HyCube: A distributed hash table based on a hierarchical hypercube geometry

Artur Olszak

Institute of Computer Science, Warsaw University of Technology
A.Olszak@ii.pw.edu.pl

Abstract. This paper presents a distributed hash table based on a hierarchical hypercube geometry. An approach employing a novel variable metric adopting the Steinhaus transform is presented. The use of this metric and the hierarchical hypercube geometry allow to reach very good performance of the network and a very high level of resilience to node failures. The architecture of the distributed hash table, routing, and search algorithms are presented. The results of the simulations have been included in the paper and compared with existing solutions.

Key words: distributed hash table, Steinhaus transform, hierarchical hypercube, routing

1 Introduction

Recently, we observe an increase in interest in peer-to-peer networks, in particular routing algorithms. Most of the networks currently developed are based on a distributed hash table algorithm (DHT). DHT systems store key-value pairs in a distributed manner, the set of keys is distributed among the nodes participating in the DHT. The nodes in DHTs usually share storage or computation resources with other nodes. Each node has its unique identifier and the resources (values) are located by searching for a node responsible for the resource key. Usually this is the node (or nodes) with the *id* closest to the key. Every node stores a set of references to other nodes - a routing table. The routing table is built in a way that allows routing messages, decreasing the distance left to the destination node in each routing step (distance between identifiers, according to the chosen metric). DHT systems can be divided into groups according to their geometry (connections graph) which defines the overlay network. The overlay network implies the structure of the routing tables. The most important DHT geometries are ring (e.g. *Chord*[1]), XOR metric (e.g. *Kademlia*[2]), tree (e.g. *Pastry*[3]), hypercube (e.g. *CAN*[4]), butterfly (e.g. *Viceroy*[5]). The geometry influences mostly the route lengths, but it also affects the level of resilience to node failures.

In [6], the authors analyze the impact of the routing geometry on static resilience[1] and on the average route length and present the results of the sim-

[1] The ability of the network to route messages between nodes in the presence of node failures without the aid of recovery mechanisms

ulations. Authors claim that flexibility in the next hop selection (which can be measured by the number of nodes in the routing table to which a message can be routed) has a considerable impact on static resilience. Different geometries imply different levels of flexibility in the next hop selection and the simulations prove that it has a great impact on the static resilience.

In some DHT systems, nodes maintain sets of so called sequential neighbors. Sequential neighbors are the preceding node and the following node (existing nodes with identifiers closest to the node's identifier that appear directly before or after it according to a certain sequence). Nodes form a logical ring using connections with predecessors and successors. The leaf set in *Pastry* and the predecessor and the successor in *Chord* are examples of sequential neighbors. Messages can always be routed to these nodes, decreasing the distance left to the destination - depending on the direction in which the destination is located, either the predecessor or the successor is chosen. In order to increase the level of resilience to node failures, in some DHT networks, nodes maintain sets of several preceding and several following nodes so that packets can be routed to the next hop, even in the presence of failures of many nodes. The results of the experiments presented in [6] show that sequential neighbors greatly improve static resilience but the average route length may significantly increase in the presence of many node failures. For this reason, nodes usually store some constant number of sequential neighbors to provide tolerance to node failures, whereas efficient routing is provided by routing tables specific for particular DHT algorithms.

This paper presents the architecture, routing geometry, routing and search algorithms of *HyCube* - a DHT peer-to-peer network based on a hierarchical hypercube geometry. Routing is based on a variable metric adopting the Steinhaus transform which defines distances between nodes. Using such a metric results in a very high degree of flexibility in the next hop selection. In comparison to schemes using sequential neighbors, this approach allows to achieve a very high level of static resilience and a shorter average path length in the presence of node failures.

The rest of this paper is organized as follows. Section 2 presents the design of *HyCube*. Section 3 describes the the routing algorithm. Neighbor selection algorithms adopted by *HyCube* are presented in Section 4. Section 5 describes the node lookup procedure. The k nearest neighbors search procedure is discussed in Section 6. The experimental results are presented in Section 7. Section 8 concludes.

2 HyCube

The routing geometry of *HyCube* is a combination of a tree geometry and a hypercube geometry. It is based on *Plaxton mesh*[7] but nodes are also logically located on a d-dimensional torus. Nodes in the network are distributed in vertices of a hierarchical hypercube. Figure 1 presents the structure of the network for 3 dimensions and 2 hierarchy levels.

