Communicating via Fireflies: Geographic Routing on Duty-Cycled Sensors

Suman Nath
Microsoft Research
sumann@microsoft.com

Phillip B. Gibbons
Intel Research Pittsburgh
phillip.b.gibbons@intel.com

ABSTRACT

Geographic routing is a useful and scalable point-to-point communication primitive for wireless sensor networks. However, previous work on geographic routing makes the unrealistic assumption that *all* the nodes in the network are awake during routing. This overlooks the common deployment scenario where sensor nodes are duty-cycled to save energy. In this paper we investigate several important aspects of geographic routing over duty-cycled nodes. First, we extend existing geographic routing algorithms to handle the highly dynamic networks resulting from duty-cycling. Second, we provide the first formal analysis of the performance of geographic routing on duty-cycled nodes. Third, we use this analysis to develop an efficient decentralized sleep scheduling algorithm for reducing the number of awake nodes while maintaining both network coverage and a (tunable) target routing latency. Finally, we evaluate via simulation the performance of our approach versus running existing geographic routing algorithms on sensors duty-cycled according to previous sleep scheduling algorithms. Our results show, perhaps surprisingly, that a network of duty-cycled nodes can have slightly better routing performance than a static network that uses comparable energy. Our results further show that, compared to previous algorithms, our sleep scheduling algorithm significantly improves routing latency and network lifetime.

Categories and Subject Descriptors: C.2.1 [**Network Architecture and Design**]: Wireless communication, C.3 [**Special-Purpose and Application-Based Systems**]: Real-time and embedded systems, D.4.1 [**Process Management**]: Scheduling

General Terms: Algorithms, Performance, Theory.

Keywords: geographic routing, sleep-scheduling algorithm.

1. INTRODUCTION

Geographic routing algorithms [11, 12, 13, 14] are attractive solutions for point-to-point communication in wireless sensor networks, because they scale better: the routing state maintained per node is dependent only on the local network density and not on the network size. Geographic routing algorithms have also been proposed as a routing primitive for data-centric storage [20], in-network indexing for multi-dimensional range queries [15], geo-spatial queries [6], etc. Recent work on geographic routing [12, 13] has shown that such routing can be done quite efficiently, even in the presence of irregular radio ranges and localization errors.

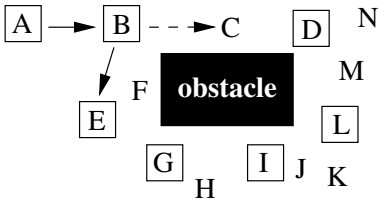


Figure 1: Geographic routing on duty-cycled nodes. Each letter represents a sensor node; awake nodes are in boxes. En-route from A to D, node B must decide between waiting for C to wake up or routing the long way around the obstacle (i.e., routing along the path B, E, G, I, L, D).

A key drawback of previous work on geographic routing is that *all existing geographic routing algorithms and their analysis assume all the nodes in the network are awake during routing*. However, in practical deployments sensors are duty-cycled to save energy [7]. With duty-cycling, nodes are awake or asleep in each time epoch (like fireflies flashing on and off) according to some scheduling algorithm, making the networks highly dynamic in terms of both global connectivity and the number of (awake) neighbors per node. We say such networks experience *time-varying connectivity (TVC)*. TVC networks raise routing issues not present in the previously-studied setting. As illustrated in Figure 1, in a TVC network, a message can either be forwarded over the currently awake nodes by using opportunistic routing algorithms [2] (e.g., take the long path in the figure), or be temporarily buffered in enroute nodes until a better next hop node wakes up (e.g., wait for C to wake up). In the former case, the number of hops may increase significantly incurring high energy overheads. In the latter case, the end-to-end latency may increase significantly (e.g., if C is not scheduled to wake up for many epochs) and the buffering requirements and waking times also increase.

In this paper, we study several important questions related to geographic routing on duty-cycled nodes. First, *how does duty-cycling affect the performance of geographic routing?* Intuitively, routing latency increases with the fraction of sleeping nodes because of sub-optimal routes or enroute buffering, but existing literature does not provide sufficient insights to formally reason about the relationship between sleep scheduling and routing latency. For example, it is not clear how much a geographic routing algorithm will suffer if a 5000-node multi-hop network chooses to keep only 10% of its nodes awake in each epoch. Moreover, in contrast to the all-nodes-awake scenarios studied previously, TVC networks experience *transient* connectivity structures (such as the local minimum

IPSN'07, April 25-27, 2007, Cambridge, Massachusetts, USA.

node B in Figure 1). It is not clear how these affect routing performance. Understanding these aspects can help system designers predict routing latency and network lifetime for a duty-cycled network, or choose appropriate duty-cycling parameters to achieve a target routing latency and network lifetime.

The second question we address is: *do existing sleep scheduling algorithms [1, 3, 8, 24], which optimize mainly for coverage and detection latency, provide good routing performance?* Existing work shows that geographic routing performs almost optimally when the average node degree is more than 12 [14]. Our results confirm the intuition that the performance of geographic routing depends not only on the average node degree, but also on the *distribution of node degrees*. Consider a dense deployment of 5000 sensors placed randomly in a 10 unit by 10 unit space such that each node has a transmission radius of 1 unit. Assume that nodes use a simple random sleep scheduling algorithm where each node independently decides to wake up in every epoch (like ExScal sensors [7]) with probability $\frac{1}{10}$. In such a network, although the average node degree (number of awake nodes) in an epoch is around 14, some of the nodes have very high (> 25) degree while others have small degree (< 5)[1] (details of the result are in Section 7.6). Small-degree nodes tend to be easily disconnected from the network, making routing failures likely. On the other hand, high-degree nodes suffer from high traffic contention and tend to draw a large portion of the routing load, running out of battery power faster than other nodes. These problems can be avoided by ensuring a more uniform node degree. Existing sleep-scheduling algorithms do not aim to optimize the node degree distribution and hence are susceptible to the above problems. For example, under one of our evaluation scenarios, the simple random scheduling scheme mentioned above generates networks that are disconnected $\approx 5\%$ of the time and where a few nodes experience as much as $10\times$ more routing load than a few other nodes.

We address the above problems in the final question: *how can we design a sleep scheduling algorithm that provides good routing performance?* In light of the above discussions, in order to optimize routing performance, a sleep scheduling algorithm should seek to maintain a uniform degree distribution around an appropriate mean value. Moreover, because the network is deployed for sensing activities such as sample collection, event detection or tracking, the network should also be connected and span the deployment space on every epoch. Satisfying all these requirements with a decentralized sleep scheduling algorithm is challenging.

In addressing the above questions, we make the following contributions. First, we extend existing geographic routing algorithms to handle the TVC networks resulting from duty-cycling.

Second, we provide the first formal analysis of the performance of geographic routing on duty-cycled nodes. Specifically, we analyze the expected increase in routing latency as the number of awake neighbors decreases. Our results can be used as tools to select sleep scheduling parameters that achieve a desired routing performance, or to predict routing performance for a particular parameter setting.

Third, we provide a scheduling algorithm that can be tuned to achieve a certain routing performance. For that, we consider the problem of minimizing the number of awake nodes while maintaining at all times (i) a small target number of awake neighbors for each awake node and (ii) network coverage of the deployment. A network with these two properties shows significantly better routing performance than networks generated by existing sleep scheduling algorithms. We show that the problem is NP-Complete and present a decentralized sleep scheduling algorithm that achieves (i) and (ii) using a number of awake nodes that is guaranteed, with high probability, to be within a logarithmic factor of optimal.

Finally, we evaluate via simulation the performance of our approach versus running existing geographic routing algorithms on sensors duty-cycled according to previous sleep scheduling algorithms. Our results show, perhaps surprisingly, that a network of duty-cycled nodes can have slightly better routing performance than a static network that uses comparable energy (e.g., 500 randomly awake nodes per epoch in random locations outperforms 500 always-on nodes in random locations). Our results further show that, compared to previous algorithms, our sleep scheduling algorithm significantly improves routing latency and network lifetime.

