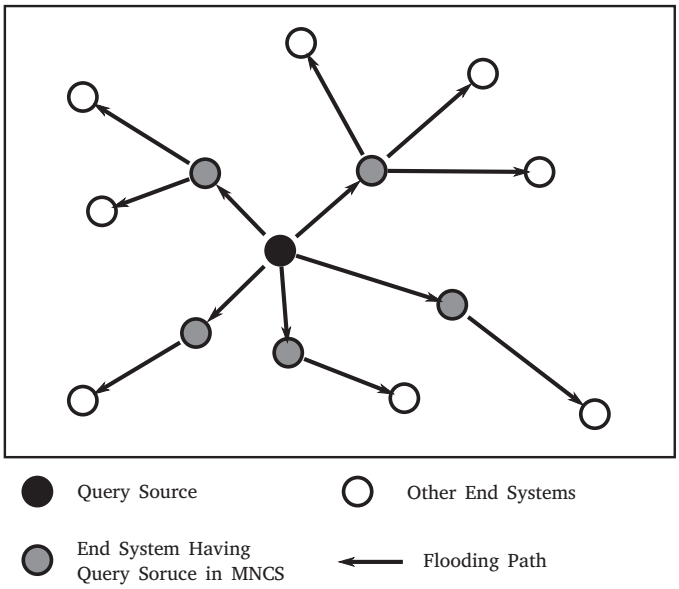

Fig. 7. Example of Query Flooding on Reverse Greedy Routing Path

Every overlay node also processes the received query and responds directly to C if it matches the query. Since we use flooding in a metric space, we might not be interested in overlay nodes providing the service and are very distant (beyond some predefined RTT r). Thus, we also may limit the radius of the query as presented in Figure 8. In this case, a query Q would be forwarded to an overlay node H only if the position H ($\mathcal{C}_H$) lies within the hyper ball with radius r around C. In other words, the distance $d(\mathcal{C}_H, \mathcal{C}_C)$ should be lower than r. By doing so, we reduce the load imposed on the overlay network, since we do not flood the whole overlay network, but only the part of the network that interests us.

