A Fault Tolerant Distributed Location Service for Geographic Ad Hoc Routing

András Benczúr[*] Tamás Lukovszki[†]

Abstract

We introduce a new distributed location service, the *hypercubic location service* (HLS) for wireless mobile ad hoc networks. It provides information about geographic position of nodes, which is required by position based routing methods. Together with a position based routing, in particular with a greedy routing in appropriate spanner graphs, HLS contributes an efficient network layer of wireless ad hoc networks. Underlying HLS is an $O(\log n)$ degree graph which is based on a dynamic version of hypercubic graphs, especially of the De Bruijn graph. We introduce the notion of effective bits of node IDs and we define the hypercubic neighborhood relation accordingly. We show how dynamic operations, insertion and deletion of a node with a random ID can be performed in logarithmic time in such hypercubic graphs.

A key feature of HLS is its very strong fault tolerance as well as scalability to large changes in the number of nodes, allowing very flexible control over the network size. A large network can be gradually built from a few nodes and then again it can be reduced to a very small size with fast network topology updates. In addition the architecture is extremely fault tolerant. It resists the simultaneous failure of a constant fraction of the nodes, including even a systematic destruction within certain regions. The combination of scalability and fault tolerance results in a network which survives even a fast destruction.

1 Introduction

This work considers the problem of routing in large wireless mobile ad hoc mobile. Ad hoc networks require no fixed infrastructure. They are thus particularly suitable for example for rescue operations in regions affected by a natural disaster. In ad hoc networks the mobile hosts also perform routing tasks. The dynamics of such networks require routing strategies substantially different from the ones used in static communication networks, since storing and updating large routing tables at mobile hosts would congest the network with administration packages very fast.

Ad hoc routing received a lot of attention in the last years. Several routing algorithms have been developed: the so-called proactive protocols that continuously maintain route informations for all destinations like DSDV [18], and the reactive ones that construct the routes for the destinations as required like DSR [9], TORA [16], AODV [17]. Neither types of strategies are fully satisfying however: the proactive ones suffer from a congestion of administrative messages when devices begin to move relatively fast; reactive strategies in contrast have the drawback of long network search routes and thus high initiation costs.

[*]Computer and Automation Research Institute of the Hungarian Academy of Sciences, `benczur@sztaki.hu` and Department of Operations Research, Eötvös University, Budapest. Supported from grants OTKA T-30132 and T-29772; NWO-OTKA; AKP 104024

[†]Heinz Nixdorf Institute and Deptartment of Mathematics and Computer Science, Paderborn University `talu@uni-paderborn.de`. Supported by the DFG-SFB 376.

Li et al. [11] present a routing strategy that combines the advantages of *proactive* and *reactive* routing. They describe a method which consists of two sublayers. The first sublayer provides a position based method, called geographic forwarding. This mechanism can be used when the source node knows the geographic position of the destination. Position based routing methods utilize the geographic position of the destination node for choosing the next node in the routing path. (For a survey on position based routing see [15].) The other sublayer, called Geographic Location Service (GLS) provides a method to obtain this position information. In GLS, each mobile node selects a few other nodes as its location servers using the idea of consistent hashing combined with location informations. Assuming that the nodes can determine their current geographic position any time – e.g. by using a global positioning system (GPS) or some other method – they periodically resend position information to the location servers by a similar routing mechanism used for normal data transmission. In order to initiate communication, the source node needs the geographic position of the destination. It has to find a location server of the destination. A location server can be determined by visiting a well-defined sequence of nodes such that each acts as a location server of the next in the sequence.

We present a new location service that, by keeping most features of the GLS [11], also improves it in several senses.

- We improve the degree of the graph underlying the location service to $O(\log n)$, which is independent from the geographic distribution of the nodes.
- We provide very strong fault tolerance that also includes the self-scaling of the network, still we keep degrees at least as low as in the GLS.
- Unlike GLS, we need very little assumption about how the density varies across geographic subregions.
- HLS also survives a systematic destruction of devices, also a destruction of all devices within a region.
- Our location service forms a sublayer or the network layer, thus providing the flexibility of combining HLS with an arbitrary position based or other routing strategy.
- We keep the amount of administrative traffic close to that of [11] even in the fault free case, with a slight loss in the locality of communication that we can fix by a multi-layer solution.

In our Hypercubic Location Service (HLS) each node is identified by a unique random ID. Such a random ID distribution can be easily achieved by either hashing the node's IP or other physical address into a sufficiently large interval or hard-wiring a random sequence into the device. The nodes are organized in a dynamic and fault tolerant network based on hypercubic network [10] with respect to their random ID. In such networks neighbor IDs can be obtained by simple bit shift and bit flip operations of IDs. In order to manage the situation that most of possible random IDs are not present in the network, we introduce the notion of *effective bits* of the IDs and define the neighborhood relation in our hypercubic graphs according to such bits.

Each node in the network maintain location information of its hypercubic neighbors by periodically sending its own position to that neighbors. When the location of a node is known, a position based routing strategy can be used for sending messages to that node.

Routing via HLS: When a node u wants to send data to another node v, first u has to detect the location of node v. This happens as follows: The random ID of v can be obtained from the random ID of u by performing a certain sequence of elementary bit operations. This sequence defines a path in the hypercubic network from u to v. Node u sends a so-called location request for v on this path. The location request contains the ID of u and v and the location of u. Since the hypercubic neighbors are storing up-to-date location information from each other, the location request will be sent to the next node on the hypercube path using position based routing. When the location request reaches v, then v sends its own location to u by using position based routing. After u has received the position of v it sends the data to v using the position based routing, as well.

The random distribution of mobile node IDs ensures two key features of the HLS:

- We prove that the location work and the storage requirements are balanced across all the nodes.
- Since the IDs are independent of the geographic location or any other sensible network property, HLS resists faults of almost all nodes in a certain region or a systematical destruction of a constant fraction of nodes.

As mentioned above, in order to route the message between two consecutive nodes of the hypercube path, we can use any position based routing strategy. In particular, we can use appropriate sparse graphs called power spanners [13] that allow low power routing and/or a greedy position based routing with guaranteed delivery [2]. We suggest the use of the so-called Yao graph [22] as an underlying geometric sublayer, a graph with a number of desirable properties including:

- Each node has a constant number of outgoing edges.
- The graph can be computed and maintained efficiently in a distributed manner (see [21] for example), allowing fast recovery from faults.
- The graph allows a very simple position based routing strategy, called sector routing, which guaranties a short routing path.
- For each pair of nodes, the graph contains a path that very closely approximates the transmission energy of an energy optimal path[13, 20, 21].