In the rest of the paper, we first present background and related work in Section 2. Section 3 presents geographic routing over TVC networks. Section 4 presents our sleep scheduling algorithm. In Section 5 we formally analyze the interaction between routing latency and sleep scheduling. Section 6 presents several important optimizations and Section 7 presents the evaluation results.

2. BACKGROUND AND RELATED WORK

This section discusses related work on sleep scheduling and geographic routing in wireless sensor networks. The traditional approach to designing routing protocols for sensor networks has been to decouple these two components: duty-cycling is typically done at the MAC layer while routing is done by the routing layer.

2.1 Sleep Scheduling

A sleep scheduler selects a subset of nodes to remain awake in a given epoch, placing the remaining nodes in a minimal power sleep state. The subset of awake nodes may change from epoch to epoch, in order to increase network lifetime by distributing the sensing, processing, and routing load across all the nodes in the network.

Existing works on sleep scheduling aim to achieve *point coverage* and/or *node coverage*. In point coverage (also called *spatial coverage*), the set of awake nodes in an epoch are chosen so that every point of the deployment space is covered; this enables fast and reliable detection of any event in the space. Existing point coverage algorithms differ in their goals of minimizing energy consumption [8, 24], minimizing average event detection latency [3], achieving good detection quality [1], etc. In this paper, we focus on improving routing performance—point coverage is considered only if requested by the application.

In node coverage (also called *network coverage*) algorithms, awake nodes are chosen so that (1) they construct a *connected backbone* and (2) sleeping nodes are immediate neighbors of at least one awake node [4, 23, 27]. The goal of such coverage is to ensure that any two nodes in the network can communicate with each other through the connected backbone. However, there are no requirements/guarantees on the routing performance (beyond maintaining connectivity). We aim to provide good routing performance between any two (awake) nodes in the network, and therefore, our requirement is stronger than that of node coverage algorithms.

A few recent works have addressed both point and node coverage [26], again without routing performance guarantees. A special case of the approach we present provides a probabilistic guarantee of point coverage, in addition to node coverage.

2.2 Geographic Routing

Geographic routing algorithms use node position information to forward a message to its destination. Such algorithms use small (constant-size) per-node state and scale very well for point-to-point communication in wireless networks.

[1]Similar results have been shown analytically and experimentally with more realistic communication models [10].

Geographic routing algorithms use *greedy routing* where possible. In greedy routing, each message is stamped with the coordinates of its destination, all nodes know their own coordinates, and a node forwards the message to the neighbor that is geographically closest to the destination. Local minima may exist where no neighbor is closer to the destination. In such cases, greedy forwarding fails, and another backup strategy must be used to continue making progress toward the destination. The early proposals for geographic routing did not have any such backup strategy, and therefore could not guarantee message delivery [23].

The first geographic routing algorithm to provide guaranteed delivery in any connected network was *face routing* [11]. It uses geometric rules to route around voids near local minima. Face routing and its variants require that the network graph is first converted to a planar graph by using a suitable planarization algorithm [25] or that problematic *cross links* are removed from the network as needed [12, 13]. Recently proposed *hull routing* [14] does not require the network graph to be planar; it uses predefined spanning trees to route messages when they end up at local minima.

Opportunistic routing protocols [2, 5, 29] extend geographic routing by dynamically choosing the forwarding node based on the best node that heard the transmitted message. These protocols typically consider link uncertainty, and adapt routing accordingly. Our approach is complimentary to this; we consider node uncertainty due to duty-cycling. Moreover, we try to adapt the sleep scheduling so that opportunistic routing performs well over the resulting set of awake nodes. The existing literature does not provide any formal analysis of the interaction between the performance of opportunistic routing and the underlying sleep scheduling algorithm.

3. GEOGRAPHIC ROUTING ON A TVC NETWORK

Throughout the paper, we make the following assumptions. Each node knows its geographic location within the deployment area. Nodes are loosely time synchronized. Time is divided into discrete *epochs* such that on every epoch each node, according to some decentralized scheduling protocol, decides to wake up or to sleep for the duration of the epoch. A node can participate in sensing, processing, and communication only when it is awake. A node can communicate only with its awake neighbors, and thus the communication graph of the network changes from epoch to epoch: the network has *time-varying connectivity* (TVC).

Routing on a TVC network mostly resembles routing on a static network. The algorithm first tries to route a message greedily: a message at x is forwarded to the awake neighbor y who, among all x's awake neighbors, is the closest to the destination. However, a TVC network raises a few challenges that must be addressed in the routing algorithm.

3.1 Choosing the Next Hop Node

Consider a message at node x for final destination d. Transmission from node x may be received by many neighbors; however, only one node y (e.g., the one closest to d) forwards the message while the rest of the nodes simply ignore the message. In a static network, x's awake neighbors remain the same over time, and hence x can determine the best next hop node y from its neighborhood information (set of neighbors and their locations). Thus, x can stamp the message with address y so that nodes other than y can ignore it.

However, in a TVC network, x may not know the current best next hop node y, because the subset of awake neighbors changes over time. Therefore, x may not be able to put any specific next hop address in the message. Special techniques are required to avoid multiple neighbors transmitting the same message. ExOR [2] uses a light-weight consensus protocol to determine a single transmitting node. To avoid this overhead, we can use an optimization described in Section 6.1 that enables node x to locally decide which of its neighbors are awake in an epoch; therefore, node x can determine the best next hop awake node y and put it in the message. Nodes other than y who receive the message simply ignore it.

3.2 (Transient) Local Minima

During the routing, a message may end up in a local minimum node x in which all its neighbors, asleep or awake, are further than x from the destination. As discussed in Section 2.2, existing solutions to route out of local minima include face routing [11, 12, 13] and hull routing [14]. We use hull routing in our experiments because of its simplicity and robustness. During this hull routing phase, a node is forced to buffer the message until the next hop node (indicated by the hull routing rule) wakes up.

With TVC networks, a message may also end up in a *transient* local minimum: x is a local minimum (among the awake nodes) in this epoch but not in all epochs (recall Figure 1). The message can either (1) be buffered at x until a better neighbor wakes up, or (2) be forwarded to a best awake neighbor y despite sending the message in the *wrong* direction. We say such routing decisions are *suboptimal*. Our evaluation shows that the second "hot potato" approach, with a good scheduling protocol, shows comparable latency to the first approach, although the second approach slightly increases the total number of hops to the destination. Intuitively, the scheduling protocol ensures that such suboptimal decisions do not happen often, and even if a message is forwarded to a suboptimal next hop, subsequent hops make progress toward the destination with high probability. Moreover, with the hot potato approach, nodes do not have to buffer messages (which can cause long latency and message drop because of the limited buffer space) and nodes with high degree suffer less from becoming routing hot spots and running out of energy. Therefore, we consider the hot potato approach in the rest of the paper.

3.3 Suboptimal Decisions with High Penalty

If the deployment is irregular with large obstacles or voids, a suboptimal decision during hot potato routing may incur a high penalty. Figure 1 gives an example if C wakes up soon after B chooses E as its next hop. In general, a node does not know the topology beyond its local neighborhood (e.g., beyond its 1-hop and possibly 2-hop neighbors), in order to keep its routing state minimal and because the network has TVC. Hence the node can not a priori distinguish between a long detour and a short one (e.g., if D were only 2 hops from E). Because it may happen frequently in TVC networks, we developed the following simple backtracking strategy to limit the "damage": if the number of subsequent suboptimal decisions crosses a threshold, we fall back to hull routing. Our evaluation shows that when used in conjunction with a good scheduling protocol, this strategy is quite effective.