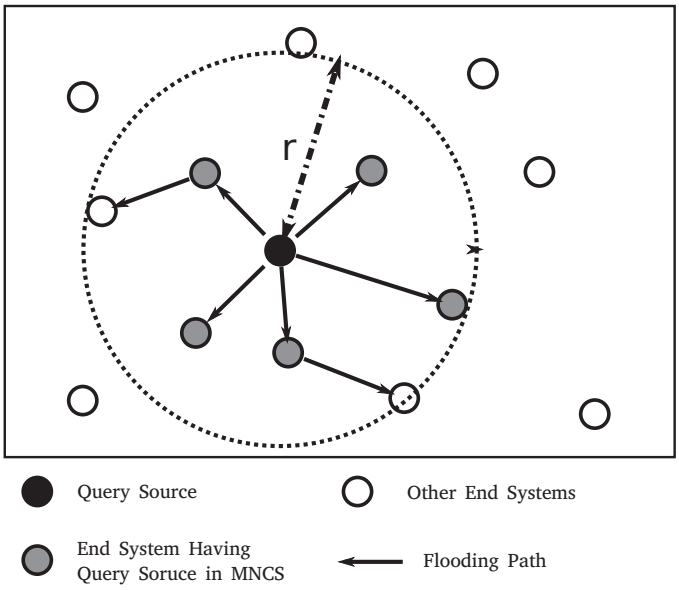

Fig. 8. Example of Spatially Bound Query Flooding on Reverse Greedy Routing Path

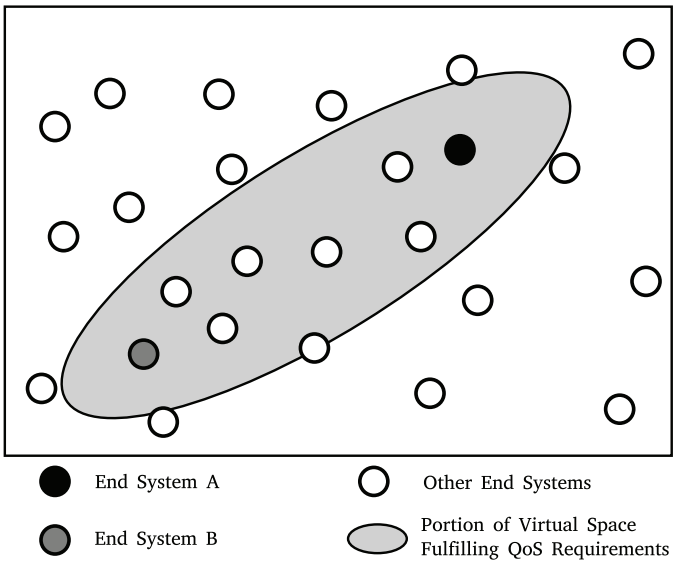

Fig. 9. Portion of Virtual Space Covered by Q_{RTT}

In the next Section, we will broaden this idea into flooding different shapes within the virtual space to provide a service, which is otherwise not easy to obtain: finding the QoS paths in overlay networks.

C. Solving Network Optimization Problems (QoS)

Providing quality of service (QoS) guarantees between two overlay nodes A and B in an overlay network means that a path in the overlay network can be found on which a given set of parameters is satisfied. Frequently used QoS parameters are bandwidth, jitter, delay, etc. In the worst case, finding such paths in an overlay network is NP-complete, since every possible path between A and B must be examined. In this Section, we propose a solution that considerably reduces the number of overlay nodes examined when searching for the optimal QoS path.

To find a path fulfilling QoS requirements Q_r between two overlay nodes A and B with positions $\mathcal{C}_A, \mathcal{C}_B$, we have to examine almost every possible path between A and B. It is relatively easy to find a path that fulfills one of the QoS requirements, e.g. RTT or number of hops. However, it is rather unlikely for that path to fulfill also all other QoS requirements, e.g. bandwidth or jitter. Thus, in our search, we may have to examine all paths to ensure that we find the one that best fulfills all required QoS parameters. In other words, to find a path that fulfills Q_r, we may have to flood the entire overlay network.

Since we have an embedding of all overlay nodes into a virtual space, we can reduce the extent of flooding. To achieve this, we only consider the portion of the virtual space that fulfills the requirement concerning the maximum RTT (Q_{RTT}). In Figure 9, we depict such a portion in a two-dimensional space. As we can see, as long as the required RTT is larger than the distance between A and B in the virtual space, the portion of the two-dimensional virtual space fulfilling Q_{RTT} is an ellipse. Should the required RTT be smaller than the distance between the two overlay nodes, the search can be terminated immediately, as it is impossible to find such a solution. In higher dimensions, this portion would be an ellipsoid.

To find a path that fulfills all given QoS requirements Q_r, among which there must be a condition Q_{RTT} limiting RTT, we propose using the same NNCS overlay network as described in Section III. We propose to flood a spatially bound section of the overlay network. Only the forward paths are flooded, i.e. overlay nodes do not need to compute the reverse paths, and, hence, they do not need to gather the NNCS sets of their direct neighbors. We initialize the flooding with request messages for finding paths at overlay node A. Each flood message M_f has the following fields:

- Q_r Original QoS request.
  This field is used to identify the request to which M_f belongs. The request must state an upper limit for the RTT of the path (Q_{RTT}).
- Q'_r Current QoS request.
  This field contains the current values of the QoS parameters after the characteristics of the path so far have been removed. For example, at each "hop" in the overlay network, the RTT of the preceding "hop" is deducted from the RTT counter, resulting in the current Q'_{RTT}. The same applies for all other additive parameters of Q'_r such as jitter.
- P Recorded path.
  This field is a list of all overlay node addresses, which the message passed along its path. This information is used to reconstruct successful paths.

In the beginning, an initial flood message M_f is sent out, where P contains only the address of the initiating overlay node A and Q'_r is equal to Q_r. Overlay node A then calculates the distances for each of its neighbors $\{N_1 \dots N_c\}$. To each neighbor N_i, for which the sum of the distances $d(A, N_i) + d(N_i, B)$ is lower than Q_{RTT}, a copy of M_f^i is sent. Message M_f^i contains Q'_r, which was adjusted with the characteristics

of the logical link between A and N_i, e.g. RTT, jitter, etc.

Before M_f^i is sent the availability of non-additive QoS parameters such as minimum bandwidth is also checked. If all parameters are fulfilled, e.g. the bandwidth between A and N_i is larger or equal to the requested bandwidth, M_f^i is sent to N_i. This procedure is repeated at each hop. All messages that reach overlay node B contain paths that fulfill the QoS requirements defined in Q_r. Then, either overlay node B chooses one of the paths and sends reservation messages backwards through the path, or every path that fulfills Q_r is sent back to overlay node A, so that A can decide which path to reserve.

