Internet of Things 9 (2020) 100152

Contents lists available at ScienceDirect

Internet of Things

journal homepage: www.elsevier.com/locate/iot

GROLL: Geographic Routing for Low Power and Lossy IoT Networks

Myounggyu Won[a], Wei Zhang[b], Chien-An Chen[c], Radu Stoleru[b,*]

[a] Department of Computer Science, University of Memphis, TN, 38152, United States
[b] Department of Computer Science and Engineering, Texas A&M University, College Station, TX, 77840, United States
[c] Palo Alto Networks, Santa Clara, CA, 95054, United States

ARTICLE INFO

Article history:
Received 5 April 2019
Revised 5 December 2019
Accepted 5 December 2019
Available online 12 December 2019

Keywords:
IoT
geographic routing
network holes
unified routing

ABSTRACT

Routing in low-power lossy networks (LLNs) remains an important research problem for Internet of Things (IoT), despite the recent standardization in RFC 6550 of RPL as an IPv6 Routing Protocol [1]. Recent deployment experiences [2] have emphasized the challenges faced by a complex design in real world deployments. One of the key challenges is versatile routing protocol design that can handle efficiently and seamlessly various combinations of routing primitives, namely unicast, multicast, and convergecast. Although RPL supports these traffic types, due to its design principle oriented around data collection, its unicast and multicast functionalities are limited. In this article, we present the design and system implementation of the first Geographic Routing framework for Low Power and Lossy Networks (GROLL). GROLL's routing engine effectively integrates routing for unicast, multicast, and convergecast and takes into account Complex Network Topologies (CNT), i.e., network holes and cuts. GROLL detects CNTs and abstracts the boundary information of detected CNTs with significantly reduced memory size. The proposed unified routing framework was implemented on real hardware and it was evaluated extensively on a testbed of 42 TelosB motes.

1. Introduction

RPL was adopted by IETF as the standard IoT routing protocol for low power and lossy networks [1]. Recent deployment experiences [2] have emphasized the complexities faced by RPL routing in real world, and pointed out some inefficiencies, especially in terms of routing path stretch. RPL is designed to support unicast, multicast and convergecast, but its design makes limiting assumptions: the predominant traffic is from nodes to the base station, followed, on rare occasions by traffic from the base station to the nodes and the traffic among nodes, while supported, is extremely rare. These assumptions may not hold well in some deployments, such as wireless sensor deployment for disaster response.

On the other hand, geographic routing is well suited for LLNs because it is simple and highly scalable; and more importantly location information of devices, sensors, and machines are often readily available in IoT applications. Furthermore, different routing primitives such as 1-1 (unicast), 1-n (multicast), and n-1 (convergecast) can be easily supported based on geographic routing. Unfortunately, to the best of our knowledge, there is no unified location-based routing framework

* Corresponding author.
E-mail addresses: mwon@memphis.edu (M. Won), wzhang@cse.tamu.edu (W. Zhang), jaychen@paloaltonetworks.com (C.-A. Chen), stoleru@cse.tamu.edu (R. Stoleru).

https://doi.org/10.1016/j.iot.2019.100152
2542-6605/

that integrates location-based routing protocols developed for different routing primitives. However, the needs for an unified routing framework has ever increased recently. In recent IoT applications [3], nodes do not just report data to the base station. For example, peer-to-peer or multicast data communications have become essential routing traffic patterns for in-network processing, data aggregation, and feedback control [3].

The main contribution of this article is the design and implementation of the first unified geographic routing framework for low power and lossy networks (GROLL) that seamlessly supports different traffic patterns. However, challenges exist in integrating geographic routing modules for different routing primitives. Complex Network Topologies (CNTs) affect the performance of location-based routing protocols: They cause arbitrarily long routing paths, unbalanced energy consumption, and increased packet loss rates [4,5]. Many researchers have proposed various location-based routing protocols designed for LLNs with CNTs for three main routing primitives: unicast [4–6], multicast [7–9], convergecast [10]. However, real-world deployments of these CNT-aware location-based routing protocols were not very successful. For example, some algorithms [11,12] are based on Unit Disk Graph (UDG) not accurately representing real network topologies. Some approaches are impacted by limited packet size [7,8]. Other protocols use a complex geometric algorithm which requires large memory space [6]. Another challenge is to make the proposed framework flexible to support various applications that require different combinations of routing traffic patterns. In addition, designing a robust and efficient (in terms of code/memory size) software design to fit into a memory-constrained sensor node is an important issue.

To address the aforementioned challenges, in this article, we present the design and implementation of the first location-based unified routing framework for low power and lossy networks. The proposed framework consists of three main components: CNT Support, Routing Engine, and Forwarder. The CNT Support detects the boundaries of CNTs under varying link qualities and abstracts the boundary information into user specified data types (e.g., polygons and convex hulls) with small memory and computational resources. The Routing Engine integrates protocols for the three routing primitives and finds efficient routing paths for varying traffic patterns. More specifically, using the information received from the CNT Support, the unicast module allows a source node to generate a routing path to a destination with small guaranteed stretch; the unicast module also uses the reachability information to see if a given destination is reachable or not, thereby minimizing unnecessary packet transmissions. The multicast module finds optimal paths (i.e., in terms of path lengths) to multicast members. The convergecast module identifies regions with concentrated network traffic and distributes the traffic, thereby increasing the network lifetime. Once the Routing Engine generates a routing path, the Forwarder is used to forward a packet to a destination node using hop-by-hop greedy geographic forwarding enabled by neighbor discovery. It also interacts with the underlying MAC layer to transmit a packet to a next-hop node.

The contributions of this paper are as follows:

- It presents the first integrated, efficient design of all important routing primitives in low power and lossy networks for IoT;
- It provides justification for engineering decisions for efficiency, made during system implementation. Our entire framework fits in the memory of a mote. In IETF circles, an unwritten rule is that “running code is standardized”;
- It presents the evaluation of the framework through a complete implementation and extensive testbed experiments.

This paper is organized as follows. In Section 2 we provide an overview of GROLL and its scenario-based operation. Before we detail the design, we briefly mention our experimental setup to help understand the operation of GROLL. Next in Section 3 we provide the details of GROLL design. Along with the descriptions of the GROLL design, experimental evaluations are presented. We give a particular focus on GROLL’s routing engine and present the details and experimental validation of the routing engine in Section 4. And then, related work focusing on an integrated routing framework is presented in Section 5, and in Section 6 we conclude this paper.

2. GROLL Overview

2.1. Software Architecture

Before we present an overview of the framework, we need to define the hole and its boundary according to [11]: A hole is a closed region bounded by a non-self-intersecting polygonal loop of network links. The polygonal loop is called the “boundary” of the hole. In particular, the nodes on the boundary are called “boundary nodes”.

The software architecture for GROLL consists of three main components: CNT Support, Routing Engine, and Packet Forwarder, as denoted by dotted (green) boxes in Figure 1. The CNT Support component has three modules: Boundary Detection, Boundary Abstraction, and Cut Detection modules. The Boundary Detection module uses the Boundary Abstraction module to abstract detected boundaries into polygons. This abstract information is used by the modules in Routing Engine as well as the Cut Detection module, thereby significantly simplifying the implementation of Cut Detection. The boundary detection/abstraction process accesses information on neighboring nodes through the Neighbor module. When the CNT detection/abstraction process is completed, the abstract CNT information is broadcast throughout the network using the Bcast module.

The Routing Engine component determines a routing path. It integrates routing protocols for three major routing paradigms: 1-1 (unicast), 1-n (multicast), and n-1 routing (convergecast). It is worth to note that although this modularized design allows for different routing protocols to be easily integrated into the proposed framework, in order to fit this

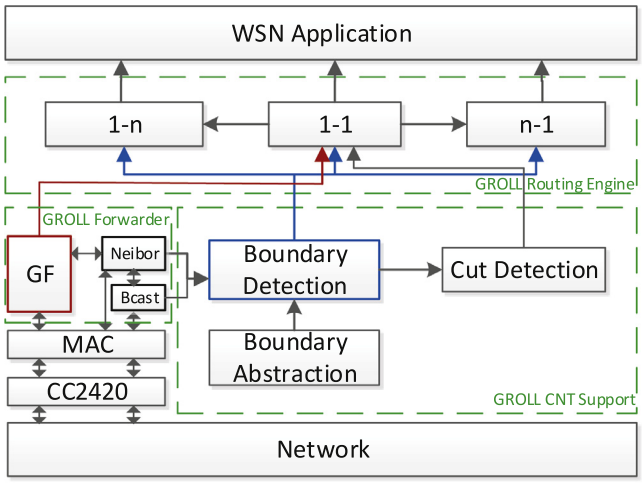

Fig. 1. Software architecture of GROLL routing framework

framework into limited code and memory space, it was designed such that some modules are highly dependent on each other. This makes the integration of other routing protocols into our framework challenging. For example, as shown in Figure 1, all three modules rely on the abstract CNT information provided by the CNT Support component. In particular, 1-1 Routing module also uses the reachability information received from the Cut Detection module; both 1-n and n-1 Routing modules reuse the functionalities of 1-1 Routing module, thereby reducing the memory size for implementation. The Packet Forwarder component includes a Neighbor Discovery module, Geographic Forwarding (GF) module, and broadcasting (Bcast) module which directly interact with an underlying MAC protocol.