4. A SLEEP SCHEDULER TUNED FOR GEOGRAPHIC ROUTING

In this section, we develop a sleep scheduling algorithm that can be tuned to achieve a target routing performance. As we will show in Section 5, the latency of geographic routing in a TVC network can be controlled by tuning the minimum number, k, of awake neighbors in an epoch for *all* nodes in the network. Thus, our algorithm enables tuning the value of k, in order to achieve a target routing performance.

Our desired sleep scheduling algorithm will have the following properties. First, each node u with d_u neighbors in total must have at least $\min(k, d_u)$ awake neighbors in each epoch. Although our algorithm will be effective in a wide variety of scenarios, we primarily focus on dense deployments where only a small fraction of the nodes are awake each epoch; in such scenarios, nearly all the nodes have $d_u \gg k$. Second, the algorithm will minimize the average number of awake nodes per epoch. In particular, all the nodes will have roughly the same number of awake neighbors (the distribution will be sharply concentrated just above k), a desirable property not ensured by existing algorithms. Third, the algorithm will guarantee node coverage—all awake nodes will be connected and every node (awake or not) will have an awake neighbor. Finally, the set of awake nodes will change from epoch to epoch.

These properties imply that a node will have neither too few (to hurt routing performance) nor too many (to make routing load skewed) awake neighbors. Another consequence is that a higher fraction of the nodes in sparser regions must be kept awake compared to denser regions. This is in contrast to purely random sleep scheduling. This consequence appears unavoidable if one wants to ensure good routing properties through sparse regions.

We also intend each sleeping node u to have not just 1 awake neighbor (as is guaranteed by node connectivity) but at least k of them. We want this for several reasons. First, if u decides to wake up to forward a time-critical message, it will have at least k awake neighbors. Second, the redundancy of awake neighbors increases the probability of point coverage, i.e., the sensing area of u is covered by awake neighbors when u is sleeping.

4.1 Connected k-Neighborhood Problem

We now formalize the problem of choosing awake nodes such that they satisfy the above mentioned properties.

The Connected k-Neighborhood (CKN) Problem: Given a constant k and an undirected graph $G = (V, E)$, find a subset of nodes $C \subseteq V$ such that C is a minimum *connected k-neighborhood.* In a connected k-neighborhood (CKN), (i) each node $v \in V$ has at least $m = \min(k, d_v)$ neighbors from C, where d_v is the degree of v in the G, and (ii) the nodes in C are connected. C is a minimum CKN if no CKN has a smaller number of nodes.

Our desired sleep scheduling algorithm computes a random CKN instance C every epoch such that only the nodes in C wake up; the nodes not in C go to sleep. Note that even a sleeping node u has at least $\min(k, d_u)$ neighbors in C.

Note that the CKN problem is different from the k-connectivity problem, which is well studied in the context of fault-tolerant ad-hoc wireless networks [16, 19]. While CKN requires a local property of k neighbors per node, k-connectivity requires a global property of k vertex-disjoint paths between any two nodes. From a previous study [22], we observe that when $k \leq 10$, a CKN is approximately $(k/2)$-connected.

The CKN Problem is NP-Complete. When $k = 1$, the CKN problem reduces to the Minimum Connected Dominating Set (MCDS) problem [21], a widely studied problem in the ad hoc wireless network community (e.g., an MCDS can be used as a connected virtual backbone for a broadcast process). It is known that the MCDS problem is NP-Complete [9], implying that the CKN problem is also NP-complete.

In the next section we develop an approximation algorithm that provides a near-optimal solution to the CKN problem. Moreover, our algorithm is distributed and incurs modest communication, computation, and memory costs (particularly when using the optimizations presented in Section 6).

4.2 Our CKN Algorithm

A scalable distributed solution to the CKN problem is challenging for several reasons. First, a node can go to sleep assuming that at least k of its neighbors will remain awake to keep it k-connected; however, all the neighbors can think alike and go to sleep, making the node disconnected. A consensus is needed among the nodes to decide who goes to sleep and who remains awake. Second, the outcome of the consensus must change over epochs, so that all nodes have an opportunity to sleep. Finally, even though nodes decide to sleep or wake up based on their local information, the whole network must be globally connected.

We address these challenges by using randomized node ranks. Each node maintains a few local invariants based on its rank, addressing the first and third challenges above. The ranks are assigned randomly on each epoch, addressing the second challenge.

ALGORITHM 1. CONNECTED K-NEIGHBORHOOD (CKN)
(Run the following at each node u *)*

1. Pick a random rank $rank_u$.
2. Broadcast $rank_u$ and receive the ranks of its currently awake neighbors N_u. Let R_u be the set of these ranks.
3. Broadcast R_u and receive R_v from each $v \in N_u$.
4. If $|N_u| < k$ or $|N_v| < k$ for any $v \in N_u$, remain awake. Return.
5. Compute $C_u = \{v | v \in N_u \text{ and } rank_v < rank_u\}$
6. Go to sleep if both the following conditions hold. Remain awake otherwise.
   - Any two nodes in C_u are connected either directly themselves or indirectly through nodes within u's 2-hop neighborhood that have $rank$ less than $rank_u$.
   - Any node in N_u has at least k neighbors from C_u.
7. Return.

The pseudo-code above depicts Algorithm CKN, our sleep scheduling algorithm. The algorithm takes an input parameter k, the required minimum number of awake neighbors per node. The parameter can be chosen depending on the target routing performance (see Section 5). The algorithm is repeated at each scheduling epoch, which may or may not be the same as the epoch used for routing.

In the algorithm, a node u first picks a random rank $rank_u$ (*e.g.*, from a random number generator, in Step 1 of the algorithm) and computes a subset C_u of neighbors having rank $< rank_u$ (Step 5). Before node u can go to sleep it needs to make sure that all nodes in C_u are connected by nodes with rank $< rank_u$ and each of its neighbors has at least k neighbors from C_u (Step 6). These invariants ensure that if a node has less than k neighbors, none of its neighbors goes to sleep and if it has more than k neighbors, at least k of them decide to remain awake. Note that these invariants are easy to compute locally with 2-hop neighborhood information. The needed ranks are exchanged in Steps 2 and 3. Moreover, because the ranks are computed randomly on each scheduling epoch, the set of awake nodes changes from epoch to epoch.

4.3 Analysis of Algorithm CKN

The next three theorems show the correctness and the performance of our algorithm.

THEOREM 1. *Suppose a node u has d_u neighbors in the original network. After running Algorithm* CKN*, u will have at least $\min(k, d_u)$ awake neighbors.*

Proof: If $d_u < k$, none of u's neighbors can go to sleep (Step 4 of the algorithm) and it will have d_u awake neighbors.

If $d_u \geq k$, then we will show by contradiction that the k lowest ranked neighbors of u all remain awake after running the algorithm, and hence, u has at least k awake neighbors. Accordingly, suppose that the i'th lowest ranked neighbor v of u, $i \leq k$, decides to sleep. Then C_v will have at most $i-1$ nodes that are neighbors of u. Since $i-1 < k$, v can not go to sleep according to the algorithm, a contradiction. □

THEOREM 2. *Running Algorithm* CKN *on a connected network produces a connected network.*

Proof: (By contradiction) Suppose that the output network is disconnected. Put the deleted nodes back in the graph in ascending order of their ranks, and let u be the first node that makes the network connected. Note that by the time we put u back, all the members of C_u are already present. Moreover, nodes in C_u are already connected since they are connected by nodes with rank $< rank_u$. Let v be a node that was disconnected from C_u but gets connected to C_u by u. But this contradicts the fact that u can sleep only if all its neighbors (including v) are connected to $\geq k$ nodes in C_u. □

Thus, Algorithm CKN outputs a connected k-neighborhood. Let CKN_k be the set of awake nodes output by the algorithm for a given k and let OPT_k be the set of awake nodes output by an optimal algorithm that finds a minimum connected k-neighborhood. Our final theorem shows that with high probability, the number of nodes in CKN_k is within a logarithmic factor of the number of nodes in OPT_k, for sufficiently dense random deployments. This theorem assumes the *disk-r communication model*, in which a node can communicate precisely with the nodes within distance r, for a suitable choice of r.[2]

THEOREM 3. *For any $k \geq 1$, suppose n nodes are placed uniformly at random within a deployment area such that the average number of neighbors per node (assuming the disk-r communication model) is $\geq 4(k + \ln n)$. Then, with high probability, $|CKN_k| = O(\ln n) \cdot |OPT_k|$.*

Proof: The challenge in this proof is to find an upper bound on $|CKN_k|$ and a lower bound on $|OPT_k|$ that are within a small factor of one another with high probability (w.h.p.). We will make no assumptions on the criteria used to select nodes for OPT_k. Moreover, it is not clear how the local properties used in Algorithm CKN translate into a global bound on $|CKN_k|$.