1.1 Our Results

We present a routing and location service algorithm for wireless mobile ad hoc networks. This constitutes a network layer of the communication network without any fixed infrastructure. The algorithm is divided into two sublayers, one for the location service and one for position based routing between known locations. For the location service we describe a novel highly fault tolerant self-scaling architecture called Hypercubic Location Service (HLS). The architecture resembles of classical structures in parallel computing [10], bringing closer the ideas of parallel computing and ad hoc networking.

As underlying topology of the Hypercubic Location Service we relax the definition of hypercubic graphs by introducing the notion of effective bits of node IDs and we define neighbors of a node according to the effective bits. We show that dynamic operations, such as insertion and deletion of a node with a random ID, only require the update of the information on a logarithmic number of nodes. The resulting location service HLS meets the following requirements.

Short paths: For all pairs of source and destination nodes there are short location search path. If nodes are evenly distributed, our basic algorithm looses a minor $O(\log n)$ factor compared to the GLS of Li et al. [11] on the average[1], a loss that becomes even less as the distribution becomes uneven. GLS is superior over the basic HLS only in the locality of search paths; this is partly an inherent effect of keeping location servers with an even geographic distribution to survive systematic faults. However the modular architecture allows easy multi-level solutions that compromise between tolerance of geographical faults and locality of search that match GLS path lengths while improving its fault tolerance.

[1]This factor can be improved to $O(\sqrt{\log n})$, choosing at each node the geographically closest hypercubic neighbor as next location server.

Low degree: Each node should store and update the location of only a very small number of other nodes; symmetrically the location of each node should be stored only at a small number of other nodes. Even simple versions of the HLS – based on the dynamized hypercube – achieve a degree of $O(\log^2 n)$. The practically satisfying $O(\log n)$ degree HLS is based on a dynamized version of the so-called De Bruijn graph [10].

Fault tolerance: Location search is fault tolerant, i.e. a source can find the location of a living destination with high probability, even if a constant fraction of the nodes crashes at the same time. (For the probabilistic model see Section 3).

Survival of attacks: HLS tolerates even the fault of a constant fraction of nodes even in the case when the faults are systematically made under a certain pattern. This desirable property is achieved by making no use of geographic or other device-sensitive information in the location service sublayer. Thus if node IDs are randomly distributed, either as a hash value of the node's IP or other physical address or as a hard-wired random sequence, no systematic attack (or disaster in a region) can disable communication as long as a large fraction of the nodes are alive.

Scalability: Starting with a network containing only one node, we can build a large network by adding new nodes. Also we may turn off most of the nodes and still keep smaller network running. This functionality is based on a novel idea in the theory of hypercubic networks. We introduce the definition of effective bits (effective ID) of a node and define its neighborhood appropriately. We prove several properties of this network and we show how dynamic operations, insertion and deletion of a node with a random ID, can be performed in $O(\log n)$ time with high probability. We are not aware of previous appearance of this idea in the literature.

Self-scaling: Dynamic scalability adds a strong support to fault tolerance. Whenever a fault is detected, the remaining nodes may restructure network topology by removing the non-responding node; later they may again rebuild the original scenario if the node recovers. Network topology updates are local in the logical layer of the location service, thus require only a small amount of administrative traffic over the network. By the simplicity and speed of restructuring the network may survive even a very large scale destruction. In order to provide this feature, alive nodes continuously remove crashed ones and update its own neighborhood.

Load balancing: In a scenario with mobile devices communicating through mobile nodes that act as location servers for the devices, fault tolerant service is provided by letting a few nodes act as the location server of the device. HLS provides a simple method of selecting location servers in a dynamic, fault tolerant and self-scaling manner. We prove that this method achieves asymptotically optimal load balancing with high probability. In order to obtain this result, unlike GLS, we does not need any assumption about the geographic distribution of the nodes.

2 Spanners, Power Spanners, The Yao Graph

Spanners: Spanner graphs have been investigated intensively in the last decades in the computational geometry (see e.g. [4] for a survey) and they has been received a lot of attention as appropriate topologies for wireless ad hoc networks in the last years [2, 6, 7, 8, 12, 13, 21].

For $u,v \in \mathbb{R}^2$, let $||u,v||$ be the Euclidean distance between u and v. Let V be a set of n points in the plane and $t > 1$ be a real constant. Let $G = (V,E)$ be a graph whose edges are straight line segments between pairs of points of V. G is a *Euclidean t-spanner*, short a *t-spanner* for V, if for all pairs of points $u,v \in V$ there exists a (directed) path $P = u,u_1,...,u_{|P|-1},v$ from u to v in G, $u_0 = u$, $u_{|P|} = v$, whose length $||P|| := \sum_{0\le i<|P|} ||u_i,u_{i+1}||$ is at most $t \cdot ||u,v||$, where P denotes the number of edges in P. Such a path is called a *t-spanner path* and t is called the *stretch factor* of G. Spanner graphs guarantee the existence of a short path between any pairs of nodes. There are spanners allowing an efficient distributed construction using only local informations about the nodes. Lot of such spanners also provide us with a simple memoryless routing mechanism, which routes the packets to he destination node along a t-spanner path.

Power spanners: A couple of sparse graph topologies have investigated concerning the energy consumption of the wireless network, where the nodes can vary their transmitting power. The energy to send over distance x is given by a function $\mathrm{pow}(x) := x^d$ for some constant $d \ge 2$ (constant factors are omitted for simplicity).

In [13] the notion of d-power t-spanners has been introduced to measure the quality of different topologies w.r.t. the energy consumption of the network. A graph $G = (V,E)$ is a *d-power t-spanner*, or (d,t)-*power spanner* for V, if for all $u,v \in V$, there is a path $P = u,u_1,...,u_{|P|-1},v$, $u = u_0$, $v = u_{|P|}$, from u to v in G such that $\mathrm{pow}(P) := \sum_{0\le i<|P|} ||u_i,u_{i+1}||^d$ is at most $t \cdot \min\{\sum_{0\le i<m} ||w_i,w_{i+1}||^d \ : \ u = w_0,w_1,...,w_m = v$, $w_0,...,w_m \in V\}$. With other words, the required energy for sending a message along P approximates the energy required on an energy optimal path up to a constant factor t.

