WEAVE: Efficient Geographical Routing in Large-Scale Networks

Michał Król
University of Grenoble Alps,
Grenoble Informatics Laboratory
Grenoble, France
Krol@imag.fr

Eryk Schiller
University of Bern
Bern, Switzerland
Schiller@iam.unibe.ch

Franck Rousseau, and
Andrzej Duda
University of Grenoble Alps,
Grenoble Informatics Laboratory
Grenoble, France
{Rousseau, Duda}@imag.fr

Abstract

We propose WEAVE, a geographical 2D/3D routing protocol that maintains information on a small number of *waypoints* and *checkpoints* for forwarding packets to any destination. Nodes obtain the routing information from partial traces gathered in incoming packets and use a system of checkpoints along with the segments of routes to *weave* end-to-end paths close to the shortest ones. WEAVE does not generate any control traffic, it is suitable for routing in both 2D and 3D networks, and does not require any strong assumption on the underlying network graph such as the Unit Disk or a Planar Graph. WEAVE compares favorably with existing protocols in both testbed experiments and simulations.

Categories and Subject Descriptors

C.2.2 [**Network protocols**]: Routing protocols

General Terms

Design, Measurement, Performance

Keywords

Wireless Sensor Networks, Geographical routing

1 Introduction

The paper concerns geographical routing in large-scale networks that forward traffic in a multi-hop way and exhibit dynamic behavior—links may go up and down, nodes may join and leave the network. Good examples are Wireless Mesh Networks (WMNs), or Wireless Sensor Networks (WSNs) with nodes that communicate using various wireless technologies and know their geographical coordinates (either 2D or 3D).

In such networks, *geographical greedy forwarding*, in which a node forwards an incoming packet to the neighbor that is the closest to the destination, presents many advantages [10, 25]: it scales as $O(1)$ with the network size, there is no signalling overhead (no need for control traffic) [10], and the route stretch is low [18] (we define the route stretch as the ratio of a given route to the shortest path), which is especially important for extending the lifetime of wireless sensor networks. However, it only works in networks with sufficient density without coverage defects such as voids, concave regions, or obstacles [1, 24]. Much research effort resulted in protocols that tried to fix greedy forwarding and guarantee packet delivery with *face routing* based on the assumption of the Unit Disk Graph (UDG) [7] through Gabriel Graph [11] planarization: Greedy-Face-Greedy (GFG) [3, 4] and Greedy Perimeter Stateless Routing (GPSR) [12] (with many other variants GOAFR [16], GOAFR+ [15], GPVFR [19]). However, Kim *et al.* [13, 14] showed that the Planar Graph required for face routing cannot be established locally. Their Cross-Link Detection Protocol (CLDP) requires expensive signalling for detecting and removing crossed edges. GDSTR (Greedy Distributed Spanning Tree Routing) was another approach for surrounding voids or obstacles based on *convex hulls* [18].

Instead of proposing a workaround in the case of problems, we adopt a radically different approach based on constructing segments of routes with a little stretch and combining them together into end-to-end routes. In this paper, we propose WEAVE, a geographical 2D/3D routing protocol that maintains information on a small number of *waypoints* and *checkpoints* for forwarding packets to any destination. Nodes obtain the routing information from partial traces gathered in incoming packets and use a system of checkpoints along with the segments of routes to *weave* end-to-end paths close to the shortest ones. WEAVE does not generate any control traffic, it is suitable for routing in both 2D and 3D networks, and does not require any strong assumptions on the underlying network graph such as the Unit Disk or a Planar Graph.

Geographical routing becomes increasingly important because of the advent of large scale deployments of sensors and things in the future Internet of Things and mobile devices, for which the knowledge of the location is required in various services. Many devices already have GPS and even in the absence of GPS, the location information can be obtained from relative or virtual positioning based on estimation of the signal strength.

The contribution of this work is threefold. First, WEAVE significantly reduces hop stretch in 2D or 3D networks and

International Conference on Embedded Wireless
Systems and Networks (EWSN) 2016
15–17 February, Graz, Austria

ISBN: 978-0-9949886-0-7

establishes routes with a high probability (cf. Sec. 5). It can thus replace greedy forwarding in many previously proposed protocols such as GDSTR or GDSTR-3D that guarantee packet delivery. Second, we prove our forwarding scheme loop-free even if nodes only use capped partial traces (i.e., a few next hops along the optimal route towards a destination) (cf. Sec. 4). Third, we evaluate WEAVE through measurements on a sensor network testbed in real wireless conditions (cf. Sec. 5.1) and through simulations for various network sizes (cf. Sec. 5.2). The results show that our protocol only uses a very small volume of routing information ($O(\log N)$) and achieves high packet delivery rate (close to 100%), low routing stretch (1.4) and energy consumption well distributed over nodes compared to existing protocols.

2 Related work

We have already reviewed the main geographic routing protocols in the introduction and this section describes other approaches closely related to WEAVE. De Couto and Morris defined Intermediate Node Forwarding: they extended DSDV to forward packets around bad geographic topologies via intermediate locations [9]. The Terminode project defined the Anchored Geodesic Packet Forwarding based on a source path method that uses a list of fixed geographical points called *anchors* [2]. Packets loosely follow the anchored path to reach a destination. Lim *et al.* developed Landmark Guided Forwarding (LGF), a protocol that mixes topological and geographical routing algorithms [20]. Nodes in LGF only maintain a small amount of topological information about their neighbors within a localized area. With respect to destinations residing in a local area, packets are forwarded using the shortest path algorithm. For remote destinations, LGF forwards packets based on loose-source routing to a geographically determined optimal Landmark node. The authors showed that LGF is adaptive to unstable connectivity and scalable to large networks. In our previous work [22], we proposed an initial scheme based on *waypoints* to improve the performance of geographic forwarding, however the proposed protocol was unfeasible: it requires recording entire packet traces (a trace contains all intermediate nodes of a given route) and relies on source routing—each packet holds the entire route to a chosen waypoint.

Another research area was the extension of 2D protocols to 3D spaces. Zhou *et al.* extended the previous 2D geographical protocols (CLDP/GPSR and GDSTR) to 3D [27]. Virtual Ring Routing (VRR) [6] is a protocol based on overlay routing algorithms in Distributed Hash Tables. It establishes a virtual ring, in which nodes became neighbors based on their virtual identifiers that are independent of the geographical positions. A node maintains forwarding entries to its successor and predecessor over underlying multihop physical paths. To reach their destinations, packets are greedily forwarded towards the node with the closest ID to the destination. VRR is resistant to network dynamics, but results in a high hop stretch and message overhead especially for large-scale networks. Small State and Small Stretch (S4) aims at large static wireless networks [21]. It is based on a random set of landmark nodes (beacons). A node maintains a routing table to all beacons and nodes in a local cluster. S4 provides high delivery ratio and a low hop stretch. However to operate properly, S4 requires a large number of beacons that need to be known by all other nodes. Beacons continuously flood the network, which may result in a significant overhead and concentration of energy consumption.

GDSTR-3D adapts the classic version of the protocol to the 3D environment and favorably compares with other protocols that may operate in 3D spaces such as CLDP/GPSR, GDSTR, VRR [6], and S4 [21] from the point of view of the performance, route stretch, and memory usage. The MDT (Multi-hop Delaunay Triangulation) protocol [17] creates Delaunay Triangulation (DT) graphs and virtual links to greedily route packets in any n-dimensional space. We compare WEAVE with MDT and GDSTR-3D in the evaluation section.

3 WEAVE Protocol

This section starts with a high-level overview of the protocol and corresponding subsections provide more details.

3.1 Protocol Overview

We adopt usual assumptions in geographical routing protocols: we assume that nodes know their coordinates and can exchange packets with some neighbors. However, unlike previous approaches such as face routing, we do not require any Unit Disk assumptions nor other properties of the underlying network graph (e.g. Planar Graph). The only requirement concerns the knowledge of neighboring nodes with whom a node has symmetrical links. The discovery of neighbors and symmetrical links of good quality depends on an underlying metric at the link layer that can be based on well studied approaches such as ETX [8]. The issue of the most suitable metric for constructing a symmetric neighborhood is out of the scope of this work.

For simplicity, we present the routing protocol principles for the simpler 2D case, however generalization to 3D is straightforward. Thus, we assume that each node lies inside a finite square address space:

$$\mathcal{A} = [x_{min}, x_{max}] \times [y_{min}, y_{max}] \quad (1)$$

and knows its geographical position $a_n = (x_n, y_n)$, a pair of coordinates such that $x_{min} \leq x_n \leq x_{max}$ and $y_{min} \leq y_n \leq y_{max}$. In the rest of the paper, we denote a node by its address a_n.