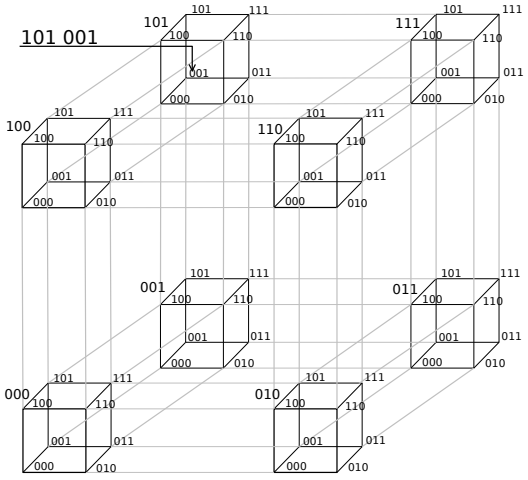

Fig. 1. A hierarchical hypercube (3 dimensions and 2 levels of hierarchy)

A hierarchical hypercube is a hypercube whose vertices are also hypercubes. Vertices of the hypercubes at the lowest level are positions which may be occupied by nodes. The position of a node in a hierarchical hypercube is determined by its identifier. The identifier of a node is a string of d-bit groups determining the positions of the node in hypercubes at particular levels (starting with the highest level). The position in a hypercube is a string of bits corresponding to the location of the node in the hypercube in particular dimensions. The length of the identifier equals $d \cdot l$, where d is the number of dimensions and l is the number of levels. The structure of a hierarchical hypercube and a tree are isomorphic. However, looking at it as on a hierarchical hypercube gives an idea of the spatial arrangement of the nodes in the network. The numbers formed from bits corresponding to particular dimensions relate to the coordinates of the node in these dimensions in the system of coordinates with the center in point 0. Thus, considering the highest level hypercube as a segment of R^d space, the distance between nodes can be defined by any metric in R^d space. The geometry of *HyCube* has one more property - the set of positions (coordinates) in each dimension is treated as on a ring. That means that after the point $2^l - 1$, point 0 is located. The network should therefore not be treated as a d-dimensional space limited in each dimension by 0 and $2^l - 1$ but as a d-dimensional torus with the perimeter equal to 2^l in each dimension. This modification will be important in determining distances between nodes - in every dimension the distance is determined like on a ring - the shorter of the distances in either direction is chosen. In order to obtain a short average path length and a reasonable size of the routing tables, in *HyCube*, the number of dimensions is 4 and the number of levels is 32 (to obtain a 128-bit address space).

3 Routing

In this section, the routing procedure of *HyCube* is described. Section 3.1 describes routing tables that nodes maintain to support the routing procedure, Section 3.2 presents the routing algorithm. The routing metric used is discussed in Section 3.3.

3.1 Routing tables

Primary routing table. The primary routing table has the same structure as in *Plaxton mesh*. It has l levels (the number of hierarchy levels). At each level there are 2^d slots (d - the number of dimensions). In the routing table of a node X in the slot j at level i ($i \geq 0$), a reference is stored to a node that is in the same hypercube at level $i+1$ and in the hypercube corresponding to the number j at level i (lower level). At each level $i > 0$, there is a slot corresponding to the hypercube in which the node X exists. This slot is empty as the routing table contains a whole level corresponding to it.

Secondary routing table. The secondary routing table of a node X contains nodes from adjacent hypercubes to the hypercube of node X in each dimension, in both directions, at each level. An adjacent hypercube is one whose coordinate in the particular dimension is greater or smaller by 1 than the coordinate of the hypercube of X at the particular level (taking into consideration passing coordinate 0 - like on a ring). The secondary routing table does not contain nodes in slots at the highest level as hypercubes corresponding to them are included in the primary routing table. Also, one of the sibling hypercubes on each level in each dimension is covered by a primary routing table slot. The slots of the primary as well as the secondary routing table also don't contain the nodes that are covered by a slot at a lower level of the secondary routing table. These ensure that the primary and the secondary routing table slot scopes do not overlap.

The secondary routing table gives a higher level of flexibility in the next hop selection. If the distance between nodes is defined by a metric in R^n space, it is very likely that the secondary routing table contains nodes that are closer to any arbitrarily chosen node.

Neighborhood set (closest neighbors set). Beside two routing tables described above, nodes maintain sets of closest to them (according to the chosen metric) nodes existing in the network. These sets allow to route messages (decreasing the distance left), even if there are no appropriate nodes in both routing tables. Such sets greatly increase the probability of delivering a message in the presence of node failures. In *HyCube*, the size of this set is 16.

The neighborhood set should provide a possibility to route packets regardless of the direction in which the destination node is located. Therefore, it is important that nodes in neighborhood sets be evenly distributed in respect of

directions. There might be a scenario where some nodes would have more neighbors in one direction and no neighbors in other directions. In such a case, the nodes would not be able to route packets in all directions (using neighborhood sets). The problem becomes more significant in the presence of node failures.

Ensuring even distribution of nodes in respect of directions may cause some more distant nodes to be included in neighborhood sets and pass over some closer nodes. Thus, both, proximity and even distribution, should be considered.

There are a lot of definitions of the even distribution of nodes, as well as a lot of possible algorithms of improving the eveness of distribution. These include projecting the coordinates of nodes on a sphere and maximizing the average or minimal distance between all the pairs. Such an approach may be improved by considering the square roots of the distances, which would eliminate the pairs of nodes close to each other with priority.

The technique of ensuring even distribution of nodes adopted by *HyCube* splits the space into parts (orthants of the system of coordinates with the center at the address of the node whose neighborhood set is considered) and attempts to ensure that in each orthant, the number of nodes is the same. Nodes are chosen to the particular orthants by proximity. Such a solution is very simple, efficient and does not require much computational overhead. The simulations proved that the efficiency of this technique is comparable with the more complex solutions and is significantly less resource-demanding.

