Querying Dynamic Wireless Sensor Networks with Non-revisiting Random Walks

Marco Zuniga[1], Chen Avin[2], and Manfred Hauswirth[1]

[1] Digital Enterprise Research Institute
National University of Ireland, Galway
{marco.zuniga,manfred.hauswirth}@deri.org
[2] Department of Communication Systems Engineering
Ben Gurion University of the Negev, Israel
avin@cse.bgu.ac.il

Abstract. The simplicity and low-overhead of random walks have made them a popular querying mechanism for Wireless Sensor Networks. However, most of the related work is of theoretical nature and present two important limitations. First, they are mainly based on simple random walks, where at each step, the next hop is selected uniformly at random among neighbors. This mechanism permits analytical tractability but wastes energy by unnecessarily visiting neighbors that have been visited before. Second, the studies usually assume static graphs which do not consider the impact of link dynamics on the temporal variation of neighborhoods.

In this work we evaluate the querying performance of Non-Revisiting Random Walks (NRWs). At each step, NRWs avoid re-visiting neighbors by selecting the next hop randomly among the neighbors with the minimum number of visits. We evaluated Pull-only and Pull-Push queries with NRWs in two ways: (i) on a test-bed with 102 tmotes and (ii) on a simulation environment considering link unreliability and asymmetry. Our main results show that non-revisiting random walks significantly improve upon simple random walks in terms of querying cost and load balancing, and that the push-pull mechanism is more efficient than the push-only for query resolution.

1 Introduction

Querying has been, and continues to be, one of the most investigated areas in the Wireless Sensor Networks community. For scenarios where nodes have no location information (location-less), querying paradigms can be classified into 2 broad categories: i) random walks [20,5,21] and ii) flooding or controlled flooding (expanding ring searches) [11,12,13].

On flooding, each node (re)transmits the querying packet once. On random walks, nodes are queried in some sequential random order. The walk starts at some fixed node, and at each step it moves to a neighbor of the current node. The random walk is called simple when the next node is chosen uniformly at random from the set of neighbors.

J. Sá Silva, B. Krishnamachari, and F. Boavida (Eds.): EWSN 2010, LNCS 5970, pp. 49–64, 2010.

The main advantage of random walks is its localized search, which avoids the unnecessary use of bandwidth and energy resources utilized by flooding-type techniques [17]. On the other hand, if the data of interest is far away from the sink, the querying cost of random walks can be super-linear in the worst case compared to the linear cost of flooding.

In this work, we investigate a variant of random walks that provides an energy-efficient querying alternative for location-less deployments: Non-Revisiting Random Walks (NRWs)[1]. The motivation behind this work is to derive a querying mechanism that combines the localized behavior of random walks and the linear cost of flooding.

Our work is inspired by the studies presented in [6,22]. These studies identify a important limitation of Simple Random Walks (SRWs): selecting the next node at random is a simple mechanism but leads to frequent revisiting nodes, which in turn leads to long delays and high expenditures of energy. Contrary to the *blind* selection performed by simple random walks, NRW selects the neighbor with the least number of visits. This Non-Revisiting mechanism maximizes the likelihood of encountering unvisited nodes, and hence, accelerates the discovering process.

Our work focuses on two types of querying scenarios: (i) Pull-Only querying and (ii) Push-Pull querying. In Pull-Only querying the sink starts a walk looking for the event. In Push-Pull querying, both, the event and sink nodes start walks and query is solved when the walks intersect.

We evaluated the performance of SRW and NRW on TWIST [3], an in-building test-bed with 102 tmotes, and we simulated larger networks using a probabilistic link model for the channel [28]. Our results provide two important contributions. First, it illustrates the difficulties faced by random walks on real deployments due to the high temporal dynamics of links. We show that polling the neighborhood immediately before transferring the token is an efficient mechanism to cope with these dynamics. Second, our results indicate that NRW, together with the simple push-pull mechanism, is an efficient querying mechanism for networks consisting of up-to thousands of nodes. NRWs with Push-Pull querying maintains the elegance of simple random walks, while at the same time provide querying costs that are linear or sub-linear (depending on the size of the network).

2 Definitions, Implementation and Metrics

First, let us present the precise definitions of the random walks types and the querying mechanisms evaluated on our work:

Definition 1 (Simple Random Walk (SRW)). *The walk starts at an initial node and at each step selects one of its neighbors uniformly at random.*

Definition 2 (Non-Revisiting Random Walk (NRW)). *The walk starts at an initial node and at each step selects the neighbor with the minimum number of*

[1] There are similar types of walks in the related literature (i.e., self-avoiding walks [15], Vertex- Reinforced Random Walks [18]), but to the best of our knowledge NRWs where not considered explicitly before.