As shown in Figure 1, each node a_n builds a partition of the address space, resulting in disjoint subsets $\mathcal{P}_n^j$ called *regions*. Farther regions are bigger. Nodes maintain the information about one or several *waypoints* per region. A waypoint will serve as an intermediary node to reach destination a_d in a given region. Nodes choose regions and waypoints independently so they may be different for each node in the network.

At the beginning, routing tables are empty and nodes forward packets using greedy forwarding. Every packet keeps a trace of h_l last hops (*cf.* routing header in Figure 1). A node receiving the packet can take its source node as the *waypoint* for the region of the source node and record its partial route in the routing entry for the region.

Waypoints stored in the routing tables can then be used to forward traffic. Each node sending a packet checks whether it has a waypoint in the same region as the destination. In

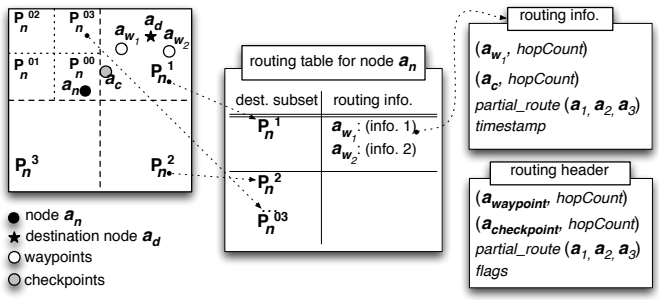


Figure 1: Principles of WEAVE

such a case, it stores the waypoint with its partial route in the packet header. The packet will be then forwarded using the partial route. Each intermediary node can update the partial route or change the waypoint if it has a better one. For instance, node a_n in Figure 1 sends a packet to destination a_d lying in region $\mathcal{P}_n^1$ by using waypoint a_{w_1} and the routing information about partial route a_1, a_2, a_3 towards a_{w_1}. Then, the packet follows the partial route and each intermediary node can refresh or improve the waypoint or the partial route, so the packet gets closer to the destination. A node uses greedy routing as a fall-back when it does not have the information on a waypoint and a partial route. To improve efficiency in large-scale networks, we introduce *checkpoints* that act as "bread crumbs". Checkpoints are chosen among nodes that lie on the border between different regions.

The subsections below present the details of the protocol. In the description, we sometimes distinguish between the *learning* and *working* phases: the learning phase stands for the initial stage of the WEAVE operation when most of routing tables are empty and nodes use greedy routing to forward packets. The working phase corresponds to the later stage when routing tables are filled and WEAVE can efficiently forward packets using partial routes. Nevertheless, there is no distinction between these phases in the protocol: WEAVE always uses *waypoints* if it finds one and never stops learning by trying to update routing tables looking for better routes.

3.2 Packet Structure

The header of WEAVE packets contains the source node, a partial route (used for forwarding) and a partial trace of the last h_l hops (used for learning routes) (cf. Fig. 2). An intermediate node that forwards a packet can use the partial trace to fill its routing table. It fills the partial route with the information from the routing table if available. When the routing table of a node is empty or it does not contain a valid *waypoint* for a given destination, the node leaves the partial route field empty. Nodes also use *checkpoints* stored in packets. We further explain this mechanism in Section 3.8.

Section 5.1 presents a comparison between the WEAVE header and other protocols. For each protocol, we assume a 3B field to store geographic locations. Note that in WEAVE, 1B fields are sufficient to store the hop ID (cf. Fig. 2), as it is a local identifier known by direct neighbors of a node.

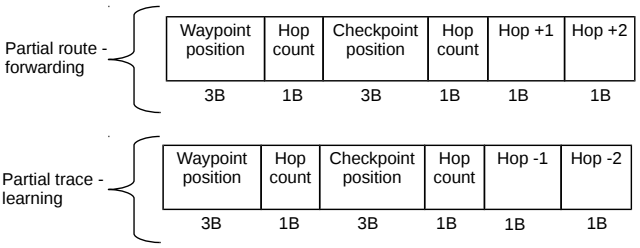


Figure 2: WEAVE packet structure for $h_l = 2$.

3.3 Principles of Packet Forwarding

Algorithm 1 defines the operation of a node that forwards packets. When the node receives a packet, it first looks up its routing table for a better waypoint (closer to the destination) to replace the one included in the packet. If it does not find such a waypoint, it looks up for a checkpoint present in the packet to replace the partial route that runs out with a longer one heading in the same direction. If it is unsuccessful, the node uses greedy routing to forward the packet towards its checkpoint, its waypoint, or the destination node. Finally, if the node still cannot forward the packet, it uses path exploration and backtracking that are detailed later on.

if *better waypoint found in routing table* **then**
  update the information in the routing header:
  (*waypoint*, *checkpoint*, *partial_route*);
else if *the same checkpoint found in routing table* **then**
  update *partial_route* in the routing header;
end
if *partial_route is not* $\emptyset$ **then**
  next_hop $\longleftarrow$ first node in *partial_route*;
else if *packet has checkpoint* **then**
  next_hop $\longleftarrow$ **greedy**(*checkpoint*);
else if *packet has waypoint* **then**
  next_hop $\longleftarrow$ **greedy**(*waypoint*);
else
  next_hop $\longleftarrow$ **greedy**(*destination*);
end
if *next_hop is* $\emptyset$ **then**
  pathExploration();
  backtracking();
end

Algorithm 1: WEAVE forwarding algorithm

Fig. 3 illustrates the principle of packet forwarding. A node maintains one or several waypoints as representatives of regions in the address space. When a node has a packet to forward to destination a_d, it determines which of its waypoints is the closest one to a_d. In our example, source a_s knows a_{w_1} as the waypoint to reach a_d, so it sends the packet towards a_{w_1} along the stored partial route. Intermediate node a_{i_1} knows waypoint a_{w_2} as a representative of the region where the final destination a_d lies, and since it is closer to a_d, it changes the packet direction to a_{w_2}. The same operation happens at intermediate node a_{i_2}, and finally, the packet reaches the destination.

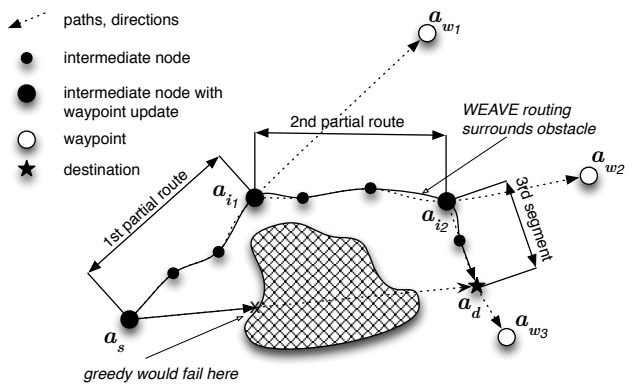


Figure 3: Principle of packet forwarding

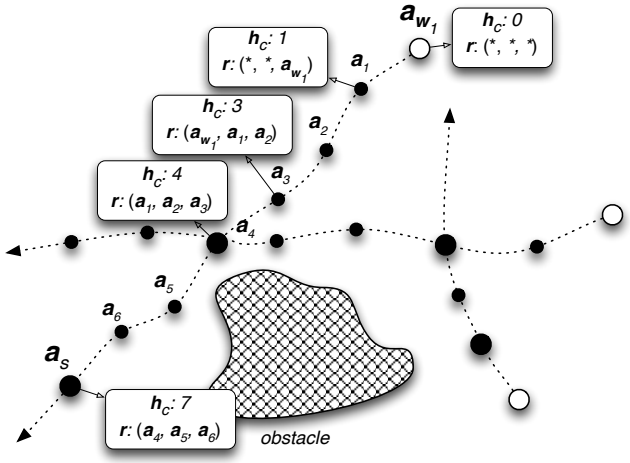


Figure 4: Learning partial routes

3.4 Learning Partial Routes

Fig. 4 illustrates the principle of learning partial routes. When nodes do not have yet sufficient information on waypoints (e.g. at the beginning of their operation), they use greedy geographical forwarding.

Each packet registers a *partial trace* r: a list of nodes limited to the last h_l hops. h_l is a protocol parameter set to a small value (e.g. it varies from 3 to 5 in our simulations). A packet also contains counter h_c strictly increasing at every hop alongside the route. Consider an example of a packet sent from a_{w_1} that reaches a_s at some point after going through six intermediate nodes a_i, $i = 1,\ldots,6$. Assume that $h_l = 3$. The partial trace is (a_{w_1}) at a_1, (a_{w1}, a_1, a_2) at a_3, and (a_1, a_2, a_3) at a_4. Note that node a_3 has deleted a_{w1} from the trace and added itself, because the trace size is limited to 3 nodes. a_s may choose a_{w1} as a waypoint for the region in which a_{w1} lies and stores the trace contained in the packet. The fact that a_s receives the packet guarantees that it can reach a_{w1}, because we only use symmetric links for packet forwarding: if a node receives a packet from a_{w1}, it can reach a_{w1} on the reverse route.