For some of the topologies, for which it was known that they are spanners, has been shown that they are also d-power spanners with a certain power stretch factor [13, 20, 21]. Now we prove a theorem which shows a general relation between spanners and power spanners. Applying this theorem we can immediately improve the known bounds on the power stretch factor of some graphs. For the proofs see Appendix B.

Theorem 2.1. *Let $G = (V,E)$ be a Euclidean t-spanner for V and let $0 < c \le t$ be a real constant. If for each pair of nodes $u,v \in V$, there is a t-spanner path $P = u,u_1,...,u_{|P|-1},v$ in G, $u_0 = u$, $u_{|P|} = v$, such that for each edge $u_iu_{i+1} \in P$, $\frac{||u_i,u_{i+1}||}{||u,v||} \le c$, then G is a d-power $c^{d-1}t$-spanner for V.*

Yao graphs: One of the most intensively studied topologies concerning energy consumption and memoryless position based routing is the Yao graph (Θ graph) [22, 5]. It and its variants have numerous applications and a rich history in various fields of computer science [1, 3, 8, 12, 13, 14, 20].

The Yao graph $G_Y(V) = (V,E_Y)$ for a set V of n points in the plane is defined as follows. Let k be an integer and $\theta = 2\pi/k$. Let h_i, $i = 0,...,k-1$, be the halfline coincident with the rotated positive x-axis around the origo by an angle $i\theta$, and let c_i be the cone between h_i and $h_{i+1 \mathrm{mod} k}$. For a point $p \in \mathbb{R}^2$ let $c_i(p)$, $i = 0,...,k-1$, be the translated cone c_i whose apex is at p. For each $p \in V$, let α_p be an angle, $0 \le \alpha_p < \theta$, and let $c'_i(p)$ be the rotated cone $c_i(p)$ around p by an angle α_p. We call the cones $c'_i(p)$ the *sectors of p*. For $p \in V$ and $c'_i(p)$, $i = 0,...,k-1$, let $n_i(p) \in S$ be the Euclidean nearest neighbor of p in $c'_i(p)$, if exists, and connect p to $n_i(p)$ by a directed edge, i.e. $E_Y = \{(p,n_i(p)) \ : \ 0 \le i < k,\ c'_i(p) \cap V \ne \emptyset\}$.

Lemma 2.2. [19] *Let V be a set of n pints in the plane, $k > 6$, and $\theta = 2\pi/k$. Then the Yao graph $G_Y(V)$ is a t-spanner for V with $t = \frac{1}{1-2\sin(\theta/2)}$.*

We remark that the definition of the Yao graph (θ-graph) in [19] is slightly different from our definition. It does not contain the parameter α_p which describes the possibility of the individual rotation of the sectors around the nodes, and it uses a slightly different distance function instead of the Euclidean distance. But each steps of the proof remains true also with our modifications.

The proof of the above lemma in [19] is a constructive one. It constructs a t-spanner path between two arbitrary points $p,q \in V$ as follows. Let $p' := p$. While $p' \neq q$ follow the edge in the sector $c'_i(p')$ which contains q, and set $p' = n_i(p')$.

The above construction of a t-spanner path immediately gives us a simple routing strategy, if following holds. (i) Each node $p \in V$ knows its own geographic position, (ii) the package which has to be routed contains the geographic position of the destination node, and (iii) p knows the geographic position of its Yao graph neighbors $n_i(p)$, $0 \leq i < k$. We call the above routing strategy *sector routing*.

The Yao graph neighbors can be determined efficiently in a distributed and power efficient manner, as described in [21] for a variation of the Yao graph. This distributed construction assumes the existence of a MAC layer, e.g. IEEE 802.11, which resolves interference problems. Furthermore, it is assumed that the mobile nodes use directed radio and they can increase the transmission power (in discrete steps) in the sectors. Each node sends ”hello” signals in each sector with increasing transmission power until at least one node hears them and responds or the maximum power is reached. When a node hears this signal it sends an acknowledge signal which also can contain its geographic position. Then the node only has to choose the closest one among the responding nodes in each sector. The nodes periodically apply this procedure in order to keep their Yao neighbors up to date.

In [13] it has been proved that the Yao graph is a (d,t)-power spanner for $t = (\frac{1}{1-2\sin(\theta/2)})^d$. Since $\theta < \pi/3$, none of the edges $u_i u_{i+1}$ in the t-spanner path constructed by the sector routing can be longer than the Euclidean distance between the source and the destination node. Hence, we can apply Theorem 2.1 with $c = 1$ and we obtain immediately an improvement on the power stretch factor of the Yao graph.

Theorem 2.3. *Let V be a set of n points in the plane, $d \geq 2$, and $0 \leq \theta < \pi/3$. Then the Yao graph is a (d,t)-power spanner with $t = \frac{1}{1-2\sin(\theta/2)}$.*

The t-spanner path produced by the sector routing does not necessary approximate the energy optimal path up to a constant factor. There are distributions of the points, where the energy of such a path can be arbitrarily higher then the energy of the energy optimal path. However, in the case of uniformly distributed points the energy of the sector routing path is not far from the optimum.

3 Location Service

We are ready to describe our Hypercubic Location Service (HLS). For the simplicity of the presentation we reach our final version HLS in simple steps of improvements. The section is organized as follows. In Section 3.1 we define a dynamic version of a hypercube where the number of nodes may change by addition and deletion. In the first version fault tolerance is not supported (removal of a node is for example allowed only after acknowledged by neighbors); we enhance the dynamic hypercube with fault tolerance in Section 3.3 by a load balanced node replication scheme over the hypercube.

In Section 3.2 we take a first step to resolve the problem of impractically high degrees in the location service that would cause an unaffordably high amount of administrative traffic across the network. High degrees of $\omega(\log n)$ result as a cumulative effect of fault tolerance, dynamic updates, and the hypercube itself. We show that a main degree factor stems from the hypercube itself. In a d-dimensional hypercube with sequences of d bits used as node indices, routing can be made by flipping address bits in an arbitrary sequence – a flexibility that we cannot take advantage of in our algorithm. Indeed, we may well just always flip the last bit that differs. As described in Section 3.2, this is exactly what the De Bruijn graph does: it rotates the bits and then flips the last bit if necessary. Thus subsequent nodes over a path may change all bits by always flipping the actual last one.