3.2 Routing algorithm

In each routing step, it is first checked if there is a reference to the destination node in the neighborhood set. If this is the case, the message is sent directly to the node. Otherwise, the primary and the secondary routing tables are searched for the node closest to the destination.

First, the slots from the primary routing table are selected that correspond to nodes that share at least one group of d bits longer prefix of id with the destination node id than with the current node id. In a hierarchical hypercube, these slots correspond to hypercubes in which the destination node is located, at lower levels than the lowest level hypercube containing both, the current and the destination node. From these nodes, the node sharing the longest prefix with the destination node is chosen and the message is routed to this node.

If there are no such nodes, the slots corresponding to nodes sharing the same prefix length with the destination node are checked and nodes closer to the destination node (according to the chosen metric) than the current node are selected.

If no node sharing longer prefix with the destination node than with the current node was found in the previous steps, the slots from the secondary routing table are selected that correspond to sibling hypercubes in the direction in which the destination node is located, in each dimension, at levels $\lfloor \log_2 d_{dim} \rfloor$ and $\lceil \log_2 d_{dim} \rceil$, where d_{dim} is the distance from the current node to the destination node in the dimension dim (the distance between the coordinates on the ring). Only those nodes that share at least one d-bit group longer prefix of

id with the destination node or share the same number of d-bit groups but are closer to the destination than the current node (according to the chosen metric) should be considered.

If still no nodes are found, all nodes from both routing tables and the neighborhood set that share at least one d-bit group longer prefix with the destination node or share the same number of d-bit groups but are closer to the destination than the current node (according to the chosen metric) are considered.

From the set of the nodes selected in the steps above, the node sharing the longest prefix of *id* with the destination (number of d-bit groups) is chosen. If there is more than one such node, the node closest to the destination (according to the routing metric in use) is chosen.

In the final part of a route, when a packet is relatively close to the destination node, the routing algorithm may omit some nodes that are close to the destination, but do not share the same long or longer prefix of *id* with the destination node than with the current node. The problem becomes more significant if a multidimensional metric is used. In *HyCube*, before choosing the next hop, each node checks if the distance to the destination is shorter than the average distance to the nodes in the neighborhood set multiplied by the factor λ: $d_{dest} < avg(d_{neigh}) \cdot \lambda$. If this condition is satisfied, all further nodes on the route are chosen based only on their distance to the destination node. They might not share the same long or longer prefix of the identifier. At first, nodes from the secondary routing table at levels $\lfloor \log_2 d_{dim} \rfloor$ and $\lceil \log_2 d_{dim} \rceil$ are checked. If no nodes are found, all nodes from both routing tables and the neighborhood set are checked. The greater the value of λ, the longer parts of routes will be determined based only on the distance left. This value should be large enough to ensure a high probability of packet delivery. However, too large values of λ would cause an increase in path lengths. The value $1,35$ was determined experimentally and is a good compromise between the path length and the probability of packet delivery.

The expected route length is $\log_{2^d} N$ hops and, on average, $\log_{2^d} N \cdot (2^d - 1)$ slots are populated in the primary routing table and $(\log_{2^d} N - 1) \cdot d$ in the secondary routing table, where N is the number of nodes in the network[2].

3.3 Metric

The choice of the metric, i.e. how distances between nodes are determined, has a great impact on the average route length and the probability of packet delivery. Choosing a one-dimensional metric allows the use of sequential neighbors, which greatly improves the static resilience. If the number of sequential neighbors is s, half are predecessors and half are successors of the node, the packet would be dropped only if all $s/2$ nodes in the appropriate direction failed. However, the use of sequential neighbors may cause a significant increase in path lengths in the case of node failures, when many routing table slots are empty. In *HyCube*, a multidimensional metric is used (a metric in a multidimensional space), which

[2] Based on the assumption that nodes are distributed evenly in the space

significantly decreases the expected path length when routing using only neighborhood sets. The expected path length equals $\sqrt[d]{N} \cdot \frac{\sqrt{d}}{2}$, while for sequential neighbors it is proportional to N. This fact becomes very important when considering network maintenance algorithms. By using a multidimensional metric, the network loses properties connected with existence of sequential neighbors. However, the use of the variable metric adopting the Steinhaus transform yields better static resilience than with the use of sequential neighbors.

Let us consider routing using only neighborhood sets and assume that in each step, the next hops are chosen only by the distance left to the destination, without ensuring the prefix condition. Only some part of the neighborhood set nodes is closer to the destination node than the current node. It is crucial that the number of such nodes be as large as possible (so even in the case of many node failures the packet will not be dropped). We may estimate the expected ratio of the number of nodes to which the packet may be routed to the number of all nodes in the neighborhood set as the probability that the packet may be routed to a single node in the neighborhood set.

The most common metrics in R^n space are Minkowski distances:

$$L_m(x,y) = \left(\sum_{i=0}^{d-1} |x_i - y_i|^m \right)^{\frac{1}{m}}, m \geq 1 \tag{1}$$

In particular, L_1 is the Manhattan (taxicab) distance, L_2 is the Euclidean distance and L_∞ is the Chebyshev distance. L_2 is the only metric from the Minkowski distances that preserves distances between nodes after space rotation. This causes some undesirable properties of metrics L_1 and L_3 to L_∞. When these metrics are used, if the destination node is located in different directions, the expected numbers of nodes in the neighborhood set to which the packet may be routed are different. For this reason, only L_2 was considered.