2.2. Use Case Scenario

To explain how various pieces of the routing framework work together, we present a simple deployment scenario. When nodes are first deployed, nodes run the Neighbor Discovery module and select their neighbors having reliable links. The sink node also performs neighbor discovery and joins the network. Based on discovered neighbors, nodes detect CNTs in the network. As the CNT Detection module identifies boundary nodes, the CNT Abstraction module abstracts the boundary information into polygons. More specifically, the CNT Abstraction module finds the *vertex nodes* for each boundary, i.e., the nodes corresponding to the vertices of a polygon enclosing the boundary. Once the vertex nodes are identified, their locations are broadcast throughout the network. Note that we refrain from broadcasting the locations of all boundary nodes to prevent the broadcast-storm problem and also to reduce the overhead for nodes to store the locations of boundary nodes. Consequently, when the broadcast is completed, all nodes in the network have the abstract information about holes in the form of polygons. Using the abstract boundary information, the Cut Detection module can find whether a given destination is reachable or not by using a point-in-polygon algorithm. It should be noted that if the sink node knows all the locations of deployed nodes, the sink node can compute the boundary of CNTs in a centralized manner, thus not having to install the CNT Detection/Abstraction modules. However, users who are interested in autonomous operations of CNT detection/abstraction to cope with unexpectedly arising CNTs, e.g., destroyed nodes due to environmental factors or hostile users, may install the CNT Detection/Abstraction modules.

Now assume that a source node, say s, sends a packet to a destination node denoted by t. When source and destination nodes are outside the convex hulls of holes in the network, we use the convex hulls of holes to determine a path with guaranteed low stretch. The details of this 1-1 Routing module are described in Section 4.1. When either a source or a destination (or both) is inside the convex hull of a hole, a network infrastructure, called *local visibility graph*, is used for guiding a packet optimally (i.e., in terms of path length) to the destination node. We defer the details on the local visibility graph until Section 4.1.

When source node s wants to send a packet to a multicast group, say M, node s uses the 1-n Routing module. Due to the limited memory space of a node, and assuming that the sink has abundant resources, we design that the sink node manages the multicast group, i.e., nodes join/leave a multicast group by sending a short control packet to the sink node. So in our 1-n routing design, the computation of paths to multicast members is done at the sink node. Our 1-n Routing module ensures that nodes find optimal routing paths to multicast members without requiring the nodes to encode all locations of multicast members in a header. The 1-n Routing module also allows for energy-efficient recovery from packet loss and offers functionality for achieving higher energy efficiency at the cost of more storage overhead. The details of the 1-n Routing module will be discussed in Section 4.2.

One distinctive characteristic of LLNs compared to other networks is the traffic pattern called convergecast (i.e., n-1 routing) where all nodes report data to the sink. This unique traffic pattern, when there are holes in a network, creates regions with higher traffic called *hot zones*. The motivation behind the design of our n-1 Routing module is to avoid hot zones, thereby increasing the network lifetime. Thus, when source node *s* wants to send a packet to the sink, it uses the n-1 Routing module that reduces the impact of the hot zones. The details on the n-1 Routing module are discussed in Section 4.3.

2.3. Experimental Setup to Validate GROLL Design

Before we describe the details of the framework components, we present our experimental setup used for evaluating our implementation. Our testbed consists of 42 TelosB motes attached to the ceiling of an office as shown in Figure 2. The ceiling is about 2.7 meters high and the size of the office is 6 by 4.5 meters. The motes form a 7 by 6 grid network, where inter-node distance is approximately 30 cm. The motes are programmed and powered via USB cables connected to a main PC running Ubuntu 12.04 with AMD Opteron Processor 252 and 16 GBytes of RAM. The transmit power of motes is fixed to 1 (i.e., CC2420_DEF_RFPOWER is set to 1 for TinyOS). Consequently, the network has at most 7 hops (without holes). We debugged our work on the central PC by allowing motes to send debug messages through USB interfaces.

Figures 3 (a) and 3(b) depict the testbed topology with different packet delivery rate (pdr) thresholds of.7 and.9, respectively. The *pdr threhold* means the minimum allowable packet reception ratio over a given link, thus a link with smaller pdr than a threshold is not considered as a neighbor. The *pdr threshold* defines the reliability of a link: the higher the *pdr threshold*, the better reliability of a link. Determining an appropriate threshold is crucial since our boundary detection module relies on the reliability of links. For our experiments we used a pdr threshold of.8. Justifications for selecting this particular threshold are described in Section 3. We created two types of holes, convex and concave holes. The topologies with holes are shown in Figures 4(a) and 4(b) for convex and concave holes, respectively. To create holes, we artificially disabled links crossing the holes.

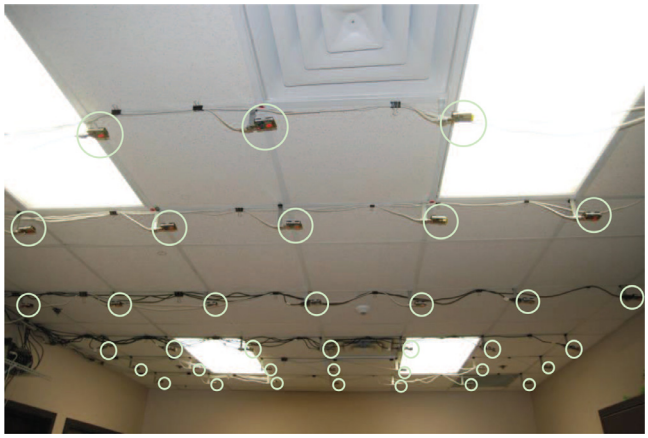

Fig. 2. A testbed with 42 Telosb motes for the performance evaluation of GROLL

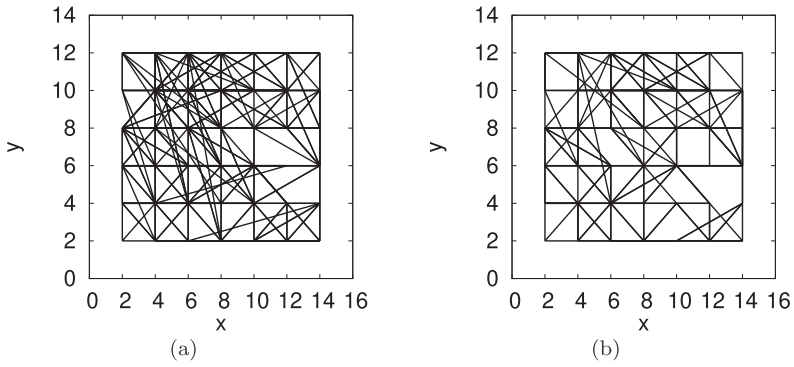

Fig. 3. The topology of the testbed with (a) pdr threshold =.7; (b) pdr threshold =.9; the unit distance equals 15 cm.

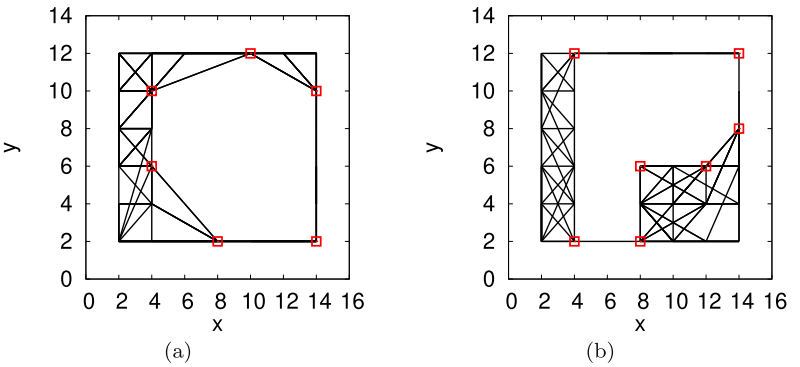

Fig. 4. The Topology of the testbed with a large "convex" hole and a large "concave" hole with pdr threshold=.8

3. GROLL Design

3.1. Boundary Detection

The Boundary Detection module is designed to detect the boundaries of network holes. Boundary detection algorithms are largely categorized into three approaches: location-based, topology-based, and statistical schemes. When node location is known, location-based boundary detection algorithms [11,12] are useful. Topology-based solutions [13–16] do not require node-location. Specifically, while these solutions have different computational and communication complexities, they share a common aspect which is to identify the boundary based only on the connectivity information. How to cope with dynamically changing topology, however, remains as a significant challenge. There are several algorithms based on statistical approach [17,18]. These solutions detect boundaries based on the number of neighbors. However, these heuristic solutions require very high node-density. In general, when node location is known, location-based boundary detection algorithms are appropriate as these algorithms are simple to implement yet very effective in detecting boundaries [15]. As such, in this paper, as node location is known, we adopt the BOUNDHOLE [11], a location-based solution. *However, as the BOUNDHOLE algorithm is designed for a theoretical domain and tested based only on simulations, to make it run smoothly on real hardware, we improve upon the original design.*

The BOUNDHOLE algorithm identifies "stuck nodes" where a packet can be stuck in a local minimum during the geographic forwarding process. Once stuck nodes are identified, one of them for each hole sends a control packet that travels along the boundary of the hole. While the control packet traverses the boundary of a hole, boundary nodes are detected. As we will show in the following section, as the control packet traverses, the boundary abstraction is also performed, i.e., the boundary is abstracted as a set of vertices of a polygon that surrounds the hole. Once the boundary detection is completed (i.e., the abstraction is also done), the locations of discovered vertices are broadcast throughout the network using the Bcast module.