V. Evaluation

To evaluate our approach, we implemented the NNCS protocol using an event based network simulator [11]. Our network model delayed the delivery of messages by half the value of RTTs contained by the provided RTT matrix. We used RTT data sets as our RTT matrix. One data set was provided by the Planet-Lab "all sites ping" experiment [19] and contained 217 overlay nodes. The other data set (KING) was obtained by measuring RTT distance between 462 overlay nodes using the King method [4]. The overlay network was built using the Fisheye overlay network [7]. To determine the positions of the overlay nodes we used NetICE9 [8]. On top of NetICE9, we built an NNCS overlay network.

All our results present the relative path stretch between two overlay nodes, which is obtained by dividing the actual RTT on the path through the overlay network by the RTT along the optimal path in the underlying network. The relative path stretch can never be smaller than 1.0. A value of 2.0, for example, means that the path through the overlay network was twice as long as the optimal path.

We compared the performance of our approach (NNCS) to the performance of Pastry [15] in terms of the relative path stretch. We related the path streches found when using greedy unicast routing in the NNCS overlay network with the path stretches achieved by Pastry unicast routing. Our goal was to determine if the NNCS overlay protocol finds better paths than another well-known and established topology-aware overlay protocol. Another goal was to deliver estimates on the possible errors for the approaches described in Sections IV-B and IV-C.

Both Pastry and NNCS use random numbers for different aspects of the protocol. To reduce the influence of randomness on the results, we performed our simulations on the same data using different initial seeds. We performed every simulation with 20 different seeds, which were used to initialize pseudo random generators in the simulations. We collected all results and present them as one cumulative distribution function (CDF) curve in Figures 10 and 11.

Since the number of dimensions used influences both, the precision of the RTT embedding performed by NetICE9 and the average number of neighbors in NNCS, we simulated NNCS with a varying number of dimensions. As stated in [2], [9], the optimal number of dimensions for embedding distances obtained from the Internet is in the range $5-7$. For this reason, we varied the number of dimensions used in our simulation from 3 to 6.

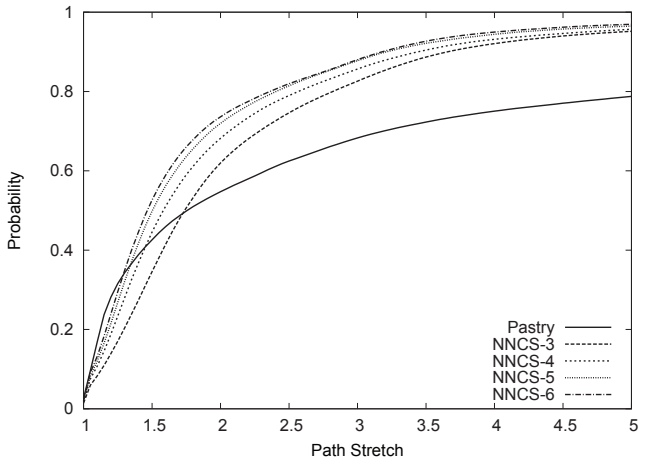

Fig. 10. CDF of Relative Path Stretch in Simulations Performed Using Planet Lab Data

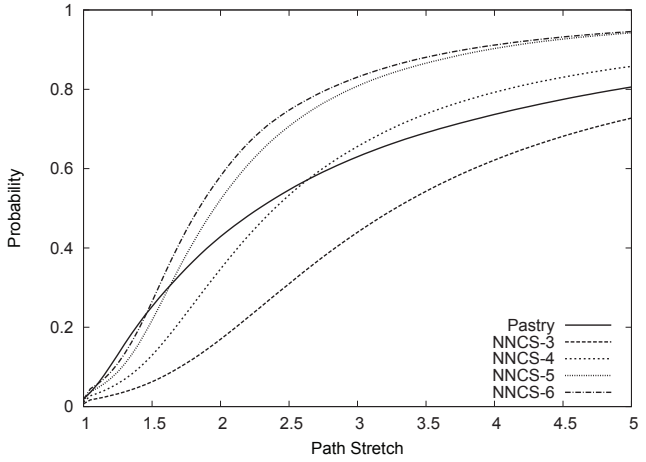

Fig. 11. CDF of Relative Path Stretch in Simulations Performed Using Data Obtained With King Method

Figure 10 shows that NNCS outperforms Pastry in the Planet-Lab data set already with $k = 3$ dimensions. With an increasing number of dimensions, the performance of NNCS improves further. The largest performance increase is achieved using 5 dimensions. With k larger than 5, there is no significant performance gain.