If the Euclidean metric is used, the probability that a packet may be routed to a node in the neighborhood set may be calculated as a function of $k = \frac{d}{r}$, where d is the distance left to the destination node and r is the distance from the neighborhood set node to the current node. It is the ratio of the number of points that are in the distance r from the current node and are closer to the destination than d, to the number of all points being at the distance r from the current node. To simplify, the numbers of points were determined as lengths of the curves, areas of the surfaces and their equivalents for spaces with more dimensions. The calculated values of the probability that a node in the neighborhood set is closer to the destination (for varying numbers of dimensions) are presented in Fig. 2. The figure shows that the number of nodes in the neighborhood set, to which a packet may be routed, strongly depends on k. The closer the packet is to the destination node, the fewer appropriate nodes. The situation gets worse as the number of dimensions increases. This fact has a great impact on static resilience - fewer node failures may cause the packet to be dropped.

In [8], the Steinhaus transform was described. The theorem presented says that if X is a set and D is a metric in this set, D' is also a metric in X for any

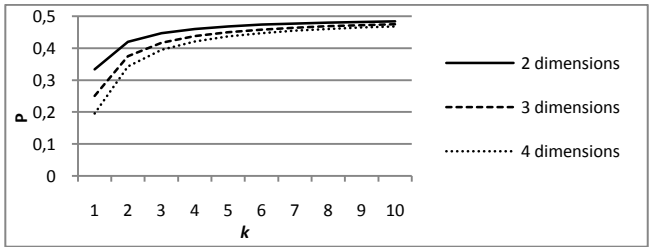

Fig. 2. Probability that a node in the neighborhood set is closer to the destination node than the current node

$a \in X$, where:

$$D'(x,y) = \frac{2D(x,y)}{D(x,a) + D(y,a) + D(x,y)} \tag{2}$$

Applying a metric with the Steinhaus transform for every route, setting a to the *id* of the source node, causes the next hops to be chosen in such a way that they are closer to the destination node and more distant from the source node. Such a solution increases the expected number of neighborhood set nodes to which packets can be routed in each step - it allows sending packets using more roundabout routes, but still being convergent to the destination node.

The use of the Steinhaus transform yields very good routing parameters and very high static resilience for networks containing relatively few nodes. For networks containing much more nodes, the addend $D(x,a)$ in the denominator, where x is the current node, has less influence on the value of the distance as its changes in particular steps are very minor compared with the value of the entire denominator. Thus, the more nodes in the network, the lesser the influence of the Steinhaus transform on the static resilience. However, some modification can be introduced - a variable metric adopting the Steinhaus transform, where point a is changed by nodes on routes. Point a is initially set to the source node *id*. The following nodes, before choosing the next hops, check whether they are closer (in terms of the Euclidean metric) to the destination than the current point a. In such a case, point a is changed and gets the value of the current node. Such a way of changing point a ensures that the routing is convergent to the destination (there will be no cycles on routes) and gives a high level of flexibility in the next hop selection along the whole route, regardless of the network size. The expected route length is still proportional to $\sqrt[d]{N}$ and owing to the increase in the flexibility in the next hop selection, static resilience is even better than in networks using sequential neighbors, which can be seen in the simulation results presented in the remainder of the paper. Figure 3 presents a comparison of simulation results for different metrics for a 4-dimensional network containing 1000 nodes. The routing algorithm simulated did enforce the common *id* prefix length condition. For comparison, if packets were routed using sequential neighbors, the curve would keep at about 0.5, regardless of the distance left.

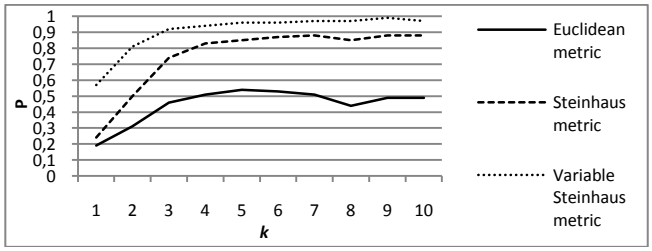

Fig. 3. Probability that a packet may be routed to a node in the neighborhood set

The final steps of routing with the use of a metric with the Steinhaus transform may cause a packet to be sent to a node that is more distant from the destination than it would be if the Steinhaus transform was not applied. When a node on a route cannot find the next hop in its routing tables and neighborhood set, it is possible that the route is broken in a point that is not the closest one to the destination in terms of the Euclidean metric. In some cases, it is crucial to reach the closest possible node if the destination node is not reached. Thus, one more modification was introduced to *HyCube* - when a packet cannot be routed by a node, the node tries to route it based only on the Euclidean distance left to the destination. All next hops after that should be chosen in the same way. Such an approach will cause that in the case the packet is dropped, a relatively close node to the destination is reached. From the experiments (for a network containing 1000 nodes, with 50% failed nodes, routing using only neighborhood sets), it appears that applying this phase in routing allows packets to be sent to a closer node in 79% cases and also increases the static resilience of the network.