However, we observe that running the BOUNDHOLE algorithm running on real hardware faces several challenges:

- The BOUNDHOLE algorithm assumes ideal links (i.e., a simple Unit Disk Graph (UDG) radio model).
- The BOUNDHOLE algorithm does not consider the link quality that dynamically changes over time.
- The BOUNDHOLE algorithm identifies even very small holes. However, small holes do not affect much the performance of location-based routing.

To address the first challenge (i.e., the UDG model), our Neighbor Discovery module takes into account the link quality in determining 1-hop neighbors and uses only the 1-hop neighbors with stable symmetric links for running the algorithm. In other words, we aim for having a closed cycle of boundary nodes that are stably connected with each other. Specifically, nodes maintain a packet delivery ratio (pdr) for each of its one-hop neighbors and chooses as its one-hop neighbor only nodes for which the pdr is greater than a predefined threshold. To choose a proper threshold for the pdr in our experimental environment, we ran experiments where the BOUNDHOLE algorithm was executed with different pdr values (i.e., pdr=0.7 and pdr=0.5). To consider the varying link quality over time, we ran the experiment 3 times a day at 12:00 PM, 6:00 PM, and 12:00 AM. We then counted the number of link-layer retransmissions for each node involved in the operation of the BOUNDHOLE algorithm. The results are depicted in Figure 5. As shown, a low threshold for the pdr resulted in a higher number of link-layer retransmissions, because we allowed nodes to select neighbors with unstable links. Especially, when we set the maximum allowable number of link-layer retransmissions to 10, the pdr threshold of 0.5 resulted in some broken links between boundary nodes. We found that in our experimental setting, generally, pdr $> .7$ gave reliable boundary.

To address the second challenge (i.e., dynamically changing link quality), we allow our Neighbor Discovery module to continually monitor the pdr of links. Whenever the pdr for a link between boundary nodes becomes smaller than the threshold, the BOUNDHOLE algorithm runs again and finds new boundaries. The third issue states that the BOUNDHOLE algorithm identifies even a very small hole – we consider holes with 4 boundary nodes or less as small holes. However, only large holes are of our interest, because small holes do not affect the path stretch much. Thus, only vertices for such

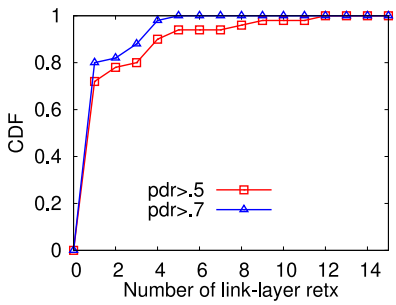

Fig. 5. Reliability of routing paths surrounding holes for different pdr's

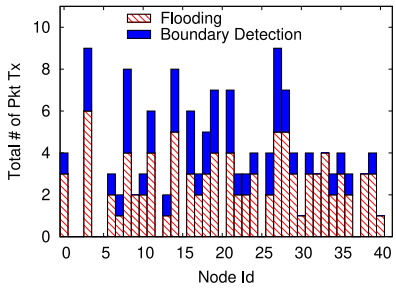

Fig. 6. Overhead for CNT detection/abstraction

large holes are broadcast throughout the network. Recall that, while the control packet is forwarded by the boundary nodes of a hole, the boundary abstraction process is also executed. The details for the boundary abstraction process are presented in the following section.

3.2. Boundary Abstraction

While the control packet for the BOUNDHOLE algorithm traverses the boundary nodes of a hole, the boundary nodes receiving the control packet perform the CNT abstraction process (i.e., a process for abstracting the boundary into a polygon). We found difficult to apply the state-of-the-art CNT abstraction algorithm [6] to a real-world setting. Its operation is conceptually simple, but it requires complex implementation with much higher time complexity mostly due to non-trivial geometric algorithms. Specifically, the state-of-the-art boundary abstraction is based on finding an arbitrarily oriented minimum bounding box which has the complexity of $O(N^2 \log N)$ [6], while the complexity of our boundary abstraction algorithm is $O(N)$ as it is based on a simple linear regression in constant two dimensions. For a resource-constrained node, we found that the implementation of the algorithm on a mote was a very challenging task. Furthermore, the state-of-the-art CNT abstraction requires the control packet to potentially store all locations of last visited boundary nodes (while traversing the boundary nodes for a hole). This requirement poses a significant limitation as the maximum packet size for 802.15 is only 133 Bytes including all headers [19]. Therefore, to make the CNT abstraction properly work in a real-world setting, we have to control the packet size and simplify the implementation to reduce the ROM size. To address these challenges, we develop a linear-regression-based CNT abstraction. It requires the control packet to contain only k (in our experimental setting, $k = 10$) locations of visited boundary nodes, where k is a user-specified parameter. When receiving the control packet, nodes perform a linear regression on at most k locations. When the correlation coefficient for the linear regression is larger than a predefined threshold, it selects the last node as a vertex and stores the location of the vertex in the control packet. Thus, when the control packet finishes traversing the boundary nodes and returns to the initiator, all vertices are identified and stored in the control packet.

Recall that the CNT Abstraction module is used by the CNT Detection module. Figure 6 shows the overhead for CNT detection/abstraction in terms of the number of packet transmissions including the link-layer retransmissions. As shown, the overhead consists of two parts: overhead for forwarding the control packet and overhead for flooding the locations of discovered vertices. Remarkably, the per-node number of packet transmissions was 2.26 on average in our experimental setting.

3.3. Cut Detection

The Cut Detection module is used by the 1-1 Routing module to determine whether a given destination is reachable or not. The Cut Detection module allows nodes to find a cut with respect to *any node*, i.e., realizing the peer-to-peer cut detection. The implementation of peer-to-peer cut detection is based on [20] which is different from existing cut detection

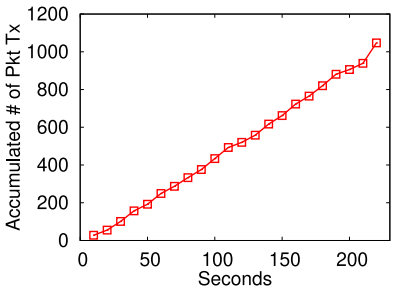

Fig. 7. Transmission overhead

algorithms [21–23] for LLNs in that existing solutions focus on finding cuts with respect to only a sink node. *One important reason why we choose [20] is that we can significantly simplify the implementation as the peer-to-peer cut detection algorithm relies on abstract CNT information, and the CNT Detection module already implements the CNT detection/abstraction.*

More specifically, the Cut Detection module uses the CNT Detection/Abstraction module, i.e., it uses the boundary information in the form of a polygon. Given the locations of source node s and destination node t, the decision on whether the two nodes can reach each other is based on an application of a point-in-polygon (PIP) problem, which finds whether a point is inside a given polygon or not. More formally, let $\mathcal{P}$ be the set of polygons that node s is aware of. We define a function $PIP(P, p) \rightarrow \{0, 1\}$, where $P \in \mathcal{P}$, and p is the location of a node, such that $PIP(P, p)$ is 1 if p is in polygon P; otherwise, $PIP(P, p)$ is 0. Then, given the two locations s and t, we can find whether the destination with location t is reachable or not by checking the condition: $\forall P \in \mathcal{P}, PIP(P, s)) \cdot PIP(P, t) > 0$. If the condition is true the destination is reachable; otherwise, it is not reachable.

As mentioned, the implementation of the Cut Detection module is significantly simplified, taking only 100 lines of codes, because it implements much of its functionality by reusing the code from the CNT Support component. However, the benefits are significant. Figure 7 depicts the accumulated number of packet transmissions when no cut detection mechanism is used. More specifically, we sent a packet from a fixed source node to a randomly selected destination node every 10 seconds. We observed that when the destination node is unreachable, packets traveled around the outer boundary of the disconnected segments of the network until the maximum TTL is reached, causing a significant number of unnecessary packet transmissions.

4. GROLL Routing Engine

4.1. 1-1 Routing

A number of routing protocols have been proposed to achieve a guaranteed small stretch. Flury et al. proposed an embedding algorithm that achieves $O(logn)$ stretch [5], where n is the number of nodes in the network. However, it is a centralized algorithm based on the Unit Disk Graph. Kermarrec and Tan [4] proposed a decomposition-based protocol with worst-case stretch of 7, but the stretch is possible by modeling a network in a continuous space domain based on a large number of time-synchronized flooding operations. Tan et al. introduced VIGOR [6] that achieves worst-case path stretch of $\Theta(1)$ without the UDG assumption and precise time-synchronization. However, the path stretch is possible with a "path-setup" process where a source node first exchanges a message with a destination node using a default geographic routing, prior to data transmission. This implies that the claimed path stretch no longer holds if we take the path built for the "path-setup" process into account. Although these protocols provide good path stretch, non of these algorithms were deployed in real-world environments.