A replication parameter r of the location service immediately leads to a degree of $4r$ if the HLS is built upon De Bruijn graphs. This degree parameter also appears in the congestion of the Yao graph edges caused by the administrative traffic to update geographic position on the location service neighbors.

Lemma 3.1. *Assume that in a degree d location service graph the average Yao graph path length corresponding to location service edges is ℓ. Then the average congestion of a Yao graph edge caused by geographic position updates is $O(d\ell)$.* □

For a uniform distribution of the nodes in a 2D square $\ell = \sqrt{n}$; in practice however the congestion over the Yao graph is very hard to estimate since we may not assume even distribution.

3.1 Dynamic hypercube

Based on the classical hypercube network [10] we define a dynamic hypercube network that supports the addition and deletion of new nodes by dynamically updating network topology. Updates are kept local which leads to lower communication costs.

First we assign to each node a sufficiently large number of independent random bits as node IDs. Let N be a sufficiently large upper bound on the potential number of nodes. Let there be $n \ll N$ nodes, each with a $\log N$ bit ID. Since not all of the N possible addresses are assigned, we consider only part of the bits acting as an address by the following rule.

> The *effective ID* of a node s is the shortest prefix of k bits of the ID, such that for all nodes that agree with s in the first k bits also agree in the $(k+1)$-st. Note that the $(k+2)$-nd and further bits may or may not be equal.

We define pairs of nodes that communicate their location to each other. For two nodes s_1 and s_2 we say that

> s_2 is a *replicate* of s_1 if their effective IDs are equal;
>
> s_2 is a *type i neighbor* of s_1, $i \geq 1$, if their effective ID differs in exactly one bit, in bit i (allowing that one effective ID is longer and no restriction is imposed upon these bits); in particular if s_1 has k effective bits, then a type k neighbor is called a *last bit neighbor*.

While the type i neighbors are not necessarily unique, we have a high probability bound on their number.

> We say that an event occurs with *high probability* (w.h.p.) its probability is $1 - n^{-c}$, where n is the current number of nodes and $c > 0$ is a constant.

For the proof of the following claims see the Appendix.

Lemma 3.2. *The maximum number of type i neighbors and replicates over all nodes is $O(\log n)$, w.h.p.*

In our network each node only knows its neighbors. This may then effect in messages sent to nonexistent IDs. It it may have several reasons: the node is switched off or it is crashed, or the source called a wrong, nonexistent number. In order to manage this situation, we simply let each node act as a server for all IDs with the same effective ID bits. The distribution of the N full IDs to nodes is unique; we also show that appropriate load balancing is achieved.

Lemma 3.3. *The effective ID of all nodes have at least $\log n - c \log\log n$ and at most $c \log n$ bits, w.h.p.*

Lemma 3.4. *Each node serves for at most $\frac{N}{n} c \log n$ IDs, w.h.p.*

Next we give algorithms for routing and adding/removing nodes from the network. We assume no faults, i.e. in our model each node has an exact knowledge of its neighborhood in the network topology. When adding a node, we must notify all affected nodes about the change and receive acknowledgments before starting operation. Finally when removing a node, we must notify all (affected) nodes and wait for acknowledgments. Algorithms are described with pseudocode in Appendix A.

Routing in a dynamic hypercube proceeds as follows. If the message destination ID equals with the node ID in the $(i-1)$ most significant bits, then the node passes it to a type i neighbor, if exists. We show that the required Type i neighbor always exists; for this assume that the message resides at node s. Let i be the index of the most significant bit where node and destination IDs differ. Then s must have a type i neighbor since the destination ID itself acts as a candidate.

Insertion of a node: When adding node s, we first connect the geographically closest node by gradually increasing the power of a network search signal. Through this node we route a special "wakeup" message in which the the destination field is filled with the ID of s. Since s has not yet joined the network, this message gets stuck at a certain node s'. Let k be the number of effective bits of node s'. By definition, the ID of s is agrees in the first k bits with the ID of s'. Furthermore, each node with the same ID prefix of length k must have the same $(k+1)$-st bit as s'.

Given node s' and the value k, we distinguish two cases. If the ID of s and s' also agree in the $(k+1)$-st bit, then s becomes a new replicate; the neighbors of s are identical to those of s' thus all neighborhood information is available from s'. Node s' may also notify all neighbors of the new neighbor node s.

Finally if the $(k+1)$-st bits differ, then s is a node with no replicate. For $i \leq k$ its type i neighbors are identical to those of s'. In addition s becomes the unique type $(k+1)$ neighbor of s' and of all of its replicates. The original neighborhood of s' now splits between the replicates of s' and s by adding a new effective address bit, following the `add_effective_bit` algorithm (see Appendix) initiated by node $s_0 = s'$.

In both cases s' is capable of updating the entire network view in a single communication round with its neighbors and s. A single message (and acknowledgment) needs to be sent to all neighbors notifying them of the arrival of s and another one to s containing the list of current neighbor locations.

Removing a node: When removing node s, we must notify all of its neighbors. By sending all neighborhood information to all neighbors, they are capable of updating the entire network view with the exception when s has no replicates (other than itself). In this case if s had k bits in its effective ID, then the type k neighbors of s must drop their last effective ID bit.

We mention that in the non-fault-tolerant model, since it may take a while till the network view is updated, a removed node s must perform routing tasks if requested as long as all messages submitted about the removal are confirmed. Once however confirmations are arrived, no neighbor of s will ever try to use s again for routing.

3.2 Low-degree hypercubic networks – De Bruijn graphs

Next we define a dynamic network which we derive from constant degree hypercubic graphs, from De Bruijn graphs [10]. The N-node De Bruijn graphs defined as follows (see Fig. 1). Each node has a unique $\log N$ bit ID. Two nodes are connected if IDs arise as a

1. left shift by one followed by the addition of an arbitrary least significant bit, i.e. an ID $b\beta$ is connected to $\beta 0$ and to $\beta 1$, $b \in \{0,1\}$, $\beta \in \{0,1\}^{\log N-1}$; or
2. right shift by one followed by the addition of an arbitrary most significant bit, i.e. an ID βb is connected to 0β and to 1β.

The N-node De Bruijn graph has degree four (two of both types) and the length of the longest path is $\log N$. Routing (by left shifts) is performed by selecting the largest number i, such that after $\log N - i$ left shifts of the node's ID the first i bits agree with the destination ID. Then the message is passed to a neighbor with left shifted ID where the quantity i increases by one; since the value i increases, the message arrives in at most $\log N$ steps.