4 Neighbor selection

As it was described in section 3.1, the way of constructing the neighborhood set is deterministic. However, there is a high level of flexibility when choosing the nodes for the routing tables. For each routing table slot, there might be many possible nodes whose *id*s satisfy the conditions for the slot. There are a lot of diffrent approaches for neighbor selection. The most widely used techniques and their implications are presented in [9] and [10]. *HyCube* adopts several approaches and is fully configurable regarding the neighbor selection techniques. As stated in [9] and [10], each technique has its advantages and drawbacks. Thus, depending of the application and the environment, one or another might be more suitable.

4.1 LNS

One of the techniques adopted by *HyCube* is the LNS, which bases the neighbor choice on node liveness information. The technique tends to populate the routing tables with the nodes that are likely to stay alive for a long time, which

would reduce the bandwidth and processing power consumption for the routing table recovery process as well as improve the lookup performance under churn. The *HyCube* LNS implementation is based on the assumption that nodes that were alive for a long time in the past will remain alive for a long time with a high probability. However, this implementation does not calculate the liveness information based on the node's join and leave times. Every node perioducally sends keepalive messages to all the nodes in its routing tables and updates the node's liveness value:

$$L_n = L_{n-1} \cdot p + (1 - p) \cdot L_{max} \tag{3}$$

if the keepalive confirmation was received, or:

$$L_n = L_{n-1} \cdot p \tag{4}$$

when the keepalive fails. When L falls below the threshold value L_T, the node is replaced by a new node and its value L is set to the initial value L_{init}. The value $0 < p < 1$ determines the sensitivity of the algorithm. L_{max} defines the maximum value of L

4.2 PNS

Another neighbor selection technique used by *HyCube* is PNS. The approach chooses the nodes for routing tables based on their physical proximity (usually the ping round trip time or the network bandwidth between two nodes). A node in a routing table slot is replaced if another node appropriate for the routing table slot is found that is physically closer than the current node. PNS improves the routing and lookup performance by reducing the latencies between particular pairs of nodes (and thus reducing the overall latency on the route).

Usually, it is not always possible to have the physical proximity determined between all pairs of node. Determining the proximity at the time the node is notified about a new routing table slot candidate is also not a good solution as the proximity value would be based on just one check and would be vulnerable to the current undelying network conditions.

In [11] and [12] two approaches for efficient network distance prediction were presented - $IDMaps$ and $GlobalNetworkPositioning$ - GNP. $IDMaps$ is a system in which special servers are deployed (trackers) that hold a topological map of the network as well as inter-host measurments and may be querried for a prediction of distance between two nodes. The $IDMaps$ major drawback is that it relies on Tracers to be close enough to a host so that a reasonably accurate distance prediction is calculated. This problem was ultimately improved with $GlobalNetworkPositioning$. GNP approach models the Internet as a geometric space and computes geometric coordinates to characterize the physical position of hosts. GNP utilizes a set of reference points called Landmarks. The ordinary hosts entering the network use first the Landmark servers as references to calculate their coordinates. The authors prove that GNP can accurately and

efficiently predict network distances. However, there is also a deployment cost - it requires the fixed infrastructure of the Landmark nodes.

Vivaldi[13] is a simple, lightweight, fully distributed algorithm that accurately predicts the communication latencies between the hosts. It assigns synthetic coordinates to hosts and requires no fixed network infrastructure and no distinguished hosts. In *Vivaldi*, the hosts hold and constantly update their coordinates. The network latency to any node can be predicted based on the coordinates stored locally. The authors show that despite beeing fully decentralized, *Vivaldi* achieves competitive accuracy with that of *GNP*. As *Vivaldi* requires no infrastructure and is easy to deploy in existing applications, it has beed adopted by *HyCube* as a PNS algorithm.

4.3 Secure neighbor selection

DHT overlays provide efficient routing and node lookup, a high level of resilience and self-organization. However, they are vulnerable to a variety of attacks. Even a small fraction of malicious nodes may prevent correct message deliveries. In [14], the authors propose secure routing tables - additional constrained routing tables, which would be used only when routing using regular routing tables fails (built using for example PNS or other neighbor selection strategy). The constrained routing tables are built based on some strong constraint on the set of node *id*s that can fill each slot of the routing table. This constraint should be strong enough to make it difficult to be manipulated by attackers. The authors propose that each entry in the constrained routing table be the closest to the desired point p in the id space. As a modification of *Pastry* DHT, they define point p as follows: for a routing table of a node with identifier i, the slot at level l and domain d (the routing table slot may contain nodes with node *id*s that share the first l digits with i and have value d in the $l+1st$ digit), it shares the first l digits with i, it has the value d in the $l+1$ digit, and it has the same remaining digits as i.