Our 1-1 Routing module is designed to generate paths with guaranteed low stretch without relying on the "path-setup" process. More specifically, using the information from the CNT Support (i.e., the abstract CNT information and the reachability information), our 1-1 Routing module generates a path with guaranteed stretch of $O(r)$, where r is the diameter of the largest hole in the network without relying on the path-setup process. The implementation of the 1-1 Routing module is based on [6,24]. *The original protocol design of [6, 24], however, has been significantly improved by taking into account the constraints for real-world implementation such as limited memory and processing.*

4.1.1. Overview

We begin with an overview of the 1-1 Routing module. When a sender sends a packet, the CNT support component may or may not have completed its CNT detection/abstraction process. If the abstract CNT information is unavailable, the 1-1 Routing module uses the Geographic Forwarder module to send a packet. In particular, for the implementation of underlying geographic forwarder for the 1-1 Routing module, we greatly saves the required memory space for the Forwarder module compared with other well-known implementations [25,26] by not using the Face Routing [25]. Instead, as proposed in [11], we use the cycle of discovered boundary nodes for routing around holes. More specifically, when a packet is stuck at a local minimum, the packet is forwarded to the neighboring boundary node according to the right-hand rule [11] until greedy routing can be resumed.

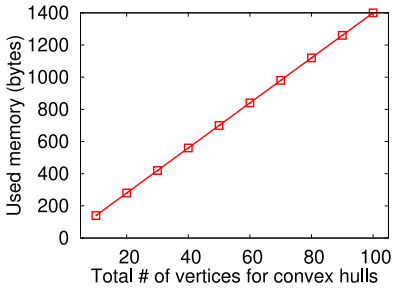

Fig. 8. Memory usage

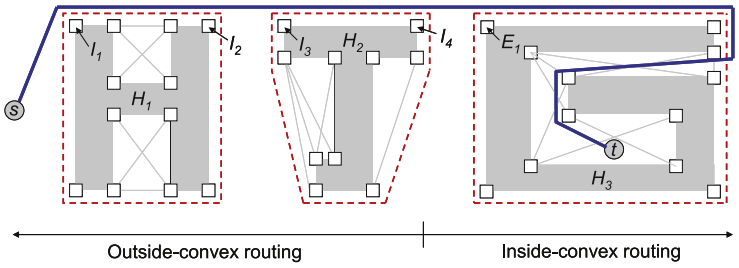

Fig. 9. An example of operation of the 1-1 routing module

If the abstract CNT information is available, the 1-1 Routing module operates in two modes: the outside-convex routing and inside-convex routing. The outside-convex routing is the default routing mode. When source node s wants to send a packet to a destination node t, s first uses the outside-convex routing to find a set of intermediate destinations; more precisely, the outside-convex routing computes a *Tangent Visibility Graph* defined by the edges corresponding to tangent lines between convex hulls of holes, and by the vertices corresponding to tangent points and source/destination nodes. Given the locations of vertices for a polygon of a hole, the convex hull for the hole can be easily computed. It then finds a shortest path between s and t on the tangent visibility graph. Consequently, the set of vertices on the shortest path on the tangent visibility graph are selected as intermediate destinations. For example, Figure 9 depicts the convex hulls of three holes denoted by red dotted lines. The hollow squares represent the vertex nodes. If node s performs outside-convex routing, it will identify I_1, I_2, I_3 and I_4 as its intermediate destinations. Source node s then sends a packet to first intermediate destination I_1, and I_2 to I_3, and so on.

The routing mode changes to the inside-convex routing in three cases: (1) s is inside a convex hull; (2) t is inside a convex hull; and (3) both s and t are inside convex hulls. The inside-convex routing uses the routing infrastructure called *Local Visibility Graph* which is constructed for each hole. The local visibility graph is defined for each hole such that the vertices are the vertex nodes of the hole, and the edges are line segments between two visible vertex nodes for the same convex hull of the hole. An example is shown in Figure 9, where there are three local visibility graphs for holes H_1, H_2, and H_3, respectively. The edges are represented as gray lines and vertices are represented as hollow squares. The implementation details on how to build a local visibility graph are provided in Section 4.1.3. Given local visibility graphs, when s is inside a convex hull, it uses the local visibility graph to route a packet optimally out of the convex hull and delivers the packet to a next intermediate destination, then outside convex routing is resumed. When the last intermediate destination node finds that destination t is inside a convex hull, it changes the routing mode to inside-convex routing and uses the local visibility graph to optimally route the packet to the destination node inside the convex hull. For example, in Figure 9, when a packet reaches the intermediate destination I_4, the node at I_4 changes its routing mode to the inside-convex routing and sends the packet along the locally optimal path inside the convex hull to the destination node t.

4.1.2. Outside-Convex Routing

Given convex hulls and the locations of source and destination nodes, a source node can construct a tangent visibility graph. The construction of a tangent visibility graph can be done in $\mathcal{O}(N \log N + h)$ and the space complexity is only $\mathcal{O}(N)$, where N is the number of vertices of convex hulls and h is the number of holes in a network [27]. More specifically, considering two end points (i.e., source/destination nodes), the outside-convex routing builds a tangent visibility graph with the two end points as degenerate convex hulls and computes the shortest path by applying the Dijkstra algorithm. Consequently, the outside-convex routing gives a set of intermediate destinations $\{I_1, I_2, ..., I_n\}$ which are the tangent points on the shortest path. To prove the feasibility for running the outside-convex routing on a mote, we measured required memory space for the outside-convex routing module by varying the number of vertex nodes. It is interesting to note that our results match the theoretical bound [27] (Figure 8), i.e., the required memory space linearly increases as the number of vertices increases.

As the figure shows, the memory space overhead for the real-world implementation is reasonable, taking up about 1 KBytes for a very large number of vertices of about 100.

4.1.3. Inside-Convex Routing

We adopt a distance vector routing to construct a local visibility graph. More specifically, for each hole, vertex nodes in the convex hull of the hole set their visible vertex nodes in the same convex hull as their virtual neighbors. Each node maintains a routing table where each entry contains a destination vertex node, the next vertex node to which a packet should be sent to reach the destination vertex node, and the cost in terms of Euclidean distance to reach the destination vertex node. This routing table is periodically – in our setting, every 10 seconds – exchanged with virtual neighbors. Since two visible neighbors might be multiple hops away from each other, to send a routing table to neighbors, we used our Geographic Forwarder. The routing table exchanges are continued until the routing table converges.

To measure the overhead for building a local visibility graph, we ran our 1-1 Routing module in the concave-hole scenario (Figure 4(b)). Figure 10 depicts the per-node communication overhead – the average accumulated number of packet transmissions and the maximum accumulated packet transmissions (including link-layer retransmissions) for building the local visibility graph. We found that in our experimental environment the convergence of the routing tables was achieved relatively quickly, i.e., with small amount of overhead. More specifically, all nodes finished constructing the local visibility graph at the 6-*th* iteration, and the average per-node communication overhead was about 10 packets.

Once vertex nodes have converged routing tables, local visibility graphs are constructed and nodes can use the inside-convex routing. Given the visibility graphs and the mechanisms for switching routing modes, our 1-1 Routing module achieves a bounded path stretch of $O(r)$, where r is the diameter of the largest hole in the network. Due to space constraints, we omit the theoretical analysis of the bound. To show that our 1-1 Routing module generates a path with small stretch, we compared the path lengths (in terms of hop counts) for our 1-1 Routing module with state-of-the-art geographic unicast routing protocol called VIGOR [6]. Note that to obtain routing paths for VIGOR, we ran VIGOR in a C++ simulator and obtained intermediate destinations off-line, and then ran VIGOR on our testbed. We also measured the path length for GPSR as a base line. We performed experiments for both scenarios for convex and concave holes. We randomly selected 20 pairs of source and destination nodes. For the same set of source-destination pairs, we measured hop counts for different routing protocols. Figures 11 and 12 depict the results for convex-hole scenario and concave-hole scenario, respectively. Interestingly, for both scenarios, our 1-1 Routing module achieved very close path lengths to the state-of-the-art protocol, without requiring the source node to send a control packet to a destination node using a default geographic routing protocol (e.g., GPSR), which is the main drawback of the state-of-the-art protocol [6]. Another interesting observation was that GPSR performed worse in the concave-hole scenario compared with convex-hull scenario. The reason is that for the concave-hole scenario, GPSR relied more on routing along the boundary of a hole than the convex-hole scenario, especially when sending a packet to nodes inside a convex hull.