Let G be the graph of all the nodes and let d be the average node degree in G. By Chernoff bounds, w.h.p., all nodes in G have degree between $d/4$ ($\geq k + \ln n$) and $4d$. Because we are only aiming for a w.h.p. result, we can safely ignore the scenarios where some node has fewer than $d/4$ neighbors or more than $4d$ neighbors.

We begin by lower bounding $|OPT_k|$. Consider running the optimal algorithm on G, and let G' be the graph induced from G by removing all the edges between sleeping nodes. Because each node in G' is required to have at least k neighbors, the total number of edges in G' is $\geq nk/2$. Moreover, because each node in G' has at most $4d$ neighbors, the total number of edges in G' is $\leq 4d \cdot |OPT_k|$. Hence, $4d \cdot |OPT_k| \geq nk/2$, i.e., $|OPT_k| \geq nk/(8d)$.

Next, we upper bound $|CKN_k|$. Let $t = (ckn \ln n)/d$, for a constant $c > 96$ determined by the analysis. (We have not attempted to minimize this constant.) Consider running Algorithm CKN on G and let $rank^*$ be the rank of the t'th smallest rank selected by a node in G. We claim that, w.h.p., all nodes with ranks $> rank^*$ go to sleep. (Some nodes with smaller ranks will also go to sleep, but this claim suffices to show our desired upper bound.) Because there are at most t nodes with rank at most $rank^*$, we have that $|CKN_k| \leq t$ w.h.p.

Note that this claim will enable us to prove the theorem. We have that $|CKN_k| \leq (ckn \ln n)/d = (8c \ln n) \cdot nk/(8d) \leq (8c \ln n) \cdot |OPT_k|$. Thus, $|CKN_k| = O(\ln n) \cdot |OPT_k|$ w.h.p.

We finish by proving the claim. We can view the process as one of first selecting the node ranks and then randomly placing the nodes. Consider a node u not ranked in the top t and let B_u be the ball of radius r around u. Because the average degree is d, we have that each node is placed in B_u with probability d/n. Let C_u be as defined in Algorithm CKN and let C'_u be the nodes in C_u whose ranks are in the top t. Then, $|C'_u|$ is distributed according to a Binomial$(t, d/n)$ distribution. By Chernoff bounds, w.h.p., there are at least $x = td/(4n) = (ck \ln n)/4$ randomly placed nodes in B_u that are ranked in the top t (and hence ranked less than $rank_u$).

Now consider any two nodes v and w in C_u, and let B_v and B_w be the ball of radius r around v and w, respectively. Also, let $B_{u/2}$ be the ball of radius $r/2$ around u; note that any two nodes in $B_{u/2}$ are neighbors. We observe that because v and w are in B_u, the area of $B_v \cap B_{u/2}$ and of $B_w \cap B_{u/2}$ are each at least 1/12 of the area of B_u. Given that $|C'_u| \geq x$ is logarithmic, one can readily show that, w.h.p., at least one node v' (w') from C'_u falls in $B_v \cap B_{u/2}$ (in $B_w \cap B_{u/2}$, respectively). Moreover, v and w are connected by the 3-hop path through v' and w'. Similarly, the expected number of nodes in C'_u that fall in $B_v \cap B_u$ is at least $x/3$. Hence, by Chernoff bounds, w.h.p. any node v in N_u has at least k neighbors from C'_u. Thus, all the conditions for node u to put itself to sleep are satisfied w.h.p., and the claim follows. □

Note that we used the density and disk-r communication model assumptions only in the optimality proof; the correctness proofs are independent of the density and the communication model.

5. ANALYSIS OF GEOGRAPHIC ROUTING ON A TVC NETWORK

In this section, we formally analyze the latency of geographic routing as a function of the number of awake neighbors, k, per node. Intuitively, routing latency decreases as k increases (because each node has more choices when selecting the awake neighbor closest to the destination), but by how much? Answering this question will enable us to tailor a sleep scheduling algorithm to achieve a target latency.

For simplicity, we analyze only the greedy forwarding component of a geographic routing algorithm. Evaluation of geographic routing shows that on average more than 90% of its routing hops use greedy forwarding [14]. Moreover, our analysis considers the idealized setting of uniformly random node placement and the disk-r communication model. These simplifications are needed only for our analysis; our simulation study will show that our techniques are effective even when considering all the aspects described in Section 3 and even in non-uniform settings with obstacles, more realistic radio models, etc.

5.1 An Underlying Markov Process

We begin by defining and analyzing a Markov process that underlies our analysis of geographic routing latency. This process is defined by specifying transition probabilities for making various discrete amounts of progress toward the destination at each step. Later, in Section 5.2, we will analyze how the number of awake neighbors k relates to these transition probabilities.

[2]To simplify the proof, we ignore the "boundary effects" of nodes whose communication disk extends outside the deployment area.

THEOREM 4. *Consider a randomized routing algorithm that at each round, moves one hop closer to the destination with probability p and moves one hop farther from the destination with probability q, where $p > q$. Then the expected number of rounds to reach the destination is $n/(p-q)$, where n is the minimum distance in hops to the destination.*

Proof: Consider a random walk starting from a node at hop distance n from the destination. By the definition of hop distance, the neighbors of a node at distance $i > 0$ are either at distance $i+1$, i, or $i-1$. Let $E(n)$ be the expected number of rounds to reach the destination from hop distance $n \geq 0$. Then, $E[0] = 0$ and, for $n > 0$,

$$E(n) = 1 + pE(n-1) + qE(n+1) + (1-p-q)E(n) \quad (1)$$

Solving this equation yields $E(n) = n/(p-q)$ (which can be readily verified by plugging this solution into Equation 1). □

We will now generalize the process defined in Theorem 4 to enable analyzing progress in Euclidean distance, not hop count. Accordingly, in the following theorems, we use a discretized measure of progress that divides the range of possible forward progress for one hop into $t \geq 2$ equally-spaced segments. (The larger the parameter t, the tighter the analysis.) The general recurrence defined will be, after some manipulation of terms, a natural generalization of Equation 1 and will be solved similarly.

THEOREM 5. **(upper bound)** *Consider a randomized routing algorithm that at each round, (1) with probability q moves farther from the destination, (2) with probability p_i, $i = 0, \ldots, t-2$, moves at least $r\frac{i}{t}$ but less than $r\frac{i+1}{t}$ closer to the destination, and (3) with probability p_{t-1} moves at least $r\frac{t-1}{t}$ closer to the destination, where $t \geq 2$ and $\sum_{i=1}^{t-1} i \cdot p_i > q \cdot t$. Then the expected number of rounds to reach within distance r from the destination is at most $\frac{D}{r} \cdot \frac{1}{(\frac{1}{t}\sum_{i=1}^{t-1} ip_i) - q}$, where D is the Euclidean distance to the destination.*

We assume that once the algorithm is within r of the destination, it simply waits until the destination wakes up and then hops directly to the destination.