The above strategy has been adopted by *HyCube*. However, the process of calculating the constraint has been modified. For each routing table slot (primary or seconadry routing table), the point p is defined as follows: it has the first l digits equal to the address of the hypercube corresponding to the routing table slot and remaining digits equal to those of i. However, before calculating the distance between the point p and the new routing table slot candidate, the candidate's *id* is xor'ed with the secret key, maintained by each node. The secret key is randomly re-generated after analyzing every N candidates. To avoid replacing the constrained routing table nodes every time the secret key is changed, every routing table slot contains the value of the distance calculated when the current node was added to it and the node is replaced only if the distance to the new node id (xor'ed with the new secret key) is less than the distance stored in the routing table slot. Such an approach makes it very difficult for any attacking node to be included in other node's routing tables.

4.4 *HyCube* mixed mode node selection

HyCube employs one more node selection approach - the mixed mode node selection, which is a combination of LNS, PNS and the Secure node selection. When considering a new candidate for a routing table slot, the algorithm calculates the weighted average of the three factors:

1. $Fact_L = \frac{L_{init} - L_{curr}}{L_{curr}}$
2. $Fact_P = \frac{P_{curr} - P_{new}}{P_{curr}}$
3. $Fact_S = \frac{S_{curr} - S_{new}}{S_{curr}}$

L_{init} is the initial value of L, L_{curr} is the value L for the current node in the routing table slot. P_{curr} and P_{new} are the proximities of the current and the new node calculated as described in the previous sections. S_{curr} and S_{new} are the distances to the current node and the new node calculated as described in the previous sections. The node is replaced if the value of *gain*:

$$gain = \alpha \cdot Fact_L + \beta \cdot Fact_P + \gamma \cdot Fact_S \tag{5}$$

is greater than δ. α, β, γ and δ are the algorithm parameters. The simulations show that the value of δ should be positive to prevent continuous replacement of nodes in the routing tables.

5 Node lookup

In the node lookup procedure, as opposed to routing, the next hop selection is made by the initiating node. The nodes receiving the lookup request don't route the message, they send back the initiating node a reference to a node (or nodes) that are the best candidates to route the message to. The decisions regarding the next node selection are made locally. To ensure the procedure is convergent to the lookup node, next hops must satisfy the same conditions as in the routing procedure, i.e. they must share a longer prefix with the destination node than with the previous node or share the same prefix length but be closer to the lookup node than the previous node. The metric used is the same as the one used in routing and the prefix mismatch heuristic also applies. When the lookup node is found, the message is sent directly from the sender to the recipient. Although node lookup requires more network trafic and the overall lookup latency is greater (the lookup time is doubled), the main advantage of such an approach is the possibility of returning and sending the lookup request to another node. This makes it possible to find the lookup node using more roundabout paths. The node lookup is initiated with three parameters:

- *id* - the id of the lookup node
- β - the maximum number of nodes returned by the intermediate nodes

– γ - the number of temporary nodes used during the lookup ($\gamma \geq \beta$)

During the node lookup, the initiating node holds the lowest distance reached so far to the lookup node (in terms of the Euclidean metric) and the closest node found. The node also maintains the set Γ containing γ closest (Euclidean) nodes found. The node initially fills this set with the γ closest nodes from its routing tables and the neighborhood set. The initial lookup request is sent to the closest node in the set Γ.

After receiving every response, the set Γ is updated to include γ closest node from the nodes currently present in Γ and the returned nodes. The next requests are sent to one of the nodes returned by the previous node - chosen like in the routing procedure - variable Steinhaus metric used, nodes sharing longer common prefix with the lookup node are considered closer, unless the prefix mismatch heuristic applies.

If the node to which the lookup request was sent does not return any new nodes, the next request is sent to the closest (Euclidean) node from Γ, which has not yet been requested.

For each virtual route (lookup request sent to node G from Γ), the initiating node holds the Steinhaus point. The Steinhaus point is updated the same way as in the routing algorithm, its initial value is the id of G and next hops are calculated as in the routing algorithm. When a node on a virtual route does not return any new nodes and another virtual route is created (new node G from Γ is selected), the Steinhaus point again is given the value of G.

The procedure continues until the lookup node is returned by an intermediate node or if there are no more nodes in Γ to which the request has not been yet sent. In that case, the result of the lookup is the closest node found during the lookup procedure.

When any intermediate node determines (based on its neighborhood set) that it is close enough to the lookup node, it may enable the prefix mismatch heuristic, which then remains enabled for this virtual route.

When the lookup node is not found (the lookup request was sent to all the nodes in Γ and no new nodes were returned), as in routing, the lookup procedure is repeated with the current Γ set, but looking for the node closest to the lookup address in terms of the Euclidean metric with the prefix mismatch heuristic enabled (like in the routing procedure).

When $\beta = 1$ and $\gamma = 1$, the node lookup procedure creates one virtual route that is exactly the one determined by the routing algorithm. By increasing β and γ, in case of failures, the algorithm is capable of returning and creating alternative routes and allows to sidestep the incostitent fragments of the connection graph.

6 *k* nearest nodes search

The purpose of the k nearest nodes search procedure is locating k nearest nodes to the given id in terms of the Euclidean metric. The nearest nodes search

procedure is used when a new node joins the network and may be also used for replicating the keys and key lookup in the distributed hash table. The nearest nodes search, like node lookup procedure, is performed locally by the initiating node, which sends requests to intermediate nodes and analyzes the responses. The decisions, to which nodes the requests are sent, are made locally by the initiating node.