We now show that our 1-1 Routing module has reliable packet delivery ratio. To measure the packet delivery ratio, we fixed our packet size to 90 Bytes (including the header) and varied the buffer size. More specifically, the packet size was

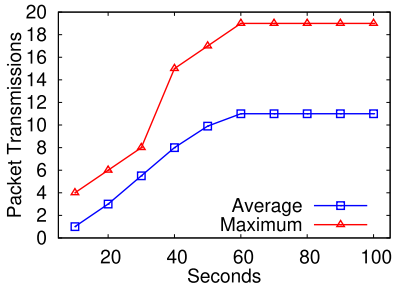

Fig. 10. Overhead for construction of local visibility graphs

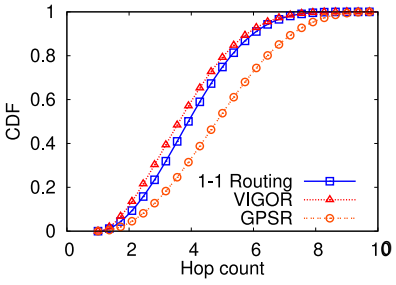

Fig. 11. Path length in hop count for the convex hole scenario

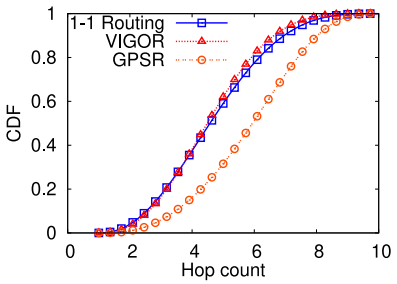

Fig. 12. Path length in hop count for the concave hole scenario

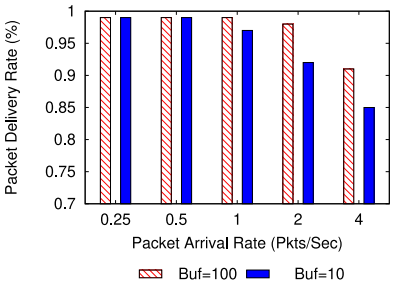

Fig. 13. Packet delivery ratio of the 1-1 routing module

selected based on the standard packet size of 127 Bytes with a maximum frame overhead of 25 Bytes for 802.15.4. Also experiments were performed with two distinctively different buffer sizes, *i.e.,* a larger one with 100 packets, and a smaller one with 10 packets to understand better the effect of the buffer size. Figure 13 shows the results. We observed that our 1-1 routing achieved reliable packet delivery ratio, mostly because it is based on underlying geographic forwarder, although with a small buffer size of 10, the packet delivery ratio decreased due to dropped packets. Interestingly, this result also proves that the underlying geographic forwarder works quite well without Face Routing.

4.2. 1-n Routing

There are largely two classes of widely-used geographic multicast routing protocols in the literature. The first kind is tree-based protocols [8,28,29]. These protocols do not require much protocol-related communication overhead (i.e., control packet transmissions). However, all locations of multicast members must be encoded in the header of a multicast packet. Since the maximum packet size for typical nodes is 133 Bytes including all headers [19], these protocols have a scalability issue when they must be implemented on real hardware. Even if we can encode the locations of multicast members in a header, larger packet sizes still cause a problem. More specifically, in our experimental environment, when we measured the total number of link-layer retransmissions used for sending a packet from one end of the test-bed to the other end, a packet of size 120 Bytes had a significantly larger number of link-layer retransmissions than a packet of 60 Bytes, as depicted in Figure 14.

Hierarchical geographic multicast routing [7,9,30] addresses the scalability issue by dividing the network into subregions and separately handling multicast members in different regions. However, hierarchical solutions fail to provide optimal paths in terms of path lengths to multicast members. In contrast to existing geographic multicast routing protocols (i.e., widely-used tree-based and hierarchical geographic multicast protocols), our 1-n routing module adopts a “hybrid method” [31] that

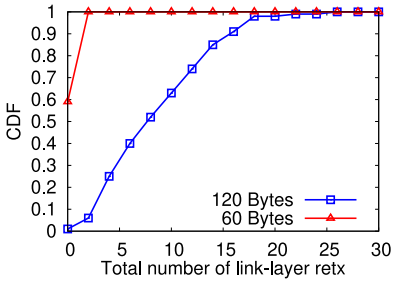

Fig. 14. Total number of link layer retransmissions on a path with different packet sizes

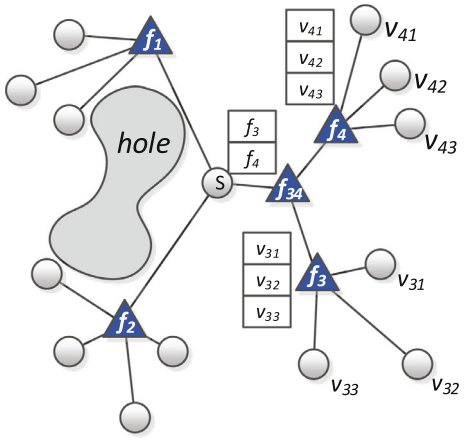

Fig. 15. An example of operation of the 1-n routing module

addresses both the packet-header-size issue and the suboptimal-routing-path problem. *However, it is important to note that we make various changes to the original protocol design [31] to enable the protocol to run on real hardware*. The design decisions and implementation details are described in the following sections.

4.2.1. Protocol Design and Implementation

Given the limited memory space of a node, we design that the sink node manages multicast members. In other words, the sink node stores the IDs and locations of multicast members for each multicast group. When a node wants to join a multicat group, it sends a *Join Message* containing the multicast group ID, and its ID and location, to the sink node; similarly when a node wants to leave from a multicast group, it sends a *Leave Message* with the multicast group ID and its ID to the sink node. When a node needs to send a packet to a set of nodes in a multicast group, it sends a *Request Message*, containing its location, the multicast group ID, and the level (we will discuss the details on "levels" shortly) to the sink node. Upon receiving the request packet, the sink node computes the optimal path to each multicast member and sends the result to the requested node. It is also worth to mention that since only the sink node manages multicast membership, communication overhaed for these multicast membership-related messages is reduced, which would otherwise be incurred if we allow any node to manage the multicast membership. There are three objectives in computing routing paths to multicast members: (1) Minimize the path lengths to multicast members (addressing the sub-optimal routing path problem); (2) Enable energy-efficient packet-loss recovery; (3) Limit the packet header size by a system parameter (addressing the scalability issue).

To achieve these objectives, our 1-n Routing module adopts the concept of a *facility node* [31]. The basic idea is that a source node first sends a packet to a designated set of nodes called *facility nodes*; then each facility node distributes the packet to assigned multicast members. More specifically, given the locations of multicast members and abstract CNT information, the sink node optimally selects facility nodes such that the sum of path lengths from the source node to all multicast members is minimized. For example, in Figure 15, sender s first sends a packet to facility nodes represented as triangles. Then the facility nodes distribute the packet to their multicast members. Also note that a facility node may serve another facility node to obtain shorter paths at the cost of storage overhead. We call this *multi-level facility mechanism*. We will discuss this mechanism in detail in Section 4.2.3. Consequently, the resulting topology for our 1-n Routing module achieves the three objectives as follows. First, the resulting hierarchical topology enables us to limit the packet header size by the capacity of a facility node, addressing the scalability issue. Second, our 1-n routing minimizes the total sum of path lengths in the presence of network holes, addressing the sub-optimal routing path problem. Third, nodes achieve energy-efficient packet-loss recovery by requesting packet-retransmission to nearby facility nodes instead of the source node.

4.2.2. Facility Node Selection

This section presents details on how to solve the problem of optimally choosing facility nodes. Facility nodes are indexed by i, and member nodes by j. Given the indices, the j-th member node assigned to facility node f_i is denoted by v_{ij}. For example, facility node f_1 may be assigned member nodes denoted by $\{v_{11}, v_{12}, v_{13}\}$. The shortest Euclidean distance between two nodes v_i and v_j is denoted by $d(v_i, v_j)$. This shortest distance is computed based on a visibility graph consisting of the sink, multicast members, and the abstract CNT information as polygons, which is known from our CNT Support component. The problem is then formulated as the following mixed integer program:

$$\text{minimize} \quad \sum_i \sum_j d(f_i, v_{ij}) x_{ij} + \sum_i d(f_i, s) y_i \tag{1}$$

$$\text{subject to:} \quad \sum_i x_{ij} = 1 \text{ for all } j, \tag{2}$$

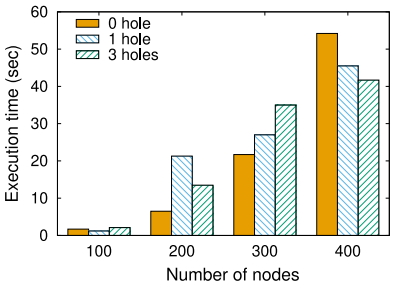

Fig. 16. Execution time of the facility-location solver

$$\sum_j x_{ij} \leq s_i y_i \text{ for all } i, \tag{3}$$

$$x_{ij} = \{0, 1\} \text{ for all } i \text{ and } j, \tag{4}$$

$$y_i = \{0, 1\} \text{ for all } i \tag{5}$$

Here x_{ij} and y_i are the indicator variables: $y_i = 1$ indicates that a facility node is available at index i. And $x_{ij} = 1$ indicates that a member node at index j is assigned to a facility node at index i. The term s_i refers to the capacity of a facility node at index i, specifying the maximum number of members that can be assigned. The first constraint guarantees that a member node is assigned to only one facility node. The second constraint specifies that no facility node can handle more than its capacity. The third and fourth constraints specify the integrality of variables x_{ij} and y_i.