Proof: Consider a random walk starting from a node at distance D from the destination. For our upper bound, we credit the algorithm with only the minimal progress in each progress range. Let $E(d)$ be the expected number of rounds to reach within r of the destination starting at a Euclidean distance $d \geq 0$. Then, for $d \leq r$, $E[d] = 0$, and for $d > r$,

$$E(d) \leq 1 + \sum_{i=0}^{t-1} p_i E(d - \frac{i}{t} \cdot r) + qE(d+r) \quad (2)$$

Let $n = d/r > 1$. Then by dividing all terms by r, Equation 2 becomes $E(n) \leq 1/r + \sum_{i=0}^{t-1} p_i E(n - \frac{i}{t}) + qE(n+1)$. As in the solution to Equation 1, $E(n)$ will be of the form n/x for some x. To solve for x, we plug into the previous equation and multiply through by x to get $n \leq x/r + \sum_{i=0}^{t-1} p_i(n - \frac{i}{t}) + q(n+1)$. Moving all the terms involving n to the left side, we get $(1 - \sum_{i=0}^{t-1} p_i + q)n \leq x/r - \sum_{i=0}^{t-1} p_i \frac{i}{t} + q$. Because we have accounted for all possible outcomes at a round in defining q and the p_i's, they sum to 1. Thus the left side equals 0. Solving for x, we get $x \geq r(\sum_{i=0}^{t-1} p_i \frac{i}{t} - q)$. Hence, $E(n) \leq \frac{n}{r(\sum_{i=0}^{t-1} p_i \cdot \frac{i}{t} - q)}$. Letting $N = D/r$, we have that $E(D) = rE(N) \leq N \cdot \frac{1}{(\frac{1}{t}\sum_{i=0}^{t-1} ip_i) - q}$, and the theorem follows. □

Note that $\frac{D}{r}$ is the best case number of rounds, occurring in an idealized deployment scenario with infinite density such that there is a straight line of nodes from the source to the destination, spaced apart at precisely the maximum communication radius r. A more accurate lower bound than the $\frac{D}{r}$ infinite density bound can be obtained by giving the algorithm $r\frac{i+1}{t}$ forward progress credit whenever the true forward progress is in $[r\frac{i}{t}, r\frac{i+1}{t})$ and charging zero backward progress on any true backward progress. This leads to the following theorem:

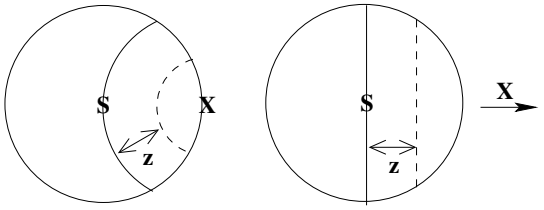


Figure 2: Limiting scenarios for analyzing stretch. The circle around the source S indicates its communication radius. On the left, the destination X is just outside the neighborhood of S. On the right, the arrow next to X indicates that the destination is far from the source, off the end of the figure. In each case, the solid (dotted) line/arc cutting through S's circle indicates the set of points at distance D ($D - z$, respectively) from X, where D is the distance from S to X.

THEOREM 6. **(lower bound)** *For the set-up in Theorem 5, the expected number of rounds to reach within distance r from the destination is at least $\frac{D}{r} \cdot \frac{1}{(\frac{1}{t}\sum_{i=1}^{t-1} ip_i) + \frac{1}{t}(1-q)}$, where $D > r$ is the Euclidean distance to the destination. This can also be written as $\frac{D}{r} \cdot \frac{1}{\frac{1}{t}\sum_{i=0}^{t-1}(i+1)p_i}$.*

Proof: Similar to the proof of Theorem 5. See [18] for details.

5.2 Impact of Duty-Cycling on Latency

We now analyze the latency of greedy geographic routing as a function of the number of awake neighbors. We consider the *relative stretch* of the routes, namely the ratio between $E_k(D)$, the expected number of epochs needed to reach the destination when only k neighbors are awake per node and $E_K(D)$, the expected number of epochs needed to reach the destination when some larger number K neighbors are awake. As in our analysis in Section 5.1, we assume uniformly random node locations and the disk-r communication model. To the best of our knowledge, this is the first such analysis, and provides important insights into how many neighbors are "good enough".

Figure 2 shows the two limiting scenarios, in which the destination X is as close as possible (any closer means we have reached a node S that has X as a neighbor, and we can simply wait for X to wake up) and is as far away as possible. Each awake neighbor can be viewed as a random point in S's disk. Let D be the distance from S to X.

Far-away Scenario. We first analyze the far-away scenario (right). In this scenario, the set of points at distance D from X splits the neighborhood in half—thus each neighbor is equally likely to be closer or farther from X. With k neighbors, the probability that the neighbor closest to X is farther than S is thus 2^{-k}. Thus in the terminology of the previous section, $q = 2^{-k}$.

Let f_i be the probability that a random neighbor of S is at least $\frac{i}{t}r$ closer to the destination X (i.e., falls to the right of the dashed line when $z = \frac{i}{t}r$). By standard formulas for the area of a segment of a circle, we have

$$f_i = \frac{1}{\pi}\left(\cos^{-1}(i/t) - \frac{i}{t}\sqrt{1-(i/t)^2}\right), \quad (3)$$

Let $p_i^{(k)}$ be the probability that, among the k random neighbors of S, the neighbor closest to the destination X is at least $\frac{i}{t}r$ closer but less than $\frac{i+1}{t}r$ closer. (This is the p_i we need to apply the theorems of Section 5.1.) This probability can be calculated as the probability that "no neighbor is at least $\frac{i+1}{t}r$ closer" times the conditional probability that "some neighbor is at least $\frac{i}{t}r$ closer given that no neighbor is at least $\frac{i+1}{t}r$ closer."

$$p_i^{(k)} = (1 - f_{i+1})^k \cdot \left(1 - \left(\frac{1 - f_i}{1 - f_{i+1}}\right)^k\right) \tag{4}$$

The term $\frac{1-f_i}{1-f_{i+1}}$ is the probability that a random neighbor that is *not* at least $\frac{i+1}{t}r$ closer is also *not* at least $\frac{i}{t}r$ closer.

Close Scenario. We now analyze the close scenario (left). By standard formulas for the intersection area of two circles with the same radii r that are centered distance r apart, the probability that a random neighbor of S is closer to X than S (i.e., to the right of the solid arc through S) can be calculated as $\frac{2}{3} - \frac{\sqrt{3}}{2\pi}$. Thus with k random neighbors, the probability that the neighbor closest to X is farther than S is:

$$q = \left(\frac{1}{3} + \frac{\sqrt{3}}{2\pi}\right)^k \approx .609^k \tag{5}$$

As above, let f_i be the probability that a random neighbor of S is at least $\frac{i}{t}r$ closer to the destination X (i.e., falls to the right of the dashed arc when $z = \frac{i}{t}r$). Applying standard formulas for the intersection area of two circles to this particular scenario yields, after simplifying terms,

$$f_i = \frac{1}{\pi}\left(w_i \cos^{-1}\left(\frac{w_i}{2}\right) + \cos^{-1}\left(1 - \frac{w_i}{2}\right) - \frac{1}{2}\sqrt{w_i(4 - w_i)}\right), \tag{6}$$

where $w_i = \left(1 - \frac{i}{t}\right)^2$ and arcosine returns radians.

The probability $p_i^{(k)}$ that, among the k random neighbors of S, the neighbor closest to the destination X is at least $\frac{i}{t}r$ closer but less than $\frac{i+1}{t}r$ closer can be computed by plugging Equation 6 into Equation 4.

Relative Stretch. We are now able to calculate an upper bound on the relative stretch incurred, as a function of the number of awake neighbors k compared to a larger number of neighbors K, for arbitrary values of our discretization parameter t. We use the equations of this section with the theorems of the previous section. To upper bound the stretch, we analyze $E_k(D)$, the expected number of epochs with k neighbors, using (the pessimistic) Theorem 5 and we analyze $E_K(D)$ using (the optimistic) Theorem 6. Section 7.5 will present some example bounds calculated in this manner, and compare them to results obtained via simulation.