Note that storing only the last hop in the a partial trace can be insufficient. It is possible that the previous node already had another waypoint for a given subspace and did not register the one a node puts in its routing tables. Storing several last hops does not introduce significant overhead and greatly increase routing efficiency (cf. Sec. 5).

3.5 Address Space Partitioning

To manage waypoints, nodes split the address space into regions and assign waypoints to every region. In 2D, every node a_i applies a *quadtree partitioning* process $P_{quadtree}$ to partition the address space $\mathcal{A}$ into a set of disjoint subsets $\mathcal{P}_i^j$:

$$P_{quadtree}(\mathcal{A}) \rightarrow \begin{pmatrix} \mathcal{P}_i^1, \mathcal{P}_i^2, \mathcal{P}_i^3 \\ \mathcal{P}_i^{01}, \mathcal{P}_i^{02}, \mathcal{P}_i^{03} \\ \vdots \\ \mathcal{P}_i^{0\ldots0} \end{pmatrix} \tag{2}$$

These subsets cover the whole address space: $\bigcup_j \mathcal{P}_i^j = \mathcal{A}$. To extend our protocol to 3D case, we only need to use *octree partitioning* in which subsets $\mathcal{P}_i^j$ are cubes and add z coordinate to node positions. The partitioning scheme is essential for forwarding to checkpoints (cf. Sec. 3.8).

At the beginning, each node a_i discovers its neighborhood denoted as Nbrhood[a_i]—a set of all directly reachable neighbors—and estimates its neighborhood diameter d_l used as the termination criterion. d_l is defined as twice the geographical distance to the farthest neighbor or ∞ if the neighborhood is empty (the node does not have any neighbor):

$$d_l = \begin{cases} 2 \max_{a_j} |a_i, a_j|, & a_j \in \text{Nbrhood}[a_i] \\ \infty, & \text{Nbrhood}[a_i] = \emptyset \end{cases} \tag{3}$$

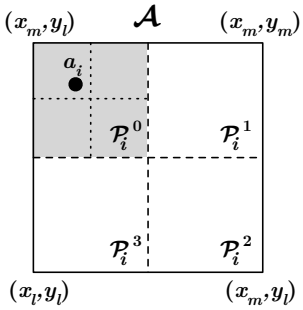


Figure 5: Quadtree address space partitioning

Fig. 5 illustrates the first step of the partitioning process in which node a_i divides $\mathcal{A}$ into four regions: $\mathcal{P}_i^0$ that contains node a_i and three other regions $\mathcal{P}_i^1, \mathcal{P}_i^2$, and $\mathcal{P}_i^3$. Next, the node repeats partitioning of $\mathcal{P}_i^0$, if Edge $\left[\mathcal{P}_i^0\right] > d_l$, which results in $\mathcal{P}_i^{00}$ and $\mathcal{P}_i^{01}, \mathcal{P}_i^{02}, \mathcal{P}_i^{03}$, and so on. The process continues until Edge $\left[\mathcal{P}_i^{0\ldots0}\right] \leq d_l$.

Note that every node has its own view of the address space: although the symbolic hierarchy is the same, the physical regions assigned to the $P_{quadtree}(\mathcal{A})$ hierarchy may be different for every node a_i. Node a_i will consider all nodes outside $\mathcal{P}_i^0$ as reachable through waypoints in each region

$\mathcal{P}_i^1, \mathcal{P}_i^2, \mathcal{P}_i^3$. Nodes inside $\mathcal{P}_i^0$, will be reachable through waypoints in subregions at the lower level, recursively, e.g. $\mathcal{P}_i^{01}$, $\mathcal{P}_i^{02}$, $\mathcal{P}_i^{03}$, and so on.

With this construction, every node builds a scalable representation of the geographical address space, has a coarse grain representation of distant regions, more precise information of the regions that are closer, and a fine grain representation of its surroundings. Note that this representation is different from approaches taken by hierarchical protocols that build a single common global hierarchy for the whole network.

3.6 Constructing Routing Tables

To forward packets, each node maintains a routing table containing up to $\mathcal{L}$ *waypoint routing entries* per region—node a_i has to know the waypoint to use for destination address a_d that lies in a given region $\mathcal{P}_i^\star$. When node a_i receives a packet from source $a_s \in \mathcal{P}_i^\star$ with *partial trace* r, *checkpoint* a_c, and *hop counter* h_c, it creates a waypoint routing entry $w = (a_w, H_w, h_c, r_w, a_c)$ containing five fields: waypoint address $a_w = a_s$, waypoint metric $H_w = |a_i, a_w| / h_c$, partial route $r_w = r^{-1}$, and checkpoint a_c described later on.

Metric H_w reflects the "quality" of a waypoint: we want to keep a set of waypoint entries with the largest H_w, because in this case, packets cross long distances per hop count. Note that the shortest path between a_s and a_d computed by OSPF would have the maximal value of H_w for this pair of nodes. The metric allows to memorize and route along OSPF-like paths, which is good, because it improves the routing performance, i.e. route stretch is small. We have tested other waypoint metrics such as: $\min|a_i, a_w|$, $\max|a_i, a_w|$, no metric (we just store last $\mathcal{L}$ entries). In all cases, we have obtained a lower reachability ratio and longer routes than for H_w.

Note also that maximizing H_w does not mean that nodes will suffer from poor performance due to long wireless links of low quality [23]: in our case, metric H_w is only applied to routes and not to links—nodes discover their neighbors using a link layer metric and choose only good quality symmetrical links.

A node may store several waypoint entries $W_{\mathcal{P}_i^\star} = \{w_1, \ldots, w_k\}$, $k \leq \mathcal{L}$ for each region $\mathcal{P}_i^\star$ and we have $\forall w_j \in W_{\mathcal{P}_i^\star} : a_{w_j} \in \mathcal{P}_i^\star$. The number of waypoint entries to store for each region is a protocol parameter $\mathcal{L}$. Each node may store up to $\mathcal{L}$ best waypoints with maximal H_w values and it discards other potential waypoint entries. Only one entry per a_w may exist in the node routing table. A packet forwarded more than once by a node can only generate a single entry at this node the first time it crosses the node, when its h_c value is small and thus H_w is large.

3.7 Details of Packet Forwarding

To forward a packet, a node inserts the address of the best waypoint routing entry into the packet header and sends it to the next hop defined in the partial route r_w. If there is no waypoint for a destination, the node uses greedy routing.

Fig. 6 presents the following example. Assume that node a_i receives a packet whose waypoint field is empty and final destination is a_d. Waypoint entry w_k is selected from $W_{\mathcal{P}_i^\star} = \{w_1, \ldots, w_n\}$ found in the routing table, such that $a_d \in \mathcal{P}_i^\star$, where $\mathcal{P}_i^\star$ is unique by construction and $\forall w_j \neq w_k \in W_{\mathcal{P}_i^\star} : |a_{w_j}, a_d| \geq |a_{w_k}, a_d|$. This means that node a_i optimizes the choice of a route to a_d by selecting among its waypoints the one that is the closest to the destination.

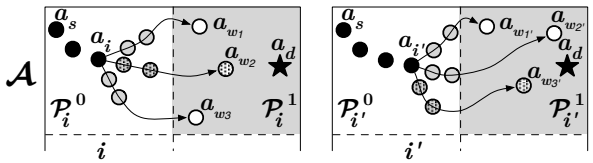


Figure 6: Packet forwarding

Node a_i inserts a_{w_k} into the packet and sends it to the next hop in r_{w_k}. The next forwarding node $a_{i'}$ may have a different set of waypoint entries and it applies the same rules with the difference that it chooses the closest waypoint to a_d belonging to its own $\mathcal{P}_{i'}^\diamond$ such that $a_d \in \mathcal{P}_{i'}^\diamond$.

To guarantee loop-free forwarding, node $a_{i'}$ applies the *waypoint optimization* principle—it replaces the waypoint in the packet with a better one if available: if $\exists w_l' \in W_{\mathcal{P}_{i'}^\diamond}$ such that $|a_{w_k}, a_d| > |a_{w_l'}, a_d|$, it inserts waypoint $a_{w_l'}$ into the packet.