This mixed integer problem selects a set of facility nodes such that the sum of Euclidean path lengths from the sink node to member nodes is minimized. We acknowledge that there are key characteristics of wireless communication such as constructive interference that can be adopted to achieve better performance [32,33]. Although the current design of the proposed framework is tightly built on top of a geographic forwarder component, it is worth to note that it is possible to incorporate different forwarding mechanisms as an underlying forwarding component, e.g., a geographically constrained broadcast based on constructive interference [32,33].

We implemented an exact solver for this problem using MATLAB 2011b 64 bit on a PC running Windows7 64 Bit with Intel i7 920 Processor and 24 GBytes of RAM. It should be noted, however, that our facility node selection problem is NP-Hard because it is formulated as a mixed integer program. This means that although the exact solver works well for the network size of our experiments, if the network size is very large, the exact solver may take too much time to calculate the facility nodes. Fortunately, there exists various heuristic solutions for the facility node problem that can be adopted for experiments with a large number of nodes. Figure 16 depicts the computation time for different numbers of multicast members and holes. It shows that there is an increasing trend of execution time with the increasing number of multicast member nodes. The results also demonstrate that our exact solver computes facility locations quite quickly for a relatively large number of multicast members. However, we were not able to find a clear trend related to the number of holes. The reason is because as we have more holes, there is a chance that member nodes may be covered by the holes and not considered for this computation. We also advise that, for a very large network with thousands of member nodes, users may adopt a heuristic solvers for capacitated facility location problem [34,35]. We then evaluated the performance of the 1-n routing by measuring the total sum of path lengths for a given set of multicast members and compare it to the state-of-the-art hierarchical geographic multicast routing called HGMR [7]. For this set of experiments, we used 6 different sets of randomly selected multicast members and ran a multicast routing 20 times for each set in a concave-hole scenario. Figure 17 shows the results. As shown, our 1-n Routing module produced paths with smaller sum of path lengths compared

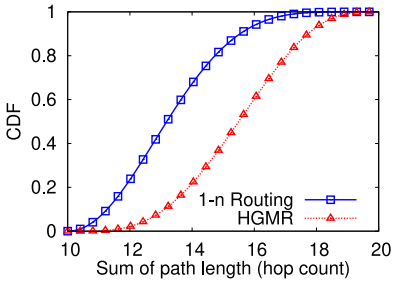

Fig. 17. Total sum of path lengths in terms of hop count

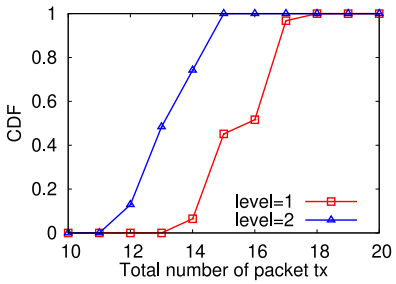

Fig. 18. Communication overhead for different facility levels

with HGMR. Interestingly, we observed that HGMR produced much worse path lengths than expected. The reasons are: first, HGMR does not optimize the path length when determining APs (i.e., the corresponding concept for our facility node) to multicast members; second, maybe more importantly, HGMR does not consider holes in the network.

4.2.3. Multi-level Facility Node Selection

The 1-n Routing module employs the multi-level facility mechanism to achieve higher energy savings by reducing the total sum of path length to multicast members. The basic idea is that after running our solver for finding the first-level facility nodes, we run our solver again, with the selected facility nodes in the first-level as multicast members, obtaining the facility nodes in the second level, and so on. Figure 15 depicts an example where f_1, f_2, f_3, and f_4, facility nodes in the first level, serve multicast members denoted by v_{ij}, and f_{34}, a facility node in the second level, serves facility nodes f_3 and f_4 in the first level. The sink node then sends the locations of selected facility nodes in the highest level to the source node (packet type (a)) and sends the locations of either assigned member nodes or facility nodes in the lower levels to selected facility nodes (packet type (b)). A source node specifies the desired level as a parameter, so that the sink computes the multi-level appropriately.

To see the benefits of the multi-level facility mechanism, a packet was sent to 8 pre-selected multicast members. We then measured the total number of packet transmissions for both facility level of 1 and facility level of 2. Figure 18 shows the results. The figure shows that when we used a higher level, we could reduce the number of packet transmissions by aggregating some paths to multicast members. However, we must note that higher facility levels require more nodes to work as facility nodes, increasing the storage overhead in the network.

4.2.4. Packet Recovery

The design of 1-n Routing module has an additional benefit of fast packet-loss recovery. For the packet loss recovery, we adopt a widely used NACK-based algorithm. More specifically, assume that a node received packets with sequence numbers 1, 2, and 4. The receiver finds that packet 3 is missing and may request retransmission to a source node. The 1-n Routing module makes this retransmission mechanism more energy efficient (in terms of number of packet transmissions used for packet-loss recovery) by allowing the receiver to request retransmissions directly to a nearby facility node. To verify the advantage of the "direct packet-loss recovery" from a facility node, we artificially caused packet losses at multicast members with probability.9. A source node sent a packet to four multicast members every second. We then counted the total number of accumulated packet transmissions for both the "direct packet loss recovery" and the "packet loss recovery from the source node". Figure 19 shows the results. As the result shows, the direct packet-loss recovery from a facility node significantly reduced communication overhead. An interesting observation is that this performance difference becomes larger and larger as time elapses – at only 1000 second, we reduced the number of packet transmissions by up to 70% even in our small test-bed environment.

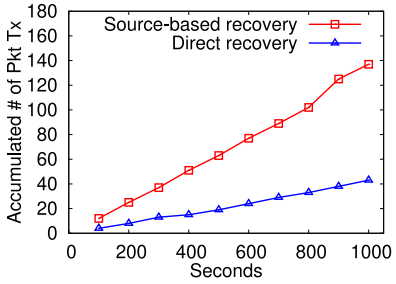

Fig. 19. Comparison of overhead for packet-loss recovery

4.3. n-1 Routing

The n-1 routing is an important routing primitive for LLNs because major part of network traffic is for nodes to report their sensed data to the sink node. One can observe that we can implement our n-1 Routing module by simply reusing the 1-1 Routing module. However, one important issue is the *energy hole problem* [36]: since all nodes send packets to the sink, the nodes around the sink consume higher energy because they handle higher network traffic. Fortunately, this type of energy hole problem has received sufficient attention, and many solutions have been proposed [10,36]. However, when there are holes in the network, especially large ones, the energy hole problem occurs in overlooked regions other than around the sink node. To illustrate this scenario, see Figure 20, where there is a large hole in the network. In this scenario, nodes in region denoted by *A*, when they use our 1-1 Routing module, will send packets along the paths that are designed to optimally detour the hole. A problem is that all such paths pass through common areas denoted by the red zones called "hot zones". An energy hole problem arises in these hot zones.

Therefore, the design of our n-1 Routing module is focused on addressing the energy hole problem around "hot zones". The key idea is to allow nodes to use *virtual boundaries* when they send a packet to the sink. The virtual boundary is formed by extending the original boundary of a hole – more precisely, the convex polygon of the hole. Figure 20 shows an example of the 2-level virtual boundary, which extends the original boundary also called level-1 boundary. When the computed virtual boundary intersects level-1 boundaries (i.e., original boundaries), the virtual boundary is not used, to prevent concentrated traffic on the level-1 boundaries; however, intersection with other virtual boundaries with levels greater than 1 is allowed. Also, depending on the locations of virtual boundaries, it is possible that there is no node at the location of the vertex for a virtual boundary. In this case, our geographic forwarder sends a packet to the closest node to the location of the vertex. The maximum level for virtual boundary is given as a system parameter.