As Figure 11 shows, the situation with the King data set is similar. Here, however, Pastry performs better than NNCS in a 3-dimensional virtual space. With an increasing number of dimensions, the performance of NNCS improves. Similar as with the Planet-Lab data, the performance of NNCS in $k = 5$ and $k = 6$ dimensions does not significantly differ. With $k = 5$ dimensions, the maximum dimensionality of the provided data is almost achieved. Thus, every increase in the number of dimensions does not further improve the embedding.

VI. Conclusions

In this paper, we have introduced a novel method for building an overlay network with overlay nodes that are embedded into a virtual metric space: the nearest neighbors convex set (NNCS) overlay network protocol. The main advantage of

NNCS is that greedy routing is always successful. Based on NNCS, we have described a trivial unicast (greedy routing) and a multicast (flooding on reverse unicast greedy routing paths) protocol. We also described how the multicast protocol can be spatially bounded to achieve limited flooding that can be used for localized searches in the overlay network, e.g., for service discovery. We also showed how NNCS can be used to reduce overhead when searching for paths that fulfill given QoS requirements. In our evaluation we showed that paths taken through an NNCS overlay network are more efficient than paths taken in Pastry.

REFERENCES

[1] P. Bose and P. Morin. Online routing in triangulations. In *ISAAC '99: Proceedings of the 10th International Symposium on Algorithms and Computation*, pages 113–122, London, UK, 1999. Springer-Verlag.

[2] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: a decentralized network coordinate system. In *SIGCOMM '04*, pages 15–26, New York, NY, USA, 2004. ACM Press.

[3] S. R. et al. A scalable content-addressable network. In *Proceedings of ACM SIGCOMM*, 2001.

[4] K. P. Gummadi, S. Saroiu, and S. D. Gribble. King: estimating latency between arbitrary internet end hosts. *SIGCOMM Comput. Commun. Rev.*, 32(3):11–11, 2002.

[5] P. Kirk. Gnutella - a protocol for a revolution url: http://rfc-gnutella.sourceforge.net/, 2003.

[6] H. Lim, J. C. Hou, and C.-H. Choi. Constructing internet coordinate system based on delay measurement. In *Internet Measurement Confgerence 03*, October 2003.

[7] D. Milic and T. Braun. Fisheye: Topology aware choice of peers for overlay networks. In *Local Computer Networks, 2009. LCN 2009. IEEE 34th Conference on*, pages 467 –474, oct. 2009.

[8] D. Milic and and T. Braun. Netice9: A stable landmark-less network positioning system. In *Local Computer Networks (LCN), 2010 IEEE 35th Conference on*, pages 96 –103, oct. 2010.

[9] T. S. E. Ng and H. Zhang. Predicting internet network distance with coordiantes-based approaches. In *IEEE Infocom02*, New York / USA, June 23-27 2002.

[10] T. S. E. Ng and H. Zhang. A network positioning system for the internet. In *USENIX 2004*, pages 141–154, Boston MA, USA, June 2004.

[11] OMNET++, avail. online:http://www.omnetpp.org, 2007.

[12] G. Pei, M. Gerla, and T.-W. Chen. Fisheye state routing in mobile ad hoc networks. In *Proceedings of ICDCS Workshop on Wireless Networks and Mobile Computing, April 2000, Taipei, Taiwan*, pages D71–D78. ICDCS, April 2000.

[13] C. Plaxton, R. Rajaraman, and A. W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *Proceedings of the ACM SPAA*, pages 311–320, Newport, Rhode Island, June 1997.

[14] S. Ratnasamy, M. H, R. Karp, and S. Shenker. Topologically-aware overlay construction and server selection. In *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, 2002.

[15] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for largescale peer-to-peer systems. IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Nov. 2001.

[16] Y. Shavitt and T. Tankel. Big-bang simulation for embedding network distances in euclidean space. *IEEE/ACM Trans. Netw.*, 12(6):993–1006, 2004.

[17] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Trans. Netw.*, 11(1):17–32, 2003.

[18] L. Tang and M. Crovella. Virtual landmarks for the internet. In *Internet Measurement Confgerence 03*, October 2003.

[19] C. Yoshikawa. Planetlab all-sites-pings experiment, 2006.