6. OPTIMIZATIONS

In this section we present two decentralized optimizations to make Algorithm CKN more efficient. The first optimization removes the communication required for a node to learn the ranks of its 2-hop neighbors in each epoch, at the expense of some local computation. The second optimization reduces the amount of computation.

6.1 Shared Random Number Generator

The basic idea is to use the same deterministic pseudo-random number generator $G(nodeid, time)$ in all nodes in the network. Then we modify Algorithm CKN for node u as follows.

1. Node u discovers N_2, the nodes within two hops in the original network. It can discover N_2 once in the beginning and can update it infrequently in case nodes join or leave the network.
2. Node u generates its rank as $rank_u = G(u, epoch)$, where $epoch$ is the current scheduling epoch number. (Step 1 of Algorithm CKN.)
3. For each node v in N_2, u generates rank $rank_v = G(v, epoch)$. (This eliminates Steps 2 and 3 of Algorithm CKN).

The rest of Algorithm CKN remains the same.

In this modified algorithm, there is *no communication between nodes* (beyond the background maintenance of N_2 and epoch clocks). This advantage comes at a cost—each node needs to maintain states for all its 2-hop neighbors and to generate random numbers for them. We will quantify this cost in Section 7.

6.2 Spatial Sampling

Our second optimization is relevant only for deployments that densely cover their deployment area. The basic idea is first to select a random subset of the nodes and then to run Algorithm CKN only on that subset. Suppose n nodes with transmission radius r are uniformly randomly placed within an area A. Assume that A is at least $2r$ wide at its narrowest point and that the sensors cover every point of the deployment area. Then after running Algorithm 2 (SPATIALSAMPLE), every point still remains covered by at least one awake sensor node with high probability. Moreover, every sampled node maintains a sufficiently large number of neighbors so that Algorithm CKN can be run on the sampled nodes only.

ALGORITHM 2. SPATIALSAMPLE
(Run the following on each node at the start of every epoch *)*

1. Let $N = 2|A|/r^2$ and $p = \min\{(N\ln(N) + cN)/n, 1\}$, c is a constant.
2. With probability p remain awake, otherwise go to sleep.

LEMMA 1. *Suppose n sensors are deployed uniformly at random in an area A such that A is at least $2r$ wide at its narrowest point and every point in A is covered by at least one sensor. Then, after running* SPATIALSAMPLE*, any point in A will remain covered by at least one awake node with probability $1 - e^{-c}$. Moreover, every node will have $\ln(N) + c$ awake neighbors on expectation.*

Proof: Can be proved by results on the Coupon Collector Problem [17]. See [18] for details. □

If SPATIALSAMPLE is used in conjunction with the first optimization, all nodes should use the same generator G in Step 2, and remain awake if its output is at most p.

After running SPATIALSAMPLE, the still awake nodes proceed to run Algorithm CKN. The benefits of running Algorithm CKN on only a sampled set of nodes, however, comes at a potential cost: If the sampled nodes are disconnected or have few neighbors, the network produced by Algorithm CKN will have bad routing performance. So the sampling parameter should be chosen carefully.

7. EVALUATION

In this section, we evaluate the performance of our sleep scheduling algorithm and geographic routing on top of it. We use a custom event-driven simulator for the evaluation.

7.1 Methodology

Evaluation Scenarios. Nodes are randomly placed in a $175m \times 175m$ square. Each message is routed between two random points in the deployment.

Communication Model. We model communication between two nodes according to the measurement results from a dense wireless sensor network deployment in a state park [28]. We primarily model packet loss rates and link asymmetry in our simulation. As reported in [28], the packet loss model has two distinct regimes: up to a distance $12m$ from the sender, packet loss rates are consistently small. Beyond this, however, there exists a *gray area* in which loss rate varies dramatically—some nodes see near 90% successful reception, while neighboring nodes sometimes see less than 50% reception rate.

Scheduling Algorithms. We compare the following five sleep scheduling algorithms. (1) CKN: on every epoch, nodes run Algorithm CKN to decide whether to be awake. (2) POINTCOVER: on every epoch, nodes run an existing point-coverage scheduling algorithm [1] to decide whether to be awake. (3) NODECOVER: on every epoch, nodes run an existing node-coverage scheduling algorithm [4] to decide whether to be awake. Note that comparing POINTCOVER and NODECOVER with CKN is not an apples-to-apples comparison because POINTCOVER and NODECOVER do not aim to optimize routing performance. Moreover, these existing algorithms do not allow tuning the number of awake nodes in the network, something that we intend to vary with CKN. Therefore, we compare CKN with two additional algorithms. (4) RANDOM: on every epoch, nodes independently decide whether to be awake with equal probability. We can vary the number of awake nodes by varying the probability (i.e., with SPATIALSAMPLE). RANDOM is a representative scheduling algorithm used in almost all real sensor systems [7], and in many proposals [8]. (5) ALWAYSON: nodes always remain awake. We can vary the total number of nodes and thus the total amount of energy consumed.

Routing Algorithm. We use the routing algorithm described in Section 3: Nodes use greedy forwarding whenever possible, and hull routing when messages end up in local minima. A message can make 10 hops in an epoch. Nodes use a reliable communication protocol to deliver messages to their next hops.

Evaluation Metrics. We use the following metrics to evaluate the performance of different algorithms. (1) *Normalized Hop (or Epoch) Count:* The normalized hop count (epoch count) of a scheduling algorithm is the ratio between the average hop count (epoch count, resp.) for the algorithm to route a message between two random nodes to that for ALWAYSON. A small value of this metric implies good routing performance. Note that the normalized hop count may be different from the normalized epoch count, as a result of the buffering of messages during the hull routing phase. (2) *Load:* The load of a node is the fraction of random messages routed through the node. Ideally, we prefer the load to be uniform across nodes. To report the above metrics under a configuration, we generate 100 different random networks and route messages between 5000 pairs of random locations.

Unless otherwise specified, our simulated networks are connected when all nodes are awake. However, because a scheduling algorithm may choose only a subset of the nodes to be awake in any given epoch, a network may sometimes become disconnected. A message is sent again by the source if it does not receive the delivery acknowledgement within a specified number of epochs (we use 10 as the threshold). We include the overheads of failed routing in our computation of hop count, epoch count, and load.

7.2 Routing Performance Comparison

To understand the performance of geographic routing over duty-cycled nodes, we route messages over 600-node networks with different node scheduling algorithms. Figure 3(a) and (b) show normalized hop- and epoch-count of different algorithms, as a function of the average number of awake nodes per epoch. NODECOVER and POINTCOVER both have one point each, representing the minimum number of nodes required to ensure node and point coverage respectively. The CKN curve has 6 points representing the results of running CKN(k) with the minimum number of awake neighbors $k = 1, \ldots, 6$. NODECOVER, POINTCOVER, CKN, and RANDOM are run on 600-node networks, while the total number of (awake) nodes in ALWAYSON is same as the number of awake nodes reported for these other algorithms.[3] The x-axis of Figure 3 represents the average number of awake nodes per epoch. The smaller the number, the less energy consumed per epoch.

These results demonstrate several important points. First, an efficient randomized sleep scheduling algorithm can improve routing performance without additional energy overhead, especially at low node densities, as indicated by the points where normalized hop and epoch counts are less than 1. Intuitively, because the connectivity of a TVC network changes over epochs, local minima in one epoch often disappear in next epoch, thus helping greedy routing to succeed most of the time.