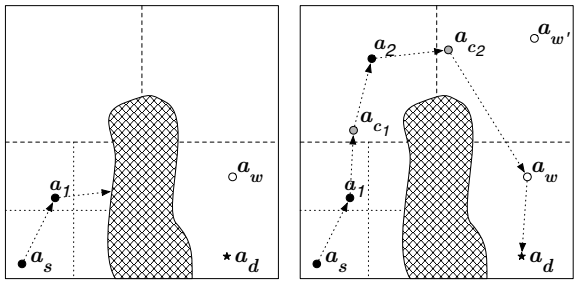
The left part of Fig. 6 illustrates the operation of node a_i on the path between source a_s to destination a_d (we assume $h_l = 2$ hops in this example). A packet sent by source a_s arrives after some hops in a_i. Let us assume that the waypoint field in the packet header is empty. Node a_i first identifies the region that contains the destination address: $a_d \in \mathcal{P}_i^1$ and the set of waypoint entries associated with $\mathcal{P}_i^1$: w_1, w_2, w_3 at locations $a_{w_1}, a_{w_2}, a_{w_3} \in P_i^1$. Node a_i can choose between three different partial routes $r_{w_1}, r_{w_2}, r_{w_3}$ towards three waypoints $a_{w_1}, a_{w_2}, a_{w_3}$, respectively. Assume that the node chooses waypoint entry w_2, because a_{w_2} is the closest to the destination: $\forall w_i \neq w_2 : |a_d, a_{w_i}| \geq |a_d, a_{w_2}|$, so it inserts waypoint entry w_2 into the packet and forwards it to $a_{i'}$, the next hop in r_{w_2}. Note that the partial route is valid up to $h_l = 2$ hops.

The right part of Fig. 6 shows what happens next at node $a_{i'}$ that has a different set of waypoint entries corresponding to the same region (note that in the example, both nodes a_i and $a_{i'}$ have the same partitioning of the address space). Node $a_{i'}$ chooses $w_{3'} = w_2$, the best one among its waypoint entries. As the partial route in the packet is still valid, i.e. $a_{i'}$ is in the previously selected partial route, $a_{i'}$ can extend the route by replacing the waypoint entry in the packet with its best waypoint $w_{3'}$. In a similar way as previously, it forwards the packet to the next hop defined by this waypoint entry. Greater h_l means that the protocol keeps larger traces, but because of that, a forwarding node can use its own waypoint entries and waypoint entries stored at $h_l - 1$ predecessors, which is good, because a forwarding node has sufficient information to continue forwarding along a certain trajectory or change to a new, more efficient one.

We can observe that even if each node only knows some partial information about paths—partial routes to known waypoints, successive nodes construct the whole path between a source and a destination. Also note that each node keeps the waypoint entries having the largest value of H_w

metric. As the whole path $(a_s, \ldots a_d)$ is a concatenation of small pieces (partial routes), the waypoint metric computed at the destination a_d : $H_w = |a_s, a_d|/h$ is also large, which means that the resulting path is close to the shortest one and the protocol constructs it without the need of any global information, a graph structure, or a graph optimization algorithm.

3.8 Checkpoint Creation



(a) Without checkpoints (b) With checkpoints

Figure 7: Waypoint forwarding

Loop-freeness of the forwarding scheme (cf. Sec. 3.7 and 4) imposes strict conditions on the update of partial routes and raises a problem of inefficient packet forwarding for small h_l. The reason is twofold. First, each node only records a small number waypoints in comparison to the large number of nodes in large regions. Second, every forwarding node optimizes the packet route by using its closest waypoint towards the destination. Subsequent forwarding nodes do not necessarily store the same waypoint (too many candidates) causing path extension impossible. The situation is illustrated in an example in Fig. 7a. Node a_s sends a packet to a_d using a_w as its waypoint. However, at a_1 the partial route is finished and subsequent forwarding nodes have to use greedy forwarding to advance towards a_w, which may lead to a drop at an obstacle. The situation results in a dramatic performance loss that we evaluate later on (cf. Fig. 15, 17).

To solve the problem, we have observed that in typical topologies, the number of nodes at the region edge is small compared to the region interior. The idea is therefore to group waypoints on a forwarding node with respect to *checkpoints* (a sort of "bread crumbs" or region entry points) residing at the region edge. If on the forwarding node, the packet partial route ends, we extend it by *borrowing* the partial route belonging to another known waypoint on the node sharing the regional checkpoint with the packet waypoint. Let us consider the example in Fig. 7b. This time, the packet contains a regional waypoint checkpoint, so instead of falling back to greedy forwarding when the partial route expires, the forwarding node sends the packet to a_{c1}. If a_1 has a partial route to a_{c1} then uses it, otherwise runs greedy routing to get there. In both cases, the packet advances in the right direction, but the information on nodes still scales as $O(\log N)$, because a checkpoint is just a *label* attached to a waypoint in the routing table.

Note that due to quadtree partitioning, all nodes residing together within a certain region share the routing table organization for external regions. Using this observation, we have discovered a local procedure to compute the checkpoint on a forwarding node for the source of an incoming packet (becoming a waypoint) associated with its corresponding region. For this purpose, a packet has a source checkpoint field (initially set to the source node a_s) being part of the routing header. Note that each packet contains both source checkpoint used for learning and checkpoint field used for forwarding to *borrow* partial routes from other waypoints. The forwarding node finds the corresponding region of the current checkpoint $a_c \in \mathcal{P}^{(c)}$ and the last hop last hop $\in \mathcal{P}^{(lh)}$ according to the local routing tables. If $\text{size}(\mathcal{P}^{(c)}) = \text{size}(\mathcal{P}^{(lh)})$, the node updates the packet checkpoint with the last hop. The packet now contains a candidate waypoint (a_s) with the associated checkpoint for region $\mathcal{P}^{(c)}$. Due to this procedure, only nodes at the borders of ever growing (or equally sized) regions update checkpoints. Other nodes share the same global partitioning at a large scale and do not modify previously established checkpoints.

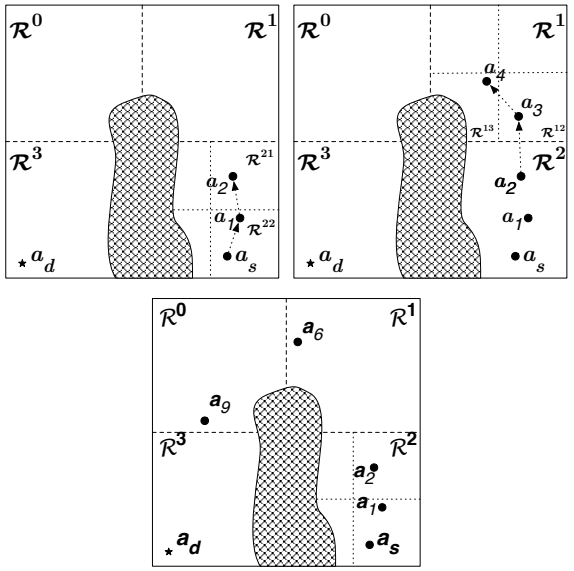


Figure 8: Learning checkpoints

Fig. 8 presents an example process of learning checkpoints and storing them in routing tables. First, a_s sends a packet to a_d. At the beginning, the source checkpoint field in the routing header is set to a_s. a_1 does not update this field as the packet does not cross any region. After receiving the packet, a_2 sets a_s as the waypoint for region $\mathcal{R}^{22}$ (in this example, we denote regions with $\mathcal{R}$ and keep the same numbering scheme for all nodes for simplicity reasons). As the previous hop lies in another region, a_2 sets a_1 as the checkpoint. Then, between a_2 and a_3, the packet crosses larger regions $\mathcal{R}^2$ and $\mathcal{R}^1$ so the source checkpoint field is set to a_2, which is valid for every node in $\mathcal{R}^1$. Note that when transmitting the packet between a_3 and a_4, nodes do not update the source checkpoint field, because the crossed regions are

smaller than the ones already crossed. Nodes then update the source checkpoint field only when crossing the border between $\mathcal{R}^0$ and $\mathcal{R}^1$ as well as between $\mathcal{R}^3$ and $\mathcal{R}^0$. The last part of Fig. 8 shows all chosen checkpoints: a_9 for nodes in $\mathcal{R}^3$, a_6 for nodes in $\mathcal{R}^0$, a_2 for nodes in $\mathcal{R}^1$, and a_1 for nodes in $\mathcal{R}^{21}$.

Nodes use checkpoints as targets in greedy routing or to extend partial routes to waypoints. Fig. 7 explains how nodes use checkpoints in forwarding. In this example, a_s sends a packet to a_d so it includes waypoint a_w, checkpoint a_{c1}, and partial route r_{aw} in the packet header and sends it to the first node in r_{aw}. When the packet reaches a_1, the node runs out of the partial route. We assume that a_1 does not have any information in its routing table to forward the packet to the waypoint so it uses greedy routing towards a_{c1} instead of a_w. When the packet arrives in a_{c1}, a_{c1} clears the checkpoint field in the header, updates the header with a new partial route, and sets the checkpoint field to a_{c2}. Upon arriving in a_2, the node runs out of the partial route, but this time a_2 has $a_{w'}$ in its routing table. As waypoint a_w in the packet and waypoint $a_{w'}$ in the routing table have the same checkpoint a_{c2}, the node inserts the partial route to a_{c2} from the routing table into the header and forwards the packet. After reaching a_{c2}, the packet continues its way to waypoint a_w and finally to the destination.