Now we define the dynamic De Bruijn graph, which will be the base of our low-degree location service. Dynamic updates in this graph are performed similarly to the hypercube. Network scaling by adding or dropping effective bits is possible by the key property of De Bruijn networks that a d-dimensional De Bruijn graph is the edge graph of the $(d+1)$-dimensional one [10]. Instead of using this fact we give direct proofs of the bit add and drop algorithm correctness (see Appendix).

For each node v, let k_v be the number of effective bits of v. Let G be the graph which contains an edge (u,v) if and only if the $k_{u,v} := \min\{k_u, k_v\}$ most significant ID bits of v can be obtained from the $k_{u,v}$ most significant bits from u either by a left shift and the addition of an arbitrary $k_{u,v}$-th (least significant) bit, or a right shift and the addition of an arbitrary most significant bit. We say that v is a *left neighbor* of u in the former and a *right neighbor* in the latter case. The graph G is called the *dynamic De Bruijn graph*.

Lemma 3.5. *The maximum number of left (right) neighbors over all nodes is $O(\log n)$, w.h.p.*

For the purposes of the discussion we use the terms *replicate* and *last bit neighbor* as in Section 3.1 with the same definition as there. Notice however that last bit neighbors are no longer neighbors in the dynamic De Bruijn graph, they are at least of distance two (siblings of a common right neighbor), or, if their number of effective ID bits differ by δ, they may reside on different paths starting from a common right ancestor of at most δ right neighbor steps.

Routing (by left shifts) in the dynamic De Bruijn graph is performed as follows. For a node v let l_v be a the minimum number of left shifts followed by the addition of an arbitrary k_v-th bit in the effective ID of v, which is necessary to obtain the first k_v bits of the destination. When a packet arrives at a node v, then it will be passed to the left neighbor v' of v, where the value $l_{v'}$ decreases and the decrease is maximal. Then the packet arrives to the destination in at most k_s steps, where k_s is the number of effective bits of the start node s, which is $O(\log n)$.

When a node v with a random ID is inserted (deleted) into the network, the number of effective bits of several nodes may change. Then the neighborhood of some nodes must be accordingly updated. The following two Lemma imply that only neighboring nodes and replicates of v are affected by the change.

Lemma 3.6. (bit addition) *Let G' be a dynamic De Bruijn graph obtained from G, such that the effective k_{s_0} ID bits of an arbitrary node s_0 are extended by an additional $(k_{s_0}+1)$-st effective bit and the edges are updated accordingly. Then the set of left and right neighbors of $s_0 \in G'$ are subsets of those in G.*

Lemma 3.7. (bit removal) *Let G' be a dynamic De Bruijn graph obtained from G, such that the last effective bit of an arbitrary node s_0 is deleted and the edges are updated accordingly. Then the set of left and right neighbors of s_0 in G' are all left and right neighbors of s_0 in G or a last bit neighbor of s_0 in G.*

Lemma 3.8. *The number of nodes affected by the insertion/deletion of a node v is $O(\log n)$, w.h.p.*

Proof. When a node v is inserted, the length of the effective ID only changes at the replicates of v. By Lemma 3.2, their number is $O(\log n)$ w.h.p. Furthermore, the node v and all of its replicates had the same neighbors, whose number, by Lemma 3.5, is $O(\log n)$ w.h.p. Thus, w.h.p., there are $O(\log n)$ nodes, where the change of the number of effective bits must be propagated.

When a node v is deleted, several nodes may become replicates of each other by shortening the effective IDs. By Lemma 3.2, the number of new replicates will be $O(\log n)$ w.h.p. They all will have the same neighborhood, which contains all old neighbors, and whose size, by Lemma 3.5, is $O(\log n)$ w.h.p. $\square$

3.3 Fault tolerance by replication

The dynamic hypercubic networks in the previous sections are prone to faults when nodes unexpectedly crash without passing update information to their neighbors. Now we define a fault-tolerant network topology, which circumvent routing failures by replicating each node in approximately r copies, where r is a prescribed function of the network size. The ID of the replicates is obtained by cutting a certain number of least significant bits from the effective IDs. Type i neighbors are defined accordingly. When a hypercubic neighbor w of a node v crashes, v can route the packages to any of the replicates of w. This replicate has the same neighbors as w had, therefore, it can continue the routing.

Using $r = c\log n$ replicates, if nodes fail with a probability bounded by a constant $p < 1$, one copy of each node will stay alive, and each packet destinated to an alive node can be delivered w.h.p.

Notice that the nodes have no knowledge of n, and hence, cannot compute $\log n$. However by Lemma 3.3 we know that the number of effective bits estimates $\log n$ within a factor of two.

Fault tolerance can be combined with the dynamic updates of the previous subsections. Whenever the routing detects a fault, the network view (i.e. the neighborhood) of the affected nodes is updated as in Section 3.1. In our model we assume that the updates can be performed fast enough that after the update the neighborhood informations are consistent in the network.

For each node s we decide to cut b bits of the effective ID according to the rule that the number of replicates must be between $c_1 r$ and $c_2 r$, for two appropriate constant $c_1 < c_2$. More precisely, we define four constants $c_0 < c_1 < c_2 < c_3$. A node request the addition (deletion) of an effective bit if its view of replicates is above $c_3 \cdot r$ (below $c_0 \cdot r$). All replicates must accept the request if their view is above $c_2 \cdot r$ (below $c_1 \cdot r$).

Finally we prove that, as long as $r = \Omega(\log n)$, the size of the replicates remains $\Theta(\log n)$ after applying the algorithms `add/drop_effective_bit`.

Lemma 3.9. *If the number of replicates of all nodes are $\Theta(r)$ for some $r = \Omega(\log n)$, then the same holds after applying* `add_effective_bit` *or* `drop_effective_bit`.

4 Conclusion

We described an ad hoc mobile routing algorithm that achieves a very high level of fault tolerance while requiring only a low amount of storage and administrative communication. The two ingredients of our solution are a novel location service with an architecture resembling of classical structures in parallel computing [10], bringing closer the ideas of parallel computing and ad hoc networking, and spanner graphs, a structure of computational geometry that has recently received much attention in mobile networking.