The k-search procedure is initiated with six parameters:

- id - the id for which k nearest nodes should be found
- k - the number of nodes to be found
- α - the number of nodes to which a request is sent in each search step
- β - the maximum number of nodes returned by the intermediate nodes
- γ - the number of temporary nodes used during the search ($\gamma \geq k, \gamma \geq \beta, \gamma \geq \alpha$)
- ITN - gnore target node - determines whether the set of k nodes should contain the node with the id id or not

During the k-search procedure, the initiating node maintains a set Γ containing at most γ nodes, closest (in terms of the Euclidean metric) to id. For every node in Γ, current Steinhaus point is stored, as well as a flag indicating whether the prefix mismatch heuristic has been applied for the node. The k-search procedure consists of two phases. In the first phase, the set Γ is initially filled with maximum γ nodes from the routing tables and the neighborhood set, that are closest to id. The initial values of the Steinhaus points for the nodes in Γ are the node ids themselves.

In the consecutive steps, the initiating node sends the search request to at most α nodes from Γ (α nodes closest to id), to which the request has not yet been sent. After receiving the request, the nodes look for at most β nodes closest to id in their routing tables and neighborhood sets. The closest β nodes don't have to satisfy the longer prefix condition nor decreasing the distance left. The nodes should be the ones sharing with id the longest prefix and nodes sharing the same prefix length are chosen based on the distance to id. Every search request contains the current Steinhaus point for the node. The intermediate node, looking for the closest β nodes uses the Euclidean metric with the Steinhaus transform using the Steinhaus point sent in the request.

After receiving a reply from a intermediate node, the set Γ is updated so that it contains at most γ nodes closest to id (Euclidean). When a new node is added to Γ, the Steinhaus point for the node is given the value of either the current Steinhaus point for the node that sent the response or the returned node id (whichever is closer to id in terms of the Euclidean metric).

Every intermediate node determines if the prefix mismatch heuristic should be applied, based on its neighborood set. If the heuristic is applied, such an information is included in the reposnse and the PMH should then be used in all the requests sent to nodes returned by the current intermediate node. In other

words, all nodes returned by searching with the PMH enabled should also apply it. Additionally, if the prefix mismatch heuristic is enabled, it also forces the use of the Euclidean metric without the Steinhaus transform. The first phase is finished if none of α nodes to which a request was sent does not return any node closer to id than the most distant node in Γ.

The second search phase is a repetition of the first phase, however, requests are sent to all the nodes in Γ, the prefix mismatch heuristic is enabled and the Euclidean metric is used. The second phase is finished when none of the nodes in Γ returned a node closer to id than the most distant node in Γ. The result of the procedure is k closest nodes to id from the set Γ.

γ may be greater than k to allow more roundabout search paths. Especially, for small values of k, such roundabout paths are important so that the procedure finds also the nodes that are located in the space on the opposite side of the point id than the initiating node.

The simulation results show that the search procedure presented is able to find the k nearest nodes to the given id even in the presence of many node failures, even for $\gamma = k$ and $\alpha = 1$.

7 Experimental results

This section presents experimental results obtained in simulations of *HyCube* and *Pastry*. The properties evaluated were the static resilience and the average route length (based only on successful routes). For this purpose, a network generator and a simulator were implemented. The network generator generates a random network (*HyCube* or *Pastry*) containing the requested number of nodes - random node identifiers and random nodes in routing tables. Neighborhood sets are generated according to the criteria described in Section 2.2. The simulator reads the network generated by the network generator and simulates routing between random pairs of nodes in the presence of varying numbers of random node failures. A node failure means removing the node from the network and from the routing tables of all nodes maintaining a reference to it.

Figure 4 presents the results of the simulations of static resilience of *HyCube* containing 1000 and 10000 nodes with the use of different metrics. It can be seen that for the network containing 10000 nodes, the influence of the Steinhaus transform is very little. However, the variable Steinhaus metric gives a significant increase in the successful path rate, regardless of the network size.

Figure 5 presents a comparison of simulation results of *HyCube* and *Pastry* networks containing 10000 nodes. In the simulations, the secondary routing table was not used (because it does not exist in *Pastry*). The figure shows that *HyCube* is more resilient to node failures and the average route length is shorter than in *Pastry*. The results prove that despite the absence of sequential neighbors, a more robust architecture was achieved.