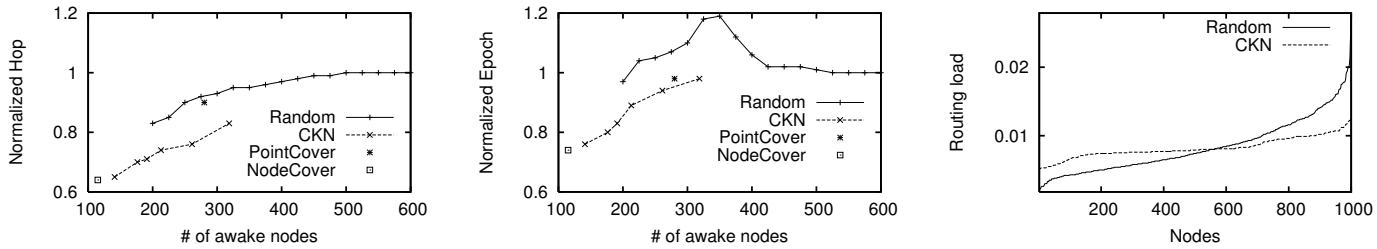
Second, selectively removing nodes from the network to make the degree distribution more uniform (as is done by CKN) outperforms algorithms that randomly removes nodes (e.g., RANDOM). With the same number of awake nodes, CKN(6) performs $\approx 20\%$ better than RANDOM. This is because CKN avoids nodes having too few neighbors, keeping the network connected in every epoch and providing each node with at least k potential next hops. In RANDOM, the network becomes disconnected in some epochs which results in message buffering and timeouts. In fact, with < 300 awake nodes, messages had to be buffered $> 5\%$ of the time with RANDOM, while CKN(6) required no buffering. The graphs also show that, to achieve the same routing performance, CKN requires $> 40\%$ fewer awake nodes per epoch compared to RANDOM and ALWAYSON. This implies that CKN can improve network lifetime over RANDOM and ALWAYSON. This happens because CKN avoids having too many awake neighbors, which gives more nodes the opportunity to sleep in a given epoch.

Finally, CKN is comparable to NODECOVER under a small number of awake nodes. As expected, POINTCOVER is not as good as CKN because POINTCOVER chooses awake nodes to ensure point coverage, not to optimize routing performance.

7.3 Routing Load

In this experiment, we investigate the routing load distribution of two different sleep scheduling algorithms. We route messages with CKN(6) and RANDOM in a 1000-node network. The parameters of RANDOM are chosen so that the expected number of awake nodes remains the same as CKN(6). The x-axis of Figure 3(c) shows all the 1000 nodes in ascending order of their routing load (i.e., fraction of times a node forwards messages), and the y-axis shows their corresponding load. As shown, RANDOM has a more skewed distribution than CKN. This can cause some nodes to run out of energy faster and the network to become disconnected. The differences in load distribution of CKN and RANDOM can be explained

[3]Because ALWAYSON uses the same number of *awake* nodes as the other scheduling algorithms, normalized hop (or epoch) counts can be less than 1. For example, CKN on a 600-node network where 300 random nodes remain awake in each epoch can perform better than ALWAYSON on a fixed 300-node network.



(a) Normalized Hop Counts (b) Normalized Epoch Counts (c) Load Distribution

Figure 3: Normalized hop- and epoch-counts and load distribution of different algorithms running over a network of 600 nodes.

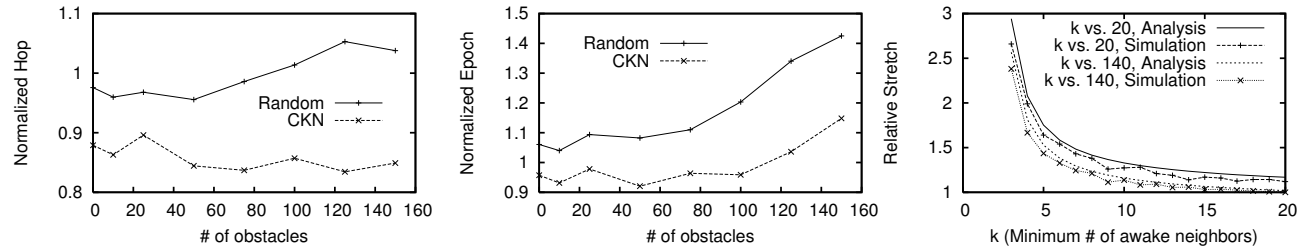


Figure 4: Normalized hop and epoch count with varying number of obstacles. Around 400 nodes remain awake in every epoch in all the algorithms.

Figure 5: Validation of analytical upper bounds on relative stretch.

by observing the number of awake neighbors per node, because the amount of routing load through a node is roughly proportional to the number of awake neighbors. With RANDOM, some nodes have too many awake neighbors while other nodes have too few (experimental validation in Section 7.6). This causes more messages to be routed through the high-degree nodes. In contrast, the output network of CKN has a more uniform degree distribution, which helps the routing load to be more evenly distributed among the nodes.

Like CKN, NODECOVER also demonstrates a roughly uniform load distribution. POINTCOVER, however, has a slightly skewed (more than CKN but less than RANDOM) load distribution. This is because some points in the deployment area are covered by only a few nodes. To ensure point coverage, these nodes remain awake more often, and hence get more routing traffic, than other nodes.

7.4 Effect of Obstacles

To evaluate the effect of obstacles, we randomly place $4m \times 4m$ square obstacles within a 1000-node network. In addition to the communication model mentioned before, two nodes fail to communicate if their line of sight intersects with any obstacle. To compare different algorithms under the same energy cost, we select the number of nodes in the networks such that on each epoch ≈ 400 nodes remain awake.

Figure 4 shows normalized hop and epoch counts of different algorithms, while varying the number of obstacles. As before, CKN outperforms RANDOM, and in fact, the presence of obstacles increases CKN's advantage over RANDOM. This happens because, with obstacles, the network tends to become (temporarily) disconnected more often with RANDOM. This results in frequent message buffering at the nodes and timeouts/retransmissions of messages. All these degrade hop- and epoch-counts with RANDOM.

7.5 Validation of Analytical Results

To validate the analytical upper bounds on relative stretch given in Section 5, we start with a dense network of 5000 nodes. In this network, every node has at least 140 neighbors. We then use CKN(k), $k = 3, \ldots, 20$, and compute the relative stretches of these resultant networks with respect to the original 5000-node network. Our simulation considers more realistic events (e.g., hull routing, retransmission, etc.) than our analysis. However, because these events are rare and make relative stretches only worse, we hope to factor them out in our validation by ignoring the top 10% of the simulation stretches. So, we compare the analytical upper bounds with the $90th$ percentile of the relative stretches obtained from simulations.

In Figure 5, the plots for "k vs. 140" show the analytical upper bounds and $90th$ percentile of relative stretches, with respect to the 5000-node network, from simulation. The graphs for "k vs. 20" considers the network given by CKN(20), instead of the original 5000-node network. As shown, our analytical results are very close (within 10%) of the simulation results. The gap between analysis and simulation is bigger at smaller values of k, because even though CKN ensures every node has at least k neighbors, some nodes have more than k neighbors, resulting in better relative stretches than the analytical bounds that assume exactly k neighbors.

7.6 CKN Characteristics Evaluation

Having shown CKN's routing performance, we now investigate a few of its properties.

Effect of Optimizations. Figure 6 shows the communication, computation, and memory cost per node per epoch for running CKN over a 5000-node network with and without the two optimizations from Section 6. The communication cost is given in terms of the number of broadcasts required by the algorithm. The computation cost reflects only the number of random numbers generated (the other computations of CKN are not affected by the optimizations). The memory cost reflects the number of states a node needs to maintain for its 2-hop neighboring nodes. Using a shared random number generator (shown as Opt 1 in Figure 6) avoids expensive communication but introduces a lot of additional computation and memory overhead. Spatial sampling (Opt 2) can reduce this computational overhead. To keep its effect on CKN's routing performance negligible ($< 2\%$ increase in the normalized hop count),

Scheme	Communi-cation	Compu-tation	Memory
No Opt	2	1	0
Opt 1	0	147	146
Opt 2	0.89	2	0
Opts 1 & 2	0	67	146

Figure 6: Optimized performance of CKN.