Our location service easily cooperates with any geographic routing method or other routing ideas. Further investigation is needed to find the empirically best topology to be used in our algorithm. Here several variants of the Yao graph should be investigated, including sparsified versions as in [13, 20, 21]. We also plan to combine the location service with caching methods such as in the AODV protocol [17] in future work.

The final parameters of our location service for n nodes are summarized as follows.

- *Paths* of *logn* location servers, each step over this path corresponding to a geographic sector routing across the network;
- *Degree* depending on the level of fault tolerance with a minimum of 4 and a value of $c' \log n$ to meet the high probability $1 - n^{-c}$ requirement;
- *Load balancing* with expected optimal value and peaks of a constant factor from optimum, w.h.p.;
- *Fault tolerance* (with high probability $1 - n^{-c}$ at any time) and *scalability* including *self-scaling* fully supported.
- *Survival of systematic attacks* is provided by a random ID distribution (such as hashing IP addresses); the location service ID thus has no relation to the geographic position or other physical notion of the nodes.

References

[1] S. Arya, G. Das, D. M. Mount, J. S. Salowe, and M. H. M. Smid. Euclidean spanners: Short, thin, and lanky. In *ACM Symposium on Theory of Computing (STOC'95)*, pages 489–498, 1995.

[2] J. Bose, P. Morin, I. Stojmenovic, and J. Urrutia. Routing with guaranteed delivery in ad hoc wireless networks. In *Workshop on Discrete Algorithms an Methods for Mobile Computing and Communications (DIALM'99)*, pages 48–55, 1999.

[3] K. L. Clarkson. Approximation algorithms for shortest path motion planning (extended abstract). In *ACM Symposium on Theory of Computing (STOC'87)*, pages 56–65, 1987.

[4] D. Eppstein. Spanning trees and spanners. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, chapter 9, pages 425–461. Elsevier, 2000.

[5] H. N. Gabow, J. L. Bentley, and R. E. Tarjan. Scaling and related techniques for geometry problems. In *ACM Symposium on Theory of Computing (STOC'84)*, pages 135–143, 1984.

[6] J. Gao, L. J. Guibas, J. Hershberger, L. Zhang, and A. Zhu. Discrete mobile centers. In *ACM Symposium on Computational Geometry (SCG'01)*, pages 188–196, 2001.

[7] J. Gao, L. J. Guibas, J. Hershberger, L. Zhang, and A. Zhu. Geometric spanner for routing in mobile networks. In *ACM Symposium on Mobile Ad Hoc Networking and Computing (MOBIHOC'01)*, pages 45–55, 2001.

[8] Y. Hassin and D. Peleg. Sparse communication networks and efficient routing in the plane. *Distributed Computing*, 14(4):205–215, 2001.

[9] D. B. Johnson and D. A. Malz. Dynamic source routing in ad hoc wireless networks. In *Mobile Computing*, volume 353. Kluwer Academic Publishers, 1996.

[10] T. Leighton. *Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*. Morgan Kaufmann, 1992.

[11] J. Li, J. Jannotti, D. S. J. De Couto, D. Karger, and R. Morris. A scalable location service for geographic ad hoc routing. In *ACM International Conference on Mobile Computing and Networking (MobiCom'00)*, pages 120–130, 2000.

[12] X.-Y. Li, G. Calinescu, and P.-J. Wan. Distributed construction of a planar spanner and routing for ad hoc wireless networks. In *IEEE INFOCOM*, 2002.

[13] X.-Y. Li, P.-J. Wan, and Y. Wang. Power efficient and sparse spanner for wireless ad hoc networks. In *IEEE International Conference on Computer Communications and Networks (ICCCN'01)*, 2001.

[14] T. Lukovszki. New results of fault tolerant geometric spanners. In *Workshop on Algorithms and Data Structures (WADS'99)*, pages 193–204, 1999.

[15] M. Mauve, J. Widmer, and H. Hartenstein. A survey on position-based routing in mobile ad hoc networks. *IEEE Network Magazine*, 15(6):30–39, 2001.

[16] V. D. Park and M. S. Corson. A highly adaptive distributed routing algorithm for mobile wireless networks. In *IEEE INFOCOM*, pages 1405–1413, 1997.

[17] C. E. Perkins, E. M. Belding-Royer, and S. Das. Ad hoc on demand distance vector (AODV) routing. In *IETF Internet Draft, (Work in Progress)* `draft-ietf-manet-aodv-09.txt`, November 2001.

[18] C. E. Perkins and P. Bhagwar. Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. In *ACM SIGCOMM '94 Conference on Communications Architectures, Protocols and Applications*, pages 234–244, 1994.

[19] J. Ruppert and R. Seidel. Approximating the d-dimensional complete Euclidean graph. In *3rd Canadian Conference on Computational Geometry (CCCG'91)*, pages 207–210, 1991.

[20] Y. Wang and X.-Y. Li. Distributed spanner with bounded degree for wireless ad hoc networks. In *Parallel and Distributed Computing Issues in Wireless networks and Mobile Computing*, 2002.

[21] R. Wattenhofer, L. Li, P. Bahl, and Y.-M. Wang. Distributed topology control for power efficient operaion in wireless multihop ad hoc networks. In *IEEE INFOCOM*, 2001.

[22] A. C. Yao. On constructing minimum spanning trees in k-dimensional spaces and realted problems. *SIAM Journal on Computing*, 11(4):721–736, 1982.

Appendix

A Algorithms

`add_effective_bit`

```
Initiator node $s_0$:
1.  notify replicates;
2.  if confirmation arrives from (all or prescribed fraction) nodes then
3.      notify all replicates and Type $i$ neighbors;
4.      for all $i$ where Type $i$ neighbors have more effective ID bits than $s_0$ do
5.          drop neighbors that do not agree in new effective bit;

Replicate $s$ of $s_0$:
1.  if second notice arrives from $s_0$ then
2.      for all $i$ where Type $i$ neighbors have more effective ID bits than $s$ do
3.          drop neighbors that do not agree in new effective bit;

Type $i$ neighbor $s$ of $s_0$:
1.  if first notice arrives from $s_0$ and $s_0$ has less effective bits than $s$ then
2.      drop Type $i$ neighbors that do not agree in new effective bit;
```