3.9 Path Exploration and Backtracking

When a packet reaches a concave node that does not have any waypoint to use for forwarding, it uses *path exploration* to find a potential route. In path exploration, a node forwards a packet tagged as *exploring* to a node that is not closer to the destination, but is the farthest from the previous hop. Such forwarding is possible only if the node sending the packet is still in the packet trace (in our simulations it means 3 or 5 hops). A node removes the *exploring* tag from a packet, if it is closer to the destination than the tagging node. It finds a node with the same *waypoint*, as the one in the packet, but with lower hop count, or it finds a *waypoint* closer to the destination than the one in the packet. When a node removes the tag, the packet continues its way based on the *waypoint* mechanism.

When none of these conditions are fulfilled, a forwarding node uses *backtracking* to explore other potential routes: it sends the packet backwards (to the previous node in the packet trace) tagging it as *reverse*. Upon receiving a reverse packet, a node repeats the selection process of the next hop by avoiding the node chosen previously as the next hop. Nodes can send packets backwards until there are no more nodes in the trace. If a node receives a reverse packet with a waypoint or a checkpoint from its routing table, it considers it as invalid and drops it.

Although both mechanisms are quite simple, they complement the main mechanism based on waypoints and checkpoints, which leads to achieving very good results presented in Section 5. The system of checkpoints provides global leads on paths, while path exploration and backtracking allows to deal with small obstacles and network dynamics, closing the gap between partial paths.

3.10 Refreshing Routing Information

Finally, we address the issue of dynamic adaptation to changing topology. In a large-scale network, to deal with a substantial part of nodes that may join and leave the network, we use route ageing. Each routing entry has an associated timestamp that a node takes into account in the choice of the suitable partial route: the node may prefer slightly longer, but more reliable partial routes to the partial routes not refreshed for a long time. Such a refreshing mechanism is sufficient in our case to deal with the network dynamics, because a node only stores the information on short partial routes (3 or 5 next hops) that indicate the direction of the complete route, so even if some nodes leave or join the network, the routing entries remain valid. Nodes close to the change in the network will learn about the modification through the backtracking mechanism.

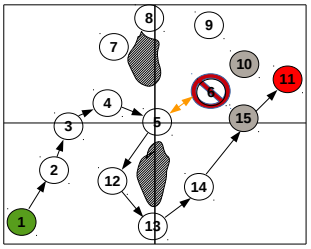


Figure 9: Backtracking and waypoint refreshment.

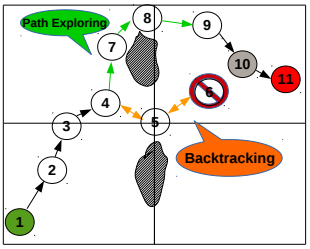


Figure 10: Backtracking and path exploration

Figure 9 presents this mechanism. Node 1 sends a packet to 11 using 10 as its *waypoint*. However, Node 6 goes down. Nodes use the *backtracking* mechanism so the packet is transferred back to Node 5 that deletes waypoint 10 from its routing table and chooses another one from the same subspace (Node 15). Other Nodes $1-4$ will soon replace Node 10 in their routing tables due to the ageing process. In Fig. 10, Nodes 4 and 5 do not have any other *waypoint* or *neighbor* closer to the destination so Node 4 invokes *path exploration* to bypass the obstacle and deliver the packet. Note that such mechanisms are much more efficient than dropping a packet because reconstructing the routing structure and resending the packet again significantly decrease the delay.

It is possible to replace both path exploration and backtracking by one of the *face routing* protocols (GPSR, GOAFR+, CLDP, GDSTR, GDSTR-3D). Such a solution does not influence WEAVE performance (as it is used only in 1-2% cases) and guarantees packet delivery. However, GPSR, GOAFR, GOAFR+ require the Planar Graph assumption, while CLDP (only 2D), GDSTR, GDSTR-3D result in huge protocol overhead for removing crossed edges (CLDP) or maintaining a global convex hull tree (GDSTR, GDSTR-3D). Currently, we implement WEAVE with path exploration and backtracking as it achieves a high packet delivery rate (cf. Sec. 5). Note also that the combination of face routing and WEAVE will increase overhead as WEAVE consumes some space in packet headers (cf. Sec. 3.2), while CLDP, GDSTR, GDSTR-3D send signaling messages to provide the information on the global topology.

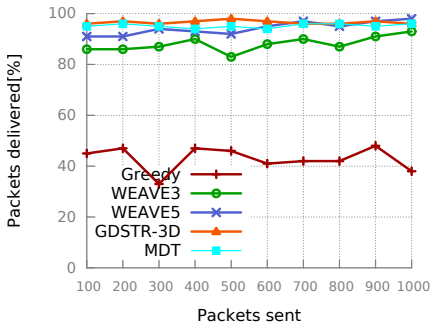


Figure 11: Packet delivery during the learning phase, Senslab

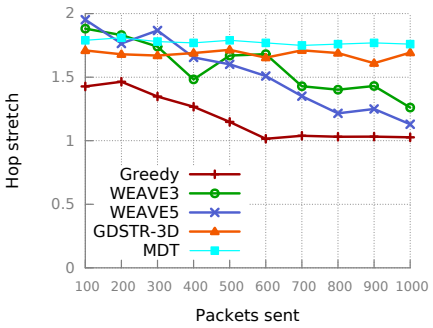


Figure 12: Hop stretch during the learning phase, Senslab

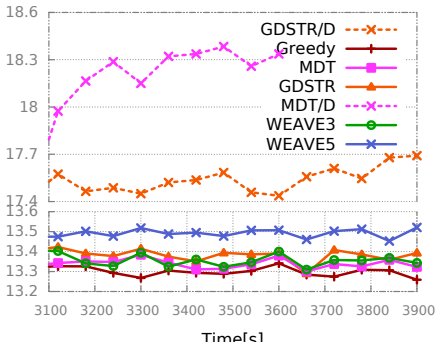


Figure 13: Energy consumption in time, Senslab

4 Loop-freeness

For simplicity reasons, we omit the concept of checkpoints, which does not change the main conclusions of our observations.

THEOREM 1. Loop free property for unbounded traces.

In the $h_l = \infty$ case (packet traces are unbounded), the protocol is loop free, so it always uses finite routes.

PROOF. The number of possible waypoints in a network is finite, because any node may be considered as a waypoint and we assume a finite number of nodes. When a node selects a waypoint, all waypoints placed equally distant or farther from the packet destination will not be used. In the case of unbounded traces, if a node sends a packet to a waypoint, it will eventually reach it by using the inverse trace towards the waypoint. Intermediate forwarding nodes may apply waypoint optimization and each replacement of a waypoint by a better one to reduce the number of still valid waypoints by at least 1. The number of possible waypoint replacements is also finite. This contradicts the condition for obtaining loops and infinite routes: an infinite number of waypoint replacements is necessary to create an infinite route from partial routes of finite length. □

THEOREM 2. *In the $h_l < \infty$ case, the protocol is also loop free and it provides finite routes.*

PROOF. Let us assume that there is a loop, so there are three possibilities. First, the waypoint is regularly replaced with a better one closer to the destination, but then the same argument as above applies: in this case, the number of legitimate waypoints is decreasing, which is in contradiction with the presence of a loop. Second, the path to the current waypoint is extended on the way by a forwarding node. Due to the fact that forwarding node can only extend the route if and only if the current h_c to the waypoint is lower than the waypoint h_c in the packet, the path cannot be extended indefinitely. Third, if the path extension to the current waypoint does not exist nor a closer waypoint was found, the procedure switches back to greedy forwarding, which does not result in loops. □

5 Evaluation

We have chosen greedy routing, MDT, and GDSTR-3D as reference protocols, because previous evaluations already showed their good performance in comparison with other proposed protocols for geographic routing in 3D networks such as CLDP/GPSR, GDSTR, AODV, VRR [6], and S4 [21]. To make our comparisons fair, we use single hop greedy routing for all geographic protocols. We configured GDSTR-3D to use two 2D hulls to approximate a 3D hull (2x2D). We use MDT for both 2D and 3D networks. We evaluate two variants of WEAVE: with the size of the partial routes $h_l = 3$ (WEAVE3) and $h_l = 5$ (WEAVE5). In parts of our evaluations, we show the impact of checkpoints and evaluate a version without them (Waypoint3 and Waypoint5). In sec. 5.8, we compare our solution against RPL [26] to show the benefits of using geographic routing.