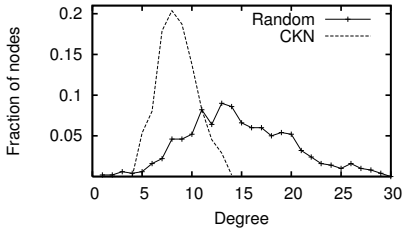


Figure 7: Degree distribution of 500 nodes chosen from 5000 nodes.

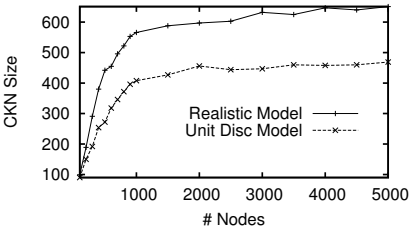


Figure 8: Size of CKN(6).

we set parameters so that around half the nodes run CKN. This reduces communication at the cost of a slight increase in computation. Finally, using both optimizations removes the communication overhead at the cost of a small computational overhead. Because generating a pseudo-random number is cheap (requires multiplication and addition), these optimizations provide a good tradeoff.

Degree Distribution. As mentioned before, a uniform degree distribution improves routing performance. To compare the degree distribution of CKN with that of RANDOM, we run them on a 5000-node network. We use CKN(6) and tune the parameter of RANDOM to produce approximately the same number of awake nodes (≈ 500). Figure 7 shows that the distribution of RANDOM has a wider range than that of CKN. Note that running CKN(6) ensures that every node with degree ≥ 6 has at least 6 awake neighbors. Therefore, the average degree of a node is typically higher than 6. E.g., the outputs of CKN(6) on a 500- and a 5000-node network have average degrees of 7.2 and 7.9, respectively.

Effect of Node Density. To understand how CKN scales, we run CKN(6) with varying node density. Figure 8 shows that increasing the number of nodes increases the number of awake nodes, which effectively reduces routing latency. However, the increase of awake nodes diminishes with node density, and even with 5000 nodes, CKN uses less than 650 awake nodes. This gives around 13% duty cycling. Although our optimality proof for CKN uses the uniform disc communication model, the results in Figure 8 show that even with the more realistic communication model used in our simulation study, CKN uses a small number of awake nodes.

8. CONCLUSION

In this paper, we have formally analyzed the performance of geographic routing over duty-cycled nodes. We have presented a sleep scheduling algorithm that can be tuned to achieve a target routing latency. We have provided analytical guarantees on the correctness and the performance of our algorithm. Through extensive simulation, we have shown that routing over our scheduling algorithm improves routing performance by $> 20\%$ compared to existing scheduling algorithms, without additional energy overhead. Moreover, it can significantly improve network lifetime without sacrificing routing performance. Future work includes studying the performance of our scheduling algorithm in real deployments.

9. REFERENCES

[1] ABRAMS, Z., GOEL, A., AND PLOTKIN, S. Set k-cover algorithms for energy efficient monitoring in wireless sensor networks. In *ACM/IEEE IPSN* (2004).

[2] BISWAS, S., AND MORRIS, R. Opportunistic routing in multi-hop wireless networks. In *ACM SIGCOMM* (2005).

[3] CAO, Q., ABDELZAHER, T., HE, T., AND STANKOVIC, J. Towards optimal sleep scheduling in sensor networks for rare-event detection. In *ACM/IEEE IPSN* (2004).

[4] CHEN, B., JAMIESON, K., BALAKRISHNAN, H., AND MORRIS, R. Span: an energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks. In *ACM MobiCom* (2001).

[5] CHOUDHURY, R. R., AND VAIDYA, N. H. MAC-layer anycasting in ad hoc networks. *ACM SIGCOMM CCR 34*, 75–80 (2004).

[6] DEMIRBAS, M., AND FERHATOSMANOGLU, H. Peer-to-peer spatial queries in sensor networks. In *IEEE P2P* (2003).

[7] DUTTA, P., GRIMMER, M., ARORA, A., BIBYK, S., AND CULLER, D. Design of a wireless sensor network platform for detecting rare, random, and ephemeral events. In *ACM/IEEE IPSN* (2005).

[8] FAN HSIN, C., AND LIU, M. Network coverage using low duty-cycled sensors: random & coordinated sleep algorithms. In *ACM/IEEE IPSN* (2004).

[9] GAREY, M. R., AND JOHNSON, D. S. *Computers and Intractability - A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York, 1979.

[10] HEKMAT, R., AND MIEGHEM, P. V. Degree distribution and hopcount in wireless ad-hoc networks. In *IEEE ICON* (2003).

[11] KARP, B., AND KUNG, H. T. GPSR: Greedy perimeter stateless routing for wireless networks. In *ACM MobiCom* (2000).

[12] KIM, Y.-J., GOVINDAN, R., KARP, B., AND SHENKER, S. Geographic routing made practical. In *Usenix NSDI* (2005).

[13] KIM, Y.-J., GOVINDAN, R., KARP, B., AND SHENKER, S. Lazy cross-link removal for geographic routing. In *ACM SenSys* (2006).

[14] LEONG, B., LISKOV, B., AND MORRIS, R. Geographic routing without planarization. In *Usenix NSDI* (2006).

[15] LI, X., KIM, Y. J., GOVINDAN, R., AND HONG, W. Multi-dimensional range queries in sensor networks. In *ACM SenSys* (2003).

[16] LI, X.-Y., WAN, P.-J., WANG, Y., AND YI, C.-W. Fault tolerant deployment and topology control in wireless networks. In *ACM MobiHoc* (2003).

[17] MITZENMACHER, M., AND UPFAL, E. *Probability and Computing : Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005.

[18] NATH, S., AND GIBBONS, P. B. Communicating via fireflies: Geographic routing on duty-cycled sensors. Tech. Rep. MSR-TR-2007-21, Microsoft Research, 2007.

[19] PENROSE, M. D. On k-connectivity for a geometric random graph. *Wiley Random Structures and Algorithms 15*, 2 (1999), 145–164.

[20] SHENKER, S., RATNASAMY, S., KARP, B., GOVINDAN, R., AND ESTRIN, D. Data-centric storage in sensornets. In *ACM HotNets* (2002).

[21] STOJMENOVIC, I., SEDDIGH, M., AND ZUNIC, J. Dominating sets and neighbor elimination-based broadcasting algorithms in wireless networks. *IEEE Trans. on Parallel and Distributed Systems 13*, 1 (2002), 14–25.

[22] SUN, F., AND SHAYMAN, M. On the average pairwise connectivity of wireless multihop networks. In *IEEE Globecom* (2005).

[23] TAKAGI, H., AND KLEINROCK, L. Optimal transmission ranges for randomly distributed packet radio networks. *IEEE Trans. on Communication 32*, 3 (1984), 246–257.

[24] TIAN, D., AND GEORGANAS, N. D. A coverage-preserving node scheduling scheme for large wireless sensor networks. In *ACM WSNA* (2002).

[25] TOUSSAINT, G. T. The relative neighbourhood graph of a finite planar set. *Pattern Recognition 12* (1980), 261–268.

[26] WANG, X., XING, G., ZHANG, Y., LU, C., PLESS, R., AND GILL, C. Integrated coverage and connectivity configuration in wireless sensor networks. In *ACM SenSys* (2003).

[27] XU, Y., HEIDEMANN, J., AND ESTRIN, D. Geography-informed energy conservation for ad hoc routing. In *ACM MobiCom* (2001).

[28] ZHAO, J., AND GOVINDAN, R. Understanding packet delivery performance in dense wireless sensor networks. In *ACM SenSys* (2003).

[29] ZORZI, M., AND RAO, R. R. Geographic random forwarding (GeRaF) for ad hoc and sensor networks: Energy and latency performance. *IEEE Trans. on Mobile Computing 2*, 4 (2003), 349–365.