`drop_effective_bit`

```
Initiator node $s_0$ with $k$ effective ID bits:
1.  notify replicates and Type $k$ neighbors;
2.  if confirmation arrives from (all or prescribed fraction) nodes then
3.      merge neighbor lists received;
4.      send neighbor list to replicates and Type $k$ neighbors;
5.      send list of replicates and Type $k$ neighbors to all Type $i$ neighbors;

Replicate or Type $k$ neighbor $s$ of $s_0$:
1.  send neighbor list to $s_0$
2.  if second notice arrives from $s_0$ then
3.      merge new neighbor list;

Type $i$ neighbor $s$ of $s_0$:
1.  merge Type $i$ neighbor list;
```

B Proofs

Proof of Theorem 2.1. Consider two nodes $u, v \in V$. Let $P^* = u, u_1, \dots u_{|P^*|-1}, v$ $u_0 = u$, $u_{|P^*|} = v$, be an energy optimal path from u to v. The edges of P^* are not necessary contained in G, for each edge $u_i u_{i+1} \in P^*$ there is a t-spanner path $P_i = u_i, u_i^1, \dots, u_i^{|P_i|-1}, u_{i+1}$ in G, $u_i^0 = u_i$, $u_i^{|P_i|} = u_{i+1}$ with $\frac{||u_i^j, u_i^{j+1}||}{||u_i, u_{i+1}||} \le c$ for all $0 \le j < |P_i|$. Then

$$\begin{aligned}
\mathrm{pow}(P_i) &= \sum_{0 \le j < |P_i|} \mathrm{pow}(u_i^j, u_i^{j+1}) = \sum_{0 \le j < |P_i|} ||u_i^j, u_i^{j+1}||^d \\
&\le \sum_{0 \le j < |P_i|} ||u_i^j, u_i^{j+1}|| \cdot (c||u_i, u_{i+1}||)^{d-1} \\
&= c^{d-1} ||u_i, u_{i+1}||)^{d-1} \sum_{0 \le j < |P_i|} ||u_i^j, u_i^{j+1}|| \\
&\le c^{d-1} ||u_i, u_{i+1}||^{d-1} \cdot t ||u_i, u_{i+1}|| \\
&= c^{d-1} t \cdot \mathrm{pow}(u_i, u_{i+1}). \qquad (1)
\end{aligned}$$

Consider the path $P = P_0 \circ \dots \circ P_{|P^*|-1}$ from u to v in G. Then after applying (1) to each P_i we obtain that $\mathrm{pow}(P) = \sum_i \mathrm{pow}(P_i) \le c^{d-1} t \sum_i \mathrm{pow}(u_i, u_{i+1}) \le c^{d-1} t \cdot \mathrm{pow}(P^*)$. This proves the claim. □

Proof of Lemma 3.2. We use standard probability calculation. By selecting N as a sufficiently large upper bound on the number of nodes and randomly distributing IDs we assume each bit of each ID is selected independently at random. Hence we will assume a fixed ID for node s and randomly select ID both of other nodes in an appropriate order of our choice.

To prove for replicates we fix node s and select the ID bits of all other nodes bit by bit. The procedure stops when the k effective bits are selected. Let r be the number of nodes having the same k most significant bits as s. In the $(k+1)$-st selection round all of this r nodes obtain the same bit as s. The probability of this event is 2^{-r}, implying $r = O(\log n)$, w.h.p.

For type i neighbors we follow a similar process except for skipping bit i in the selection process. Let s be a fixed node with k effective bits. Let r be the number of nodes having the same bits as s at positions $1, \dots, i-1, i+1, \dots, k$, and an arbitrary bit at position i. These nodes are either replicates or Type i neighbors of s. Now we consider the selection of the $(k+1)$-st bit together with the i-th. Choosing an i-th bit different from the i-th bit of s the $(k+1)$-st bit can be selected independently. But if we choose the same i-th bit as the i-th bit of s then we only have a single choice for the $(k+1)$-st bit: it also must be equal to the $(k+1)$-st bit of s. The probability of this event is $(3/4)^r$, implying $r = O(\log n)$, w.h.p. □

Proof of Lemma 3.3. For the lower bound we may appropriately modify the constants to assume $n = 2^r$. We may model the selection of the most significant r ID bits as randomly distributing n balls into n bins. Notice that an effective ID of $r - c\log r$ bits means that $2^{c \log r} = r^c$ consecutive bins remain empty, started with an index $i \cdot r^c$, $0 \le i < \frac{n}{r^c}$. This has a probability $\frac{n}{r^c} \cdot (1 - \frac{r^c}{n})^n \le \frac{n}{r^c} \cdot e^{-r^c} = n^{-c'}$.

For the upper bound we consider a pair of type $c \log n$ neighbors. The probability that $c \log n - 1$ bits of their ID are equal and while the last are different is $2^{-c \log n} = n^{-c}$. The claim follows. □

Proof of Lemma 3.4. The upper bound follows by Lemma 3.3. □

Proof of Lemma 3.5. We prove the claim for left neighbors. The proof for right neigbors is analogous. We fix a node s. Let s' be a left neighbor of s having a minimum number of effective bits among the left neighbors of s. Let $k = \min(k_s, k_{s'})$ be the minimum number of effective bits of s and s'. Let $Lk-1$ be the set of nodes having the same $k-1$ most significant bits as s'. Clearly, all left neighbors of s are contained in L_{k-1}. Note, L_{k-1} may also contain nodes which are not left neighbors of s. Let r be the number of nodes in L_{k-1}. It is an upper bound on the number of left neighbors of s. In order to give a bound on r, we again use the model of distributing n balls into n bins uniformly at random. Then determining r is equivalent to the counting the balls in consecutive bins, whose index start with the same $\max(\log n, k-1)$ most significant bits as the ID of s'. By Lemma 3.3, $k \geq \log n - c \log\log n$, w.h.p. Therefore, we have to count the balls in at most $c' \log n$ bins. By a Chernoff bound we obtain that the number r of balls in at most $c' \log n$ consecutive bins is $O(\log n)$, w.h.p. Consequetly, also the probability of the joint event $k \geq \log n - c \log\log n$ and $r = O(\log n)$ is at least $1 - n^{-c''}$. $\square$

Proof of Lemma 3.6. If a node s has $k_s \leq k_{s_0}$, then their neighborhood relation remains unchanged. Hence we may let $k = k_{s_0}$ and assume $k_s \geq k+1$; let the $(k+1)$-st, k-th, and $(k-1)$-st ID bits of s_0 be a, b and c, respectively.