To evaluate the performance of our n-1 Routing module, we placed a hole in the network as shown in Figure 21. In this figure, the virtual boundary for the hole is represented as red dotted convex polygon. We made nodes with IDs {28, 30, 34, 33, 32, 31, 29, 4, 25, 10, 19, 37} send a packet to the sink node with ID 40. We measured the total number of packet transmissions including link-layer retransmissions. Figure 22 shows the results. As shown, when our virtual boundary mechanism was not used, the network traffic was concentrated on the boundary of the hole and on the path to the sink. However, when the virtual boundary mechanism was used, the energy consumption was more evenly distributed, resulting

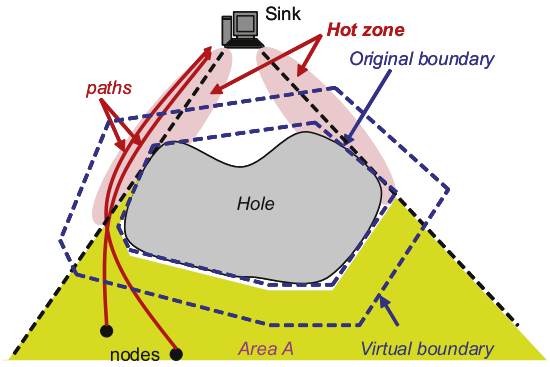

Fig. 20. The energy hole problem in "hot zones"

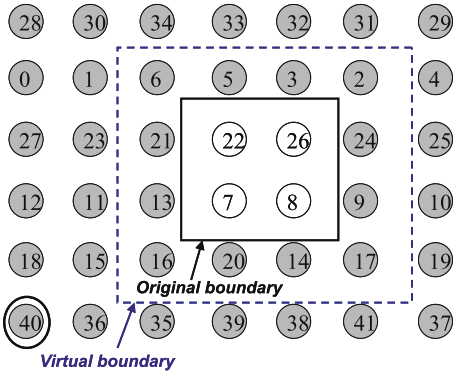

Fig. 21. A scenario for evaluation of n-1 Routing

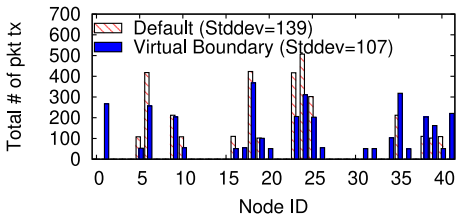

Fig. 22. Energy consumption for default convergecast and convergecast based on virtual boundary

in the reduction of the standard deviation for the total number of packet transmissions from 139 to 107 – achieving a 23% decrease.

4.3.1. Large-scale Simulation for n-1 Routing Module

Large-scale simulations were performed to supplement the results obtained from our small-scale test-bed consisting of 41 TelosB motes. Simulation results for large-scale networks for the 1-1 and 1-n modules have been reported [31,37]. Thus, in this section, we focus on analyzing the performance of the n-1 module in a large-scale network consisting of 5,220 nodes in an area of 2,000m by 2,000m, by comparing with the state-of-the-art convergecast protocol, i.e., CTP (Collection Tree Protocol) [38]. Specifically, the impact of convergecast protocols on the energy hole problem was studied. Nodes had a communication range of 60m. The communication range was modeled based on [39] to simulate realistic wireless communication. The sink node was placed at the bottom left corder at location (0,0) (Figure 24(a)). To study the energy hole problem, a bar-shaped obstacle was installed in the middle of the network. Source nodes were then randomly and uniformly selected to send a packet to the sink node. This process was repeated 10,000 times, and we counted the number of packet transmissions for each node in the network, which was used to represent the energy consumption of nodes.

Figures 24 (a), 24(b), and 24(c) illustrate the number of packet transmissions of nodes (i.e., represented by a color as shown in the bar graph next to each figure) for CTP, n-1 routing module with 2-level virtual boundary, and n-1 routing with 3-level virtual boundary, respectively. In this simulation, the CTP built a minimum cost tree with the hop count to reach the sink node as the main cost. In these figures, a spot with red color represents an area where a significantly large number of packet transmissions were performed. As such, the energy hole problem around "hot zones" was observed in Figure 24(a). As illustrated in Figures 24(b) and 24(c), when we increased the virtual boundary level, the color of the red spots was blurred, which indicates that packet transmissions concentrated on nodes around the hot zones were successfully spread over surrounding nodes

To understand better the number of packets transmissions, Figure 23 displays the CDF of the number of packet transmissions for CTP and n-1 routing module with 3-level virtual boundary. About 98% of nodes had less than 600 packet transmissions for both protocols as shown in the magnified graph for number of packet transmissions between 0 and 1200. Interestingly, the number of nodes with more than 600 packet transmissions for n-1 routing was smaller than that of CTP.

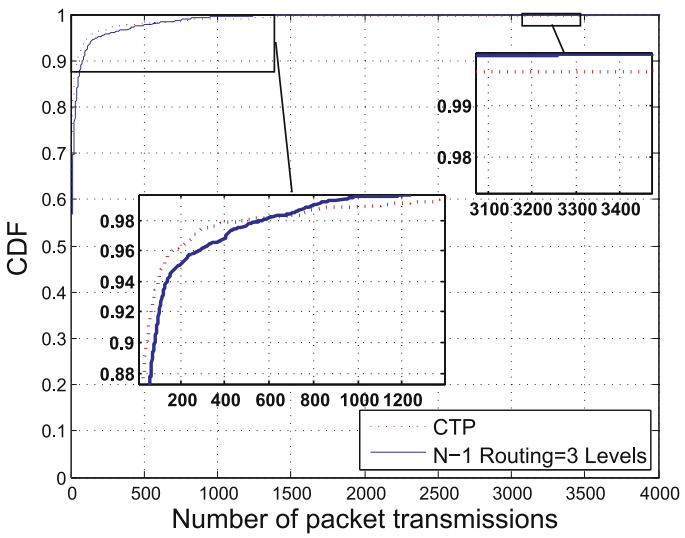

Fig. 23. The CDF of number of packet transmissions for CTP and n-1 routing module

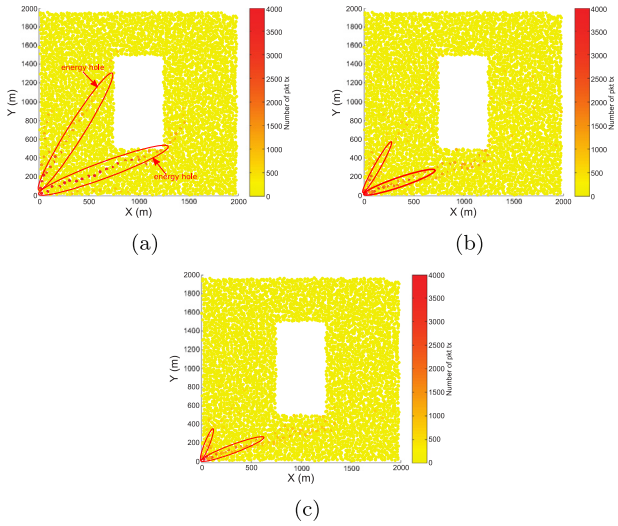

Fig. 24. The number of packet transmissions for (a) CTP; (b) n-1 routing with 2 levels; (c) n-1 routing with 3 levels

Although these nodes take at most 2% of all nodes, they are the nodes that cause the energy hole problem. It can be seen that the number of these nodes were effectively reduced by our n-1 routing module.

5. Related Work

Developing an unified routing framework was considered important but hard to achieve. Nasir et al. [40] remarked the importance of developing an unified routing framework and left it as a challenge. Le et al. [41] mentioned that an unified routing scheme is needed especially for Internet of Things where heterogeneous devices such as smartphones, vehicles, and nodes are deployed, but the details were not presented. Despite the difficulties, there have been some efforts to develop unified routing schemes [42,43]. Garcia et al. [42] considered a unified routing protocol for ad-hoc networks. However, the luxury of enough data storage, i.e., one of the major requirements for the protocol, is not usually available in LLNs for IoT. In the vehicular networks research domain, researchers considered developing a unified routing design [43], but the details of the protocol design were not provided.

In many cases, the concept of unifying routing protocols has been approached from various perspectives, rather than just unifying routing protocols designed for different traffic patterns. For example, Le et al. [41] focused on unifying protocols designed for heterogeneous device types. Kwon et al. [44] focused on unifying routing protocols designed with different system elements, i.e., transmission power, interference, and residual energy. Nasir et al. [40] considered the problem of unifying routing protocols designed for different multi-radio environments, e.g., distinct bandwidth differences. Chatterjee and Das [45] provided a unified routing scheme for different routing protocols with different QoS constraints, e.g., packet deliver ratio, end to end delay, and routing overhead. Musoleski, similarly, tried to unify routing protocols for different networking domains [46].

RPL (IPv6 Routing Protocol for Low-Power and Lossy Networks) [1] was developed by IETF “Routing Over Low power and Lossy (ROLL) networks” working group. It provides a mechanism whereby multipoint-to-point, point-to-multipoint, and point-to-point traffic are supported. However, this protocol is based on a Destination-oriented Directed Acyclic Graph (DODAG). Once nodes are deployed, they form DODAG by selecting one or more nodes as the root nodes and forming a acyclic tree from the root nodes. As can be seen from the topology construction mechanism, the protocol is mainly designed for a particular traffic pattern: many-to-one. Although RPL supports other traffic types than just convergecasting, unfortunately, it does not provide fully efficient performance due to its inherent protocol design for data collection. More specifically, RPL provides only a basic structure for point-to-point traffic. In fact, in their specification [1], the authors state that “RPL neither specifies nor precludes additional mechanisms for computing and installing potentially more optimal routes to support arbitrary P2P traffic”. The literature also notes that the point-to-multipoint traffic has been somewhat overlooked as well, resulting in the development of alternative multicast solutions based on RPL [47].

6. Conclusions

We have presented the first CNT-aware location-based unified routing framework. It integrates routing modules for three main routing primitives (i.e., 1-1, 1-n, and n-1 routing) and seamlessly provides unified routing service to applications. We expect that our GROLL framework benefits many IoT applications for real-world deployments where diverse traffic patterns are needed in challenging environments where the network topology dynamically changes creating irregular topologies like

holes and cuts. For example, developing an unified framework for 5G would be an interesting future research direction because a 5G network will consist of various kinds of nodes such as low-power sensors, connected vehicles, mobile phones, *etc.* which will require diverse communication patterns such as device to device, device to multiple devices, and device to all devices in a specific region, *etc.*

Declaration of Competing Interest

All authors have participated in (a) conception and design, or analysis and interpretation of the data; (b) drafting the article or revising it critically for important intellectual content; and (c) approval of the final version.