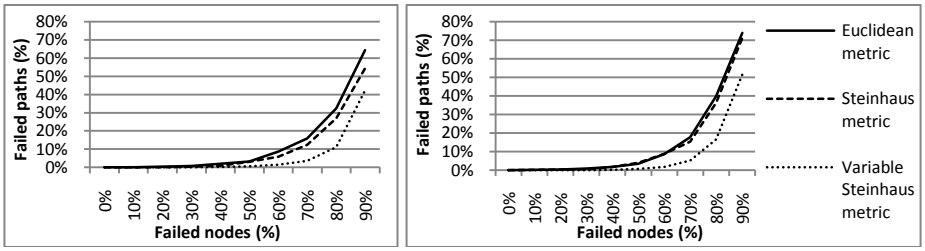

Fig. 4. Static resilience of *HyCube* - 1000 nodes (left) and 10000 nodes (right)

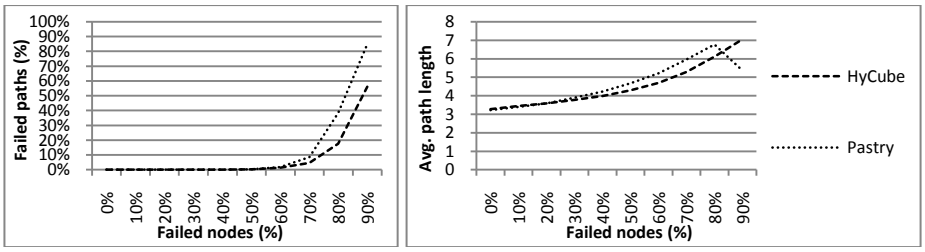

Fig. 5. Static resilience and path length increase in *HyCube* and *Pastry* (10000 nodes)

8 Conclusion

In this paper, the routing geometry and the routing algorithm of *HyCube* were presented - a DHT network based on a hierarchical hypercube geometry.

The experimental results indicate that the most crucial in terms of efficiency and static resilience was the choice of an appropriate metric and neighbor selection algorithms (the way the nodes are chosen to the routing tables). The simulations proved that the approach presented gives shorter average path lengths in the case of many node failures in comparison to solutions using sequential neighbors. The decrease in path lengths results from the use of a multidimensional metric. However, despite the absence of sequential neighbors, a very high level of static resilience was reached, which was achieved by the use of the variable metric adopting the Steinhaus transform.

In comparison with *Pastry*, *HyCube* gives better results both, in respect of the packet delivery probability and the path length increase in the presence of node failures. Moreover, *HyCube* has a very important advantage over *Pastry* - when routing using only neighborhood sets, the average path length is proportional to $\sqrt[d]{N}$, whereas when routing using only sequential neighbors, the expected path length is proportional to the number of nodes in the network.

To summarize, *HyCube* is an efficient and credible implementation of a distributed hash table. It is scalable and provides efficient routing of messages, even in the case of a large number of node failures.

References

1. Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., Balakrishnan, H.: Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In Proc. of the ACM SIGCOMM 2001 Technical Conference, 149-160 (2001)
2. Maymounkov, P., Mazières, D.: Kademlia: A Peer-to-peer Information System Based on the XOR Metric. In Proc. of the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02)
3. Rowstron, A. I., Druschel, P.: Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In Proc. of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)
4. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Schenker, S.: A Scalable Content-Addressable Network. In Proc. of the ACM SIGCOMM 2001 Technical Conference
5. Malkhi, D., Naor, M., Ratajczak, D.: Viceroy: A Scalable and Dynamic Emulation of the Butterfly. In Proc. of the 21st ACM Symposium on Principles of Distributed Computing (PODC '02)
6. Gummadi, K., Gummadi, R., Gribble, S., Ratnasamy, S., Shenker, S., Stoica, I.: The impact of DHT routing geometry on resilience and proximity. In Proc. of the 2003 Conference on Applications, Technologies, Architectures, and Protocols For Computer Communications (SIGCOMM '03)
7. Plaxton, C. G., Rajaraman, R., Richa, A. W.: Accessing Nearby Copies of Replicated Objects in a Distributed Environment. In Proc. of the 9th Annual ACM Symposium on Parallel Algorithms and Architectures, 311-320 (1997)
8. Clarkson K. L.: Nearest-Neighbor Searching and Metric Space Dimensions. Nearest-Neighbor Methods for Learning and Vision: Theory and Practice. MIT Press (2006)
9. Yingwu Zhu, Xiaoyu Yang: Implications of Neighbor Selection on DHT Overlays. In Proc. of the 14th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS '06)
10. Byung-Gon Chun, Ben Y. Zhao, John D. Kubiatowicz Impact of Neighbor Selection on Performance and Resilience of Structured P2P Networks. In Proc. of the 4th International Workshop on Peer-To-Peer Systems (IPTPS 2005), LNCS 3640, pp. 264-274, 2005.
11. P. Francis S. Jamin C. Jin Y. Jin D. Raz Y. Shavitt L. Zhang: IDMaps: A Global Internet Host Distance Estimation Service. In IEEE/ACM Transactions on Networking, Volume 9, Issue 5, pp. 525-540, Oct. 2001
12. T. S. Eugene Ng , Hui Zhang: Towards Global Network Positioning. In Proc. of the First ACM SIGCOMM Workshop on Internet Measurement, 2001
13. Frank Dabek , Russ Cox , Frans Kaashoek , Robert Morris: Vivaldi: A Decentralized Network Coordinate System. In Proc. of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM'04), 2004
14. Miguel Castro, Peter Druschel, Ayalvadi Ganesh, Antony Rowstron, Dan S. Wallach: Secure routing for structured peer-to-peer overlay networks. In Proc. of the 5th Usenix Symposium on Operating Systems Design and Implementation, Dec. 2002