5.1 Experiments on a Testbed

To validate the performance of WEAVE in real world conditions, we have run experiments on the Senslab testbed [5] with 256 WSN430 nodes placed in a 3D grid. The testbed supports both operating systems used in our evaluations (TinyOS and Contiki) and the code required for different protocols (GDSTR-3D on TinyOS and Contiki for other protocols). We have used low transmission power to create a topology with multiple hops. For each test, we have performed at least 10 000 transmissions between a random source and destination pair with 50B UDP packets.

Figs. 11 and 12 show the packet delivery rate and the hop stretch during the learning phase. All protocols experience some packet loss caused by unreliable radio communication. WEAVE achieves very similar delivery rate and a significantly lower hop stretch than other protocols. After the learning phase, nodes send one 50B packet every 15s to measure the energy consumption. We have measured the energy consumption of GDSTR-3D and MDT also during the update of the topology (denoted as GDSTR/D and MDT/D respectively). WEAVE3, MDT and GDSTR-3D have similar header sizes (8B/4B difference), so energy consumption for transmissions is almost the same. Increasing the trace size to 5 in the WEAVE5 variant, increases the header size and thus energy consumption, but less than 1%. During topology modifications, GDSTR-3D consumes 30% more energy to send updates to every neighbor in the spanning tree. MDT requires even more control traffic to discover all DT neighbors.

5.2 Simulations

To evaluate WEAVE for a larger parameter space, we have run simulations using the following tools:

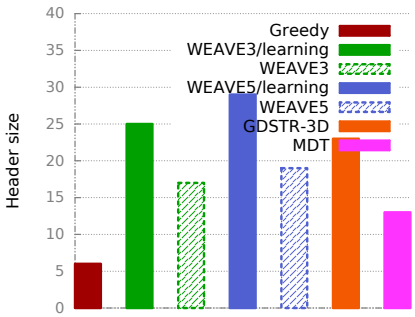


Figure 14: Header size of tested protocols.

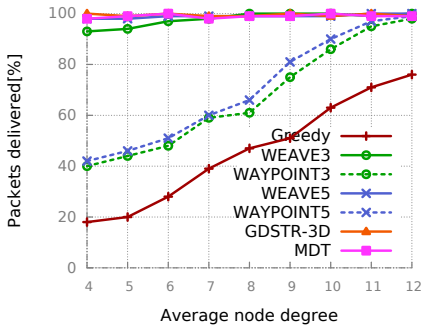


Figure 15: Packet delivery rate, network with 800 nodes.

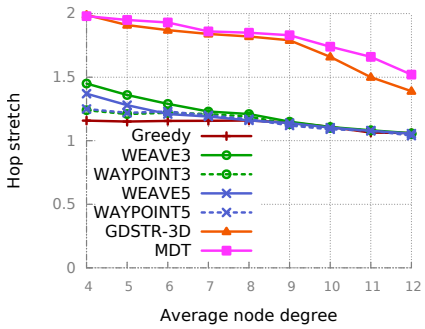


Figure 16: Hop stretch, network with 800 nodes

ns-3: greedy routing, GDSTR-3D, MDT and WEAVE for large-scale networks ($>$ 1000 nodes).
Cooja(v.2.6): greedy routing, MDT and WEAVE for small networks ($\leq$ 1000 nodes). We have used Sky Motes as the execution platform with CC2420 2.4 GHz radio and ContikiMAC at Layer 2.
TOSSIM(v.2.1): GDSTR-3D for small networks ($\leq$ 1000 nodes), Micaz Motes with an ideal radio channel as the execution platform. The source code comes from the authors [27].

There are multiple reasons for using three different simulators. First, ns-3 uses a simplified representation of lower layers, so we can test the behavior of the protocols in large-scale networks. Second, through Cooja and TOSSIM, we study a real protocol stack implementation executed in a controlled simulated environment, but the number of simulated nodes is highly limited. As GDSTR-3D is implemented on TinyOS, TOSSIM is required to run the code. Other protocols were implemented in Cooja under Contiki. We argue, however, that the performance of a routing protocol is only marginally affected by the type of the operating system and lower layer protocols.

Unless stated differently, we have used the same packet loss rate as experienced during test on Senslab Testbed (1%) in all our simulations. For each set of parameters, we have randomly generated at least 20 topologies, performed at least 10 000 transmissions between random pairs of nodes and averaged the results. The hop stretch is only computed for packets reaching the destination.

5.3 Initial Simulation Comparisons

Figure 14 presents a comparison between data packet header sizes of tested protocols. We assume 3D coordinates (x, y, z) for the packet source and destination. WEAVE 3 and WEAVE 5 use 25B and 29B header respectively. The WEAVE header is only a few bytes larger than the GDSTR-3D header, while in the forwarding only version, it is even smaller. MDT has a significantly smaller header size than other protocols. However, the WEAVE header is the only overhead introduced by the protocol, while all other protocols (except greedy routing) need a significant amount of control traffic to fill and maintain routing tables.

We evaluate the packet delivery rate in a network with 800 nodes for different network densities (cf. Fig. 15) in the stable state after the learning phase. For each network configuration, we have generated at least 10 random networks. As expected, both GDSTR-3D and MDT achieve 97-99% for all tested networks. WEAVE achieves 95% delivery rate for low density networks and almost 100% for networks with a higher average node degree. The versions without checkpoints perform significantly worse, especially in sparse networks. Note that in WEAVE, the routing tables are constantly being updated. If a route is not found, it does not mean that there is no connectivity between two nodes. Resending the same packet, after a short period of time, usually results in successful delivery. During our simulations, we did not observe a pair of nodes without connectivity.

Fig. 16 presents the hop stretch (the ratio between the length of a route for a given protocol and the shortest path) in the same configuration. For low density networks, both MDT and GDSTR-3D perform almost twice worse than the shortest path. By default, GDSTR-3D performs greedy routing and tries to recover using a spanning tree so the protocol may go into a local minimum and then look for another route, which increases the hop stretch. MDT uses its virtual links to connect a DT neighbor, which creates routes far from optimal, especially for sparse networks.

After the learning phase, WEAVE directly uses routes close to the shortest ones trying to avoid local minima. Removing checkpoints slightly reduces the hop stretch, as only packets with shorter paths are successfully delivered. Greedy routing has an almost constant hop stretch.

Next, we evaluate the packet delivery rate for constant network density (average node degree of 7) for different network sizes (cf. Fig. 17). GDSTR-3D and MDT maintain 98% delivery rate while WEAVE3 slightly degrades to 92% in the largest networks and WEAVE5 keeps its delivery rate at 97%. The version without checkpoints, once again, achieves much lower delivery rate, as partial routes are not enough to deliver all packets, especially in larger networks. With longer routes, the efficiency of greedy routing drops to 20% for the largest networks. In further experiments, for better readability, we present only WEAVE results for the version with checkpoints, as they always perform significantly better.

In large networks, we can observe an important increase of the hop stretch for GDSTR-3D (cf. Fig. 18), because the protocol enters local minima more often. The root of the spanning tree is also farther away, so the recovery process

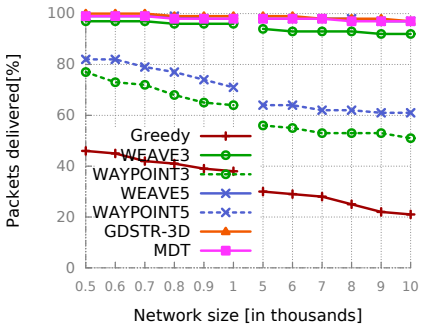


Figure 17: Packet delivery rate for various network size.

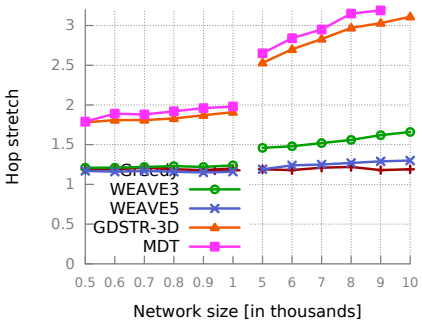


Figure 18: Hop stretch for various network size.

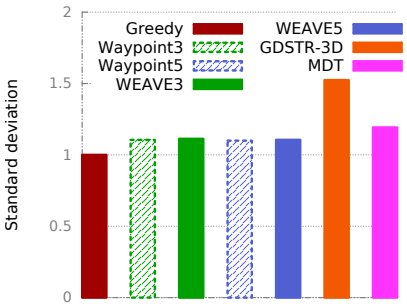


Figure 19: Standard deviation of number of packets forwarded by each node

takes more time. MDT uses longer virtual links more often, which also increases the hop stretch. Both versions of WEAVE obtain much lower hop stretch growth that does not exceed 1.7 (WEAVE3) and 1.4 (WEAVE5). Greedy routing results in almost constant hop stretch for all tested networks.