This manuscript has not been submitted to, nor is under review at, another journal or other publishing venue.

The authors have no affiliation with any organization with a direct or indirect financial interest in the subject matter discussed in the manuscript

References

[1] T. Winter, RPL: IPv6 routing protocol for low-power and lossy networks, 2014, (https://tools.ietf.org/html/rfc6550). Accessed: 2015-07-21.
[2] U. Herberg, T. Clausen, Y. Igarashi, J. Yi, A. Verdiere, Observations of rpl: Ipv6 routing protocol for low power and lossy networks, 2012, (https://tools.ietf.org/html/draft-clausen-lln-rpl-experiences-05). Accessed: 2015-07-21.
[3] S.M. George, W. Zhou, H. Chenji, M. Won, Y. Lee, A. Pazarloglou, R. Stoleru, P. Barooah, DistressNet: a wireless AdHoc and sensor network architecture for situation management in disaster response, IEEE Communications Magazine 48 (3) (2010).
[4] A.-M. Kermarrec, G. Tan, Greedy geographic routing in large-scale sensor networks: a minimum network decomposition approach, in: Proc. of MobiHoc, 2010.
[5] R. Flury, S. Pemmaraju, R. Wattenhofer, Greedy routing with bounded stretch, in: Proc. of INFOCOM, 2009.
[6] G. Tan, M. Bertier, A.-M. Kermarrec, Visibility-graph-based shortest-path geographic routing in sensor networks, in: Proc. of INFOCOM, 2009.
[7] D. Koutsonikolas, S.M. Das, Y.C. Hu, I. Stojmenovic, Hierarchical geographic multicast routing for wireless sensor networks, Wireless Network 16 (2) (2010) 449–466.
[8] J. Sanchez, P. Ruiz, I. Stojmnenovic, Gmr: Geographic multicast routing for wireless sensor networks, in: Proc. of IEEE SECON, 2006.
[9] X. Xiang, X. Wang, Y. Yang, Stateless multicasting in mobile ad hoc networks, IEEE Transactions on Computers 59 (2010) 1076–1090.
[10] H. Zhang, H. Shen, Balancing energy consumption to maximize network lifetime in data-gathering sensor networks, IEEE Transactions on Parallel and Distributed Systems 20 (10) (2009) 1526–1539.
[11] Q. Fang, J. Gao, L.J. Guibas, Locating and bypassing holes in sensor networks, in: Proc. of IEEE INFOCOM, 2004.
[12] M. Fayed, H.T. Mouftah, Localised alpha-shape computations for boundary recognition in sensor networks, Ad Hoc Networks 7 (6) (2009) 1259–1269.
[13] A. Kröller, S.P. Fekete, D. Pfisterer, S. Fischer, Deterministic boundary recognition and topology extraction for large sensor networks, in: Proc. of SODA, 2006.
[14] Y. Wang, J. Gao, J.S. Mitchell, Boundary recognition in sensor networks by topological methods, in: Proc. of MobiCom, 2006.
[15] D. Dong, Y. Liu, X. Liao, Fine-grained boundary recognition in wireless ad hoc and sensor networks by topological methods, in: Proc. of MobiHoc, 2009.
[16] S. Funke, Topological hole detection in wireless sensor networks and its applications, in: Proc. of DIALM-POMC, 2005.
[17] S.P. Fekete, A. Kroeller, D. Pfisterer, S. Fischer, C. Buschmann, Neighborhood-Based Topology Recognition in Sensor Networks, in: Proc. of ALGOSENSORS, 2004.
[18] S.P. Fekete, M. Kaufmann, A. Kroeller, K. Lehmann, A New Approach for Boundary Recognition in Geometric Sensor Networks, eprint arXiv:cs/0508006 (2005).
[19] C.-J.M. Liang, N.B. Priyantha, J. Liu, A. Terzis, Surviving wi-fi interference in low power zigbee networks, in: Proc. of Sensys, 2010.
[20] M. Won, R. Stoleru, Destination-based cut detection in wireless sensor networks, in: Proc. of IEEE/IFIP EUC, 2011.
[21] M. Won, M. George, R. Stoleru, Towards robustness and energy efficiency of cut detection in wireless sensor networks, Elsevier Ad Hoc Networks 9 (3) (2011) 249–264.
[22] P. Barooah, H. Chenji, R. Stoleru, T. Kalmár-Nagy, Cut detection in wireless sensor networks, IEEE Transactions on Parallel and Distributed Systems 23 (3) (2012) 483–490.
[23] N. Shrivastava, S. Suri, C. Tóth, Detecting cuts in sensor networks, ACM Transactions on Sensor Networks 4 (2) (2008) 1–25.
[24] M. Won, R. Stoleru, H. Wu, Geographic routing with constant stretch in large scale wireless sensor networks with holes, in: Proc. of WiMob, 2011.
[25] B. Karp, H.T. Kung, GPSR: greedy perimeter stateless routing for wireless networks, in: Proc. of MobiCom, 2000.
[26] Y.-J. Kim, R. Govindan, B. Karp, S. Shenker, Geographic routing made practical, in: Proc. of NSDI, 2005.
[27] M. Pocchiola, G. Vegter, Topologically sweeping visibility complexes via pseudotriangulations, 1996.
[28] M. Mauve, H. Fussler, J. Widmer, T. Lang, Position-based multicast routing for mobile ad-hoc networks, SIGMOBILE Mob. Comput. Commun. Rev. 7 (3) (2003) 53–55.
[29] S. Wu, K. Candan, GMP: Distributed geographic multicast routing in wireless sensor networks, in: Proc. of IEEE ICDCS, 2006.
[30] S.M. Das, H. Pucha, Y.C. Hu, Distributed hashing for scalable multicast in wireless ad hoc networks, IEEE Transactions on Parallel Distributed System 19 (3) (2008) 347–362.
[31] M. Won, R. Stoleru, Energy effcient and robust multicast routing for large scale sensor networks, in: Proc. of EUC, 2011.
[32] M. Doddavenkatappa, M.C. Chan, B. Leong, Splash: Fast data dissemination with constructive interference in wireless sensor networks., in: Proc. of NSDI, 2013.
[33] F. Ferrari, M. Zimmerling, L. Thiele, O. Saukh, Efficient network flooding and time synchronization with glossy, in: Proc. of IPSN, 2011.
[34] D.B. Shmoys, E. Tardos, K. Aardal, Approximation algorithms for facility location problems, in: Proc. of STOC, 1997.
[35] M.R. Korupolu, C.G. Plaxton, R. Rajaraman, Analysis of a local search heuristic for facility location problems, in: Proc. of SODA, 1998.
[36] X. Wu, G. Chen, S.K. Das, Avoiding energy holes in wireless sensor networks with nonuniform node distribution, IEEE Transactions on Parallel and Distributed Systems 19 (5) (2008) 710–720.
[37] M. Won, R. Stoleru, A low-stretch-guaranteed and lightweight geographic routing protocol for large-scale wireless sensor networks, ACM Transactions on Sensor Networks (TOSN) 11 (1) (2014) 18.
[38] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, P. Levis, Collection tree protocol, in: Proc. of SenSys, 2009.
[39] T. He, C. Huang, B.M. Blum, J.A. Stankovic, T. Abdelzaher, Range-free localization schemes for large scale sensor networks, in: Proc. of MobiCom, 2003.
[40] M.K. Nasir, M.K. Sohel, M.T. Rahman, A. Islam, A review on position based routing protocol in vehicular adhoc network, American Journal of Engineering Research (2013).
[41] V.-D. Le, H. Scholten, P. Havinga, Unified routing for data dissemination in smart city networks, in: Internet of Things (IOT), 2012 3rd International Conference on the, 2012.
[42] J.J. Garcia-Luna-Aceves, D.A. Beyer, T.J. Frivold, Unified routing scheme for ad-hoc internetworking, 2007, US Patent 7,159,035.
[43] Y. Qian, N. Moayeri, Design of secure and application-oriented vanets, in: Proc. of VTC Spring, 2008.

[44] S. Kwon, N.B. Shroff, Energy-efficient unified routing algorithm for multi-hop wireless networks, IEEE Transactions on Wireless Communications 11 (11) (2012) 3890–3899.
[45] S. Chatterjee, S. Das, Ant colony optimization based enhanced dynamic source routing algorithm for mobile ad-hoc network, Information Sciences 295 (2015) 67–90.
[46] M. Musolesi, C. Mascolo, A framework for multi-region delay tolerant networking, in: Proceedings of the 2008 ACM workshop on Wireless networks and systems for developing regions, 2008.
[47] G. Oikonomou, I. Phillips, T. Tryfonas, Ipv6 multicast forwarding in rpl-based wireless sensor networks, Wireless personal communications 73 (3) (2013) 1089–1116.