In the above setting we consider the possible $(k+1)$-st, k-th, and $(k-1)$-st ID bits of a left neighbor s of s_0. In G these bits are bxy where x and y are arbitrary while in G' they are bcz where z is arbitrary. Since the remaining most significant bits are uniquely determined, we see the neighbors in G with $x = c$ remain neighbors in G' while the remaining are not.

We may resolve right neighbors by a similar argument. Let abc be as before; now a right neighbor in G has an arbitrary most significant bit and a k-th bit with value a while in G' we have k-th and $(k+1)$-st bits ab instead. Again s remains a neighbor if its $(k+1)$-st bit is b and we have no other neighbors. $\square$

Proof of Lemma 3.7. If a node s has $k_s < k_{s_0}$, then their neighborhood relation remains unchanged. Hence we may let $k = k_{s_0}$ and assume $k_s \geq k$; let the $(k-2)$-nd, $(k-1)$-st, and k-th ID bits of s_0 be a, b and c, respectively.

By notifying last bit neighbors we may identify all replicates of node s that arise by complementing the k-th ID bit of s. Then all new left neighbors of s_0 are neighbors of the new replicate s, hence after finding the s, we are done by identifying new neighbors in a second communication round.

We may resolve right neighbors by a similar argument. There is no change if the right neighbors have less than k ID bits. Otherwise the new right neighbors are the last bit neighbors of the old right neighbors. $\square$

Proof of Lemma 3.9. The first part follows by a Chernoff bound for the number of nodes with given new effective bit 0 or 1. And the second part follows by a similar argument as in Lemma 3.2 since the new replicates are all last bit neighbors. $\square$

C Figures

We included the screenshot of our tests of a 256-node fault tolerant degree 16 hypercubic location service described in Section 3.2. We distributed nodes uniform at random within a square and built the Yao graph below (Figure 2).

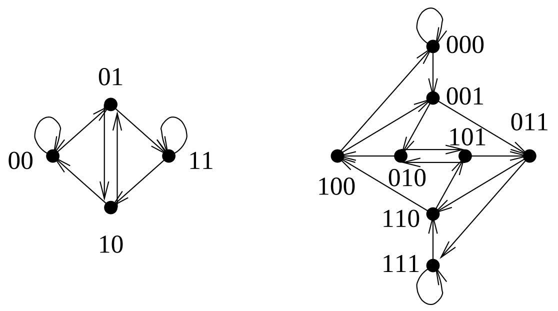

Figure 1: The low-degree location service base networks for $N = 4$ and $N = 8$. Arrows point from right neighbor towards left neighbor.

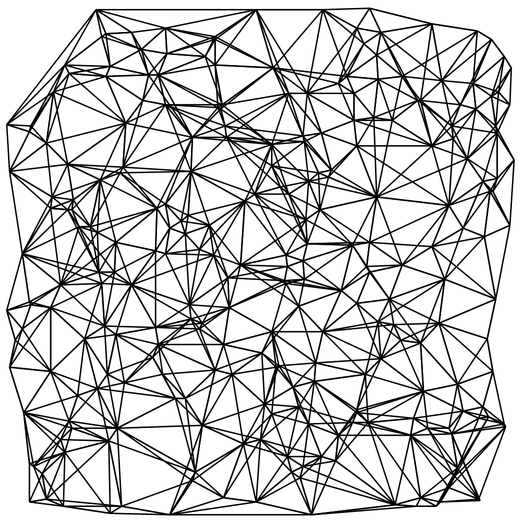

Figure 2: The Yao graph of the 256 nodes distributed uniformly at random within a square.

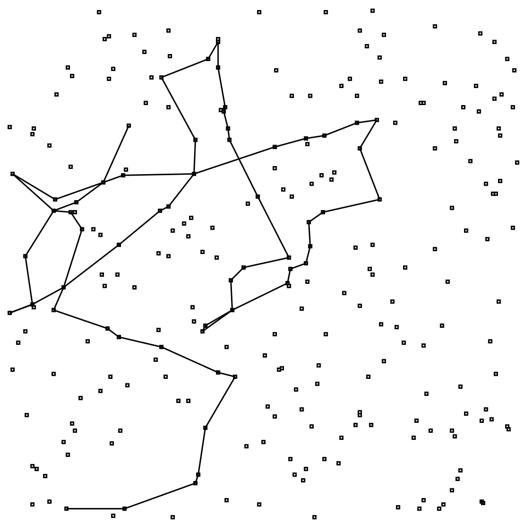

Figure 3: The location path in the De Bruijn graph based HLS, when choosing an arbitrary neighbor

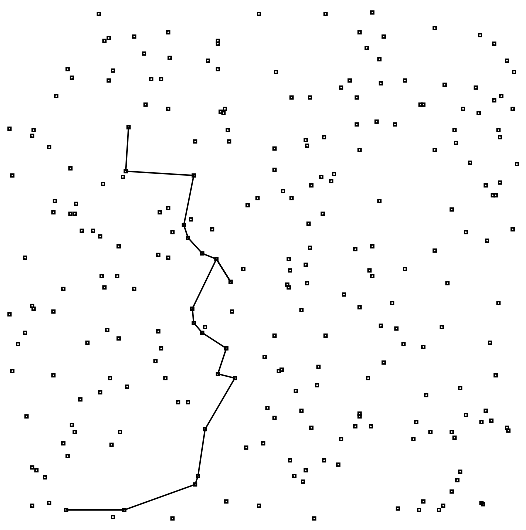

Figure 4: The location path in the De Bruijn graph based HLS, when choosing the closest neighbor

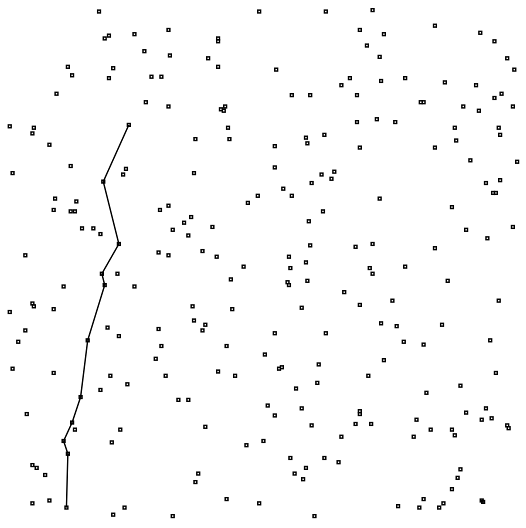

Figure 5: The Yao graph path obtained by sector routing