To test the distribution of energy consumption over nodes, we have measured the number of packets forwarded by each node (cf. Fig. 19). In WEAVE, each node chooses its waypoints independently, so the distribution is balanced. Moreover, in most cases, waypoints are not reached by packets. Intermediary nodes keep changing waypoints for better ones to forward a packet to its final destination. Moreover, checkpoints do not tend to attract more traffic than ordinary nodes. In MDT, the end of virtual links and nodes near obstacles forward much more packets than the others. GDSTR-3D nodes placed near the tree root also receive significantly more control and data packets, which can reduce their lifetime.

5.4 Learning Phase

In this section, we evaluate WEAVE during the learning phase: Fig. 20 presents the delivery rate for the first 1000 packets exchanged in the network (800 nodes, average node degree of 6). At the beginning, the routing tables for WEAVE are empty and the both versions of WEAVE obtain more than 90% delivery rate. The result comes from the path exploration and backtracking mechanisms. Nevertheless, their drawback is an increased hop stretch during the initial phase when exchanging the first few hundred packets (cf. Fig. 21). Nevertheless, the hop stretch for both WEAVE versions decreases rapidly while the performance of GDSTR-3D and MDT remains at the same level (1.8).

5.5 Dynamic Networks

To evaluate the performance in dynamic networks with node churn, we switch off a given amount of random nodes (off nodes) and specify the change frequency—50% change frequency means that for every packet forwarded in the network, there is a 50% probability to turn off one of the working nodes and turn on one of the nodes that were shutdown.

Fig. 22 presents the packet delivery rate for different change frequencies. In this scenario, the performance of GDSTR-3D significantly decreases. Every time a node is turned on or off, the protocol needs to rebuild its spanning tree. If a forwarded packet happens to be in the part of the tree being rebuilt, the packet is dropped to avoid loops. The same thing happens for MDT: the protocol needs to maintain connections between DT neighbors and cannot keep the communication if some of intermediary nodes are down. WEAVE does not maintain complete routes so it can deal even with frequent node churn, which results in an almost constant packet delivery rate.

Fig. 23 presents the results for a fixed amount of nodes turned off, fixed frequency, and different network sizes. Even for a constant frequency, the performance of GDSTR-3D decreases with the network size. Topology changes, especially near the tree root, affect a larger part of the network, which causes more packet losses. MDT virtual links get longer and easier to break by a shutdown of a random node. As in the previous scenario, the results for both versions of WEAVE remain almost unaffected by the network size.

All protocols need to use a hello message mechanism to discover direct neighbors. Usually, the time interval between sending those messages needs to be carefully adjusted. Sending too many of them increases the protocol overhead, while sending too few, delays the protocol reaction to topology changes. However, while it is crucial for GDSTR-3D and MDT to maintain a valid spanning tree/virtual links, WEAVE can just update its neighbor table when a node does not succeed to send a packet, thus, it is not affected by the hello timer interval.

Fig. 24 illustrates this phenomenon: for given network dynamics parameters (10% nodes off and 20% frequency), the performance of GDSTR-3D and MDT decreases for the increasing hello interval. WEAVE remains unaffected by the interval, so nodes can choose large hello intervals to reduce energy consumption. Note also that with each topology change, both MDT and GDSTR-3D generate a significant amount of the control traffic, while WEAVE does not exchange any control messages.

5.6 Concave Obstacles

To push routing to the limits, we test several 2D and 3D networks with carefully placed nodes and large concave obstacles in the middle of the topology. We have tried to introduce a large amount of local minima, so that greedy routing almost always fails and all protocols need to use their mechanisms to recover and deliver packets. We observe sending packets between any two pairs chosen at random. By default, GDSTR-3D uses greedy routing that forwards packets

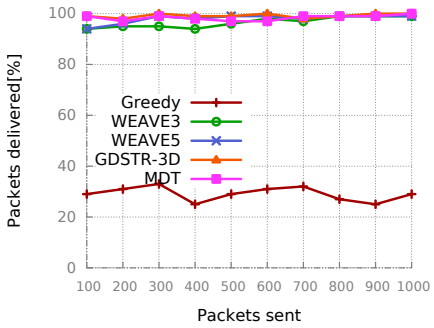


Figure 20: Packet delivery rate upon the learning phase.

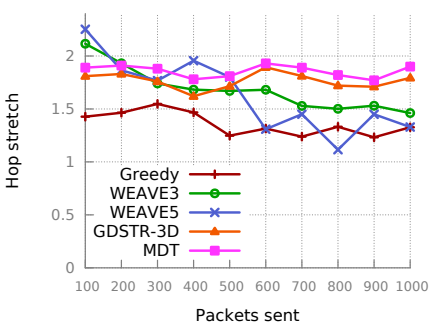


Figure 21: Hop stretch during the learning phase.

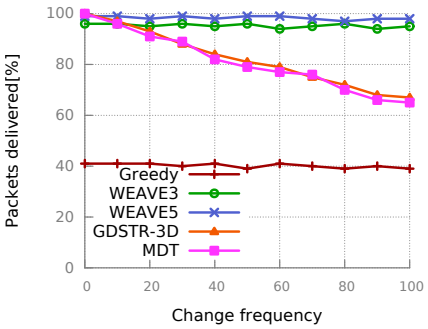


Figure 22: Packet delivery rate with 10% nodes off.

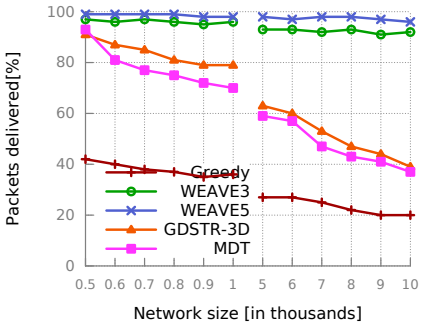


Figure 23: Packet delivery rate with 10% nodes off and 50% dynamic.

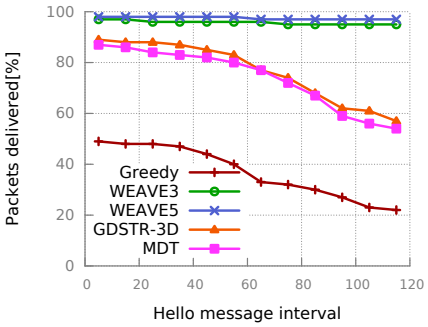


Figure 24: Hello interval impact on packet delivery

Figure 25: Partial map of a city used in experiments.

Table 1: Summary of results for networks with obstacles.

Aspect	Greedy	WEAVE3	WEAVE5	MDT	GDSTR-3D
Delivery rate	**46%**	**98%**	**99%**	**99%**	**99%**
Hop stretch	**1.0**	**1.06**	**1.02**	**1.6**	**1.7**

toward local minima and then tries to recover using a spanning tree. It results in longer paths as shown in Fig. 26 for a chosen source and destination pair. MDT also performs greedy forwarding between DT neighbors, which can result in non optimal detours (cf. Fig. 27). On the other hand, our protocol uses waypoints with the lowest metric, which creates almost optimal paths (cf. Fig. 28). Table 1 summarizes the results for the scenario. WEAVE delivers almost 100% of packets while having a much lower hop stretch than GDSTR-3D. When packets under greedy routing arrive at the destination, they use the optimal route, so the hop stretch is 1.

5.7 Realistic Geographic Topology

We have generated a 2D topology based on a map of Grenoble by placing a node in all buildings and adjusting the distances between them to obtain a fully connected graph (cf. Fig. 25). The resulting network contains 18144 nodes with the average node degree of 5. Table 2 presents the results for all protocols. Even for such a large-scale network, WEAVE achieves a high delivery rate while maintaining very low hop stretch. We have repeated our tests with some network dynamics (5% nodes off, 50% frequency), which significantly decreases the GDSTR-3D and MDT performance, while leaving the results of WEAVE almost unaffected. WEAVE also requires less memory and does not use any control messages.

Table 2: Summary of results for the city network.

Aspect	Greedy	WEAVE3	WEAVE5	MDT	GDSTR-3D
Delivery rate	**36%**	**91%**	**96%**	**98%**	**98%**
Delivery rate(dynamic)	**36%**	**94%**	**98%**	**80%**	**83%**
Packet stretch	**1.19**	**1.7**	**1.5**	**1.8**	**2.4**
Packet stretch(dynamic	**1.19**	**1.74**	**1.6**	**3.2**	**3.5**
Overhead(per node)	**0B**	**0B**	**0B**	**1850B**	**1600B**
Memory used(per node)	**0B**	**800B**	**1060B**	**980B**	**1400B**

Table 3: Memory usage of routing tables for different network sizes.

Size(nodes)	WEAVE3	WEAVE5	MDT	GDSTR-3D	RPL
800	**83B**	**99B**	**96B**	**112B**	**25600B**
5000	**178B**	**202B**	**189B**	**240B**	**160000B**
18144	**800B**	**1060B**	**980B**	**1400B**	**580608B**

5.8 Comparison with Standard Routing

Compared to classical routing protocols, geo-routing requires node locations, which may introduce some additional overhead, like retrieving or computing coordinates. However, a standard routing protocol such as RPL uses huge amounts of memory to store routing tables when address aggregation is infeasible. Table 3 presents memory usage for different network sizes. Even for relatively small networks (800 nodes), RPL requires more than 25kB of storage per node for the routing table. For our biggest tested topology, WEAVE uses more than 700 times less memory.

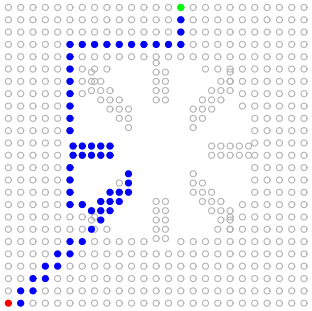


Figure 26: Concave obstacle - GDSTR-3D

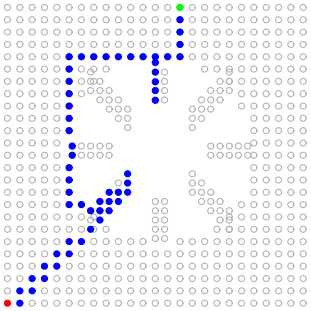


Figure 27: Concave obstacle - MDT

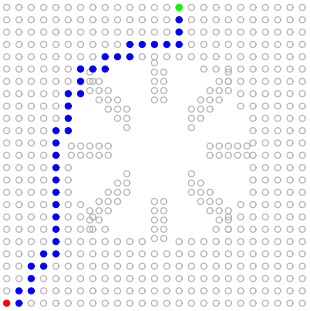


Figure 28: Concave obstacle - WEAVE

6 Conclusion

We have presented WEAVE, a geographical routing protocol for large-scale dynamic multi-hop wireless networks. Our protocol does not use any control traffic and fills up routing tables only by observing incoming traffic. Instead of maintaining the information on whole routes, WEAVE constructs them out of partial routes to waypoints. The key element of WEAVE is a system of checkpoints used as "bread crumbs".

We have compared WEAVE against greedy routing, MDT [17], and GDSTR-3D [27] through measurements on a sensor network testbed and simulations for various network sizes. Our results show that WEAVE achieves a high packet delivery rate, low stretch, and balanced energy consumption.

Acknowledgements

This work was partially supported by the French Ministry of Research project IRIS under contract ANR-11-INFR-016 and DataTweet under contract ANR-13-INFR-0008-01.

7 References

[1] ARAD, N., AND SHAVITT, Y. Minimizing Recovery State in Geographic Ad Hoc Routing. *IEEE Transactions on Mobile Computing 8*, 2 (Feb 2009), 203–217.

[2] BLAZEVIC, L., LE BOUDEC, J.-Y., AND GIORDANO, S. A Location-Based Routing Method for Mobile Ad Hoc Networks. *IEEE Transactions on Mobile Computing 4*, 2 (2005), 97–110.

[3] BOSE, P., MORIN, P., STOJMENOVIC, I., AND URRUTIA., J. Routing with Guaranteed Delivery in Ad Hoc Wireless Networks. In *DIAL'M* (Seattle, USA, 1999), pp. 48–55.

[4] BOSE, P., MORIN, P., STOJMENOVIC, I., AND URRUTIA, J. Routing with Guaranteed Delivery in Ad Hoc Wireless Networks. *Wireless Networks 7*, 6 (November 2001), 609–616.

[5] BURIN DES ROSIERS ET AL., C. SensLAB: Very Large Scale Open Wireless Sensor Network Testbed. In *Proc. 7th TridentCOM Conference* (Shanghai, Chine, Apr. 2011).

[6] CAESAR, M., CASTRO, M., AND NIGHTINGALE, E. B. Virtual Ring Routing: Network Routing Inspired by DHTs. In *Proc. of ACM SIGCOMM* (2006), pp. 351–362.

[7] CLARK, B. N., COLBURN, C. J., AND JOHNSON, D. S. Unit disks graphs. *Discrete Mathematics 86* (December 1990), 165–177.

[8] DE COUTO, D. S. J., AGUAYO, D., BICKET, J., AND MORRIS, R. A High-Throughput Path Metric for Multi-Hop Wireless Routing. In *Proc. of MOBICOM* (New York, NY, USA, 2003), pp. 134–146.

[9] DE COUTO, D. S. J., AND MORRIS, R. Location Proxies and Intermediate Node Forwarding for Practical Geographic Forwarding. Tech. Rep. MIT-LCS-TR-824, MIT Laboratory for Computer Science, June 2001.

[10] FINN, G. G. Routing and Addressing Problems in Large Metropolitan-Scale Internetworks. Tech. Rep. ISI/RR-87-180, Information Sciences Institute, Mars 1987.

[11] GABRIEL, K., AND SOKAL, R. A New Statistical Approach to Geographic Variation Analysis. *Systematic Zoology 18* (1969), 259–278.

[12] KARP, B., AND KUNG, H. GPSR: Greedy perimeter stateless routing for wireless networks. In *Proc. of MOBICOM* (Boston, USA, August 2000), ACM, pp. 243–254.

[13] KIM, Y., GOVINDAN, R., KARP, B., AND SHENKER, S. Geographic Routing Made Practical. In *Proc. of NSDI* (2005), pp. 112–124.

[14] KIM, Y.-J., GOVINDAN, R., KARP, B., AND SHENKER, S. Lazy Cross-Link Removal for Geographic Routing. In *Proc. of SenSys* (2006), pp. 112–124.

[15] KUHN, F., WATTENHOFER, R., ZHANG, Y., AND ZOLLINGER, A. Geometric Ad-Hoc Routing: of Theory and Practice. In *Proc. of ACM PODC* (2003).

[16] KUHN, F., WATTENHOFER, R., AND ZOLLINGER, A. Worst-Case Optimal and Average-Case Efficient Geometric Ad-Hoc Routing. In *Proc. of ACM MOBIHOC, Annapolis, Maryland, USA* (June 2003).

[17] LAM, S. S., AND QIAN, C. Geographic routing in d-dimensional spaces with guaranteed delivery and low stretch. In *Proc. of ACM SIGMETRICS* (2011), ACM, pp. 257–268.

[18] LEONG, B., LISKOV, B., AND MORRIS, R. Geographic Routing Without Planarization. In *Proc. of the USENIX NSDI Conference* (2006).

[19] LEONG, B., MITRA, S., AND LISKOV, B. Path Vector Face Routing: Geographic Routing with Local Face Information. In *Proc. of ICNP* (Boston, Massachusetts, November 2005).

[20] LIM, M., CHESTERFIELD, A., CROWCROFT, J., AND CHESTERFIELD, J. Landmark Guided Forwarding. In *Proc. of ICNP* (2005), pp. 169–178.

[21] MAO, Y., WANG, F., QIU, L., LAM, S. S., AND SMITH, J. M. S4: Small State and Small Stretch Routing Protocol for Large Wireless Sensor Networks. In *Proc. of the USENIX NSDI Conference* (2007).

[22] SCHILLER, E., STARZETZ, P., ROUSSEAU, F., AND DUDA, A. Binary Waypoint Geographical Routing in Wireless Mesh Networks. In *Proc. of ACM MSWiM* (2008).

[23] SEADA, K., ZUNIGA, M., HELMY, A., AND KRISHNAMACHARI, B. Energy-Efficient Forwarding Strategies for Geographic Routing in Lossy Wireless Sensor Networks. In *Proc. of SenSys* (2004), pp. 108–121.

[24] STOJMENOVIC, I. Position-Based Routing in Ad Hoc Networks. *IEEE Communications Magazine 40*, 7 (Jul 2002), 128–134.

[25] TAKAGI, H., AND KLEINROCK, L. Optimal Transmission Ranges for Randomly Distributed Packet Radio Terminals. *IEEE Transactions on Communications 32*, 3 (Mar 1984), 246–257.

[26] WINTER, T., THUBERT, P., CLAUSEN, T., HUI, J., KELSEY, R., LEVIS, P., PISTER, K., STRUIK, R., AND VASSEUR, J. RPL: IPv6 Routing Protocol for Low Power and Lossy Networks, RFC 6550. *IETF ROLL WG* (2012).

[27] ZHOU J., CHEN Y., L. B. Practical 3D Geographic Routing for Wireless Sensor Networks. In *Proc. of SenSys* (New York, NY, USA, 2010), pp. 337–350.