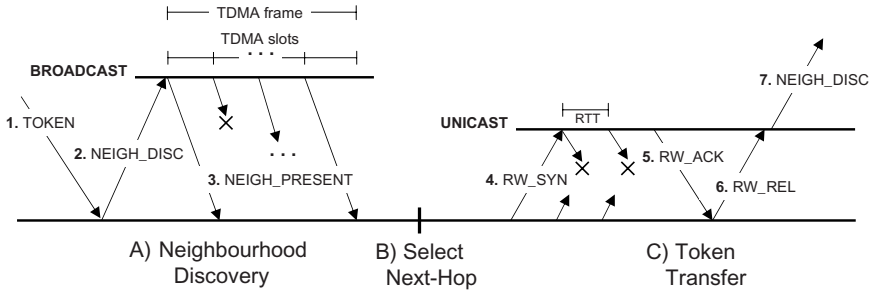


Fig. 1. Protocol Implementation of Random Walk

visits (which could be 0). If more than one node have the same minimum number of visits, the next node is selected uniformly at random among these nodes.

We consider two mechanisms for query resolution. In both of them the *event-node* has some data of interest, and the *sink-node* issue a query to find that piece of data.

Definition 3 (Pull-Only Querying). *The data remains on the event-node and only the sink-node starts a random walk (i.e., pull). The query is solved when the walk reaches the event node.*

In the push-pull case the event-node *publishes* its data.

Definition 4 (Push-Pull Querying). *The event-node starts a random walk to publish its data of interest (i.e., push). The sink-node starts a random walk based query (i.e., pull). The query is solved when the paths of the walks intersect.*

We do not discuss here the way the data of interest is routed back to the sink after query resolution, but this could be done for example by using a trace left by the query walk. For the reminder of the paper, the term *token* is used to denote the *presence* of the walk on a node.

2.1 Walk Implementation

Contrary to theoretical studies, where the neighborhood of a node is assumed to remain constant, in real scenarios, link dynamics such as asymmetry, unreliability and temporal variation pose significant challenges to the robust dissemination of the walk. In order to cope with these dynamics, our implementation of a random walk utilizes the following three procedures: (a) Neighborhood Discovery, (b) Selection of Next-Hop and (c) Transferring of Token. These procedures are presented in Figure 1.

Neighborhood Discovery. Upon reception of the token, a node broadcasts a `NEIGH_DISCOVERY` message. Nodes within the transmission range of the sender reply with `NEIGH_PRESENT` messages. In order to avoid collisions caused by the

concurrent transmission of `NEIGH_PRESENT` messages, we implemented a MAC TDMA scheme. In this TDMA scheme, nodes are assigned different transmission slots based on their *id*.

Selection of Next Hop. The token-holder waits until the end of the TDMA frame and selects the next node among the received `NEIGH_PRESENT` messages. Depending on the type of walk to be performed, the selection follows the guidelines presented in Definitions 1 and 2.

Transferring of Token. This procedure is similar to the 3-way handshake mechanism utilized in the TCP protocol. The token-holder sends an initial `RW_SYN` packet to communicate a node that it has been selected as the next step. Upon reception of a `RW_SYN`, the receiver sends a `RW_ACK` packet. Finally, the sender completes the transfer by sending a `RW_REL` packet. In order to cope with packet losses, `RW_SYNs` and `RW_ACKs` are sent every `RTT` (round trip time). The sender stops transmitting `RW_SYNs` after receiving a `RW_ACK`, and the receiver stops transmitting `RW_ACKs` after receiving a `RW_REL`. `RW_REL` packets are sent only upon reception of a `RW_ACK`.

2.2 Metrics

In this subsection we present the metrics used to quantify the performance of SRW and NRW. Let us denote G_n as the communication graph formed by a network of n nodes and s as the number of steps performed by a random walk. Based on this notation, a simple random walk performing s steps on graph G_n is denoted by $SRW(G_n, s)$, and a non-revisiting random walk is denoted by $NRW(G_n, s)$.

Once a random walk starts, each node $u \in G_n$ stores locally the following information:

- T_u^{min}: time of first visit.
- T_u^{max}: time of last visit.
- V_u: number of visits.

In our work, the time t is represented by the number of steps. For example, a node u that is visited for the first time at the k^{th} step of the walk will have an entry $T_u^{min} = k$.

Two important properties of random walks are directly related to querying [16]: (i) cover time and (ii) hitting time. The cover time $C_u(G_n)$ is the expected number of steps for a walk starting at u to visit all the nodes in graph G_n. The *partial cover time* $C_u(G_n, f)$ is the expected number of steps for a walk starting at u to first visit a fraction f of the graph G_n. The hitting time h_{uw} is the expected time taken by a walk starting at u to reach w for the first time. In this paper we evaluate the *average* hitting time $H_u(G_n)$ from a sink u which is given by:

$$H_u(G_n) = \frac{\sum_{w \in G_n} h_{uw}}{n-1} \tag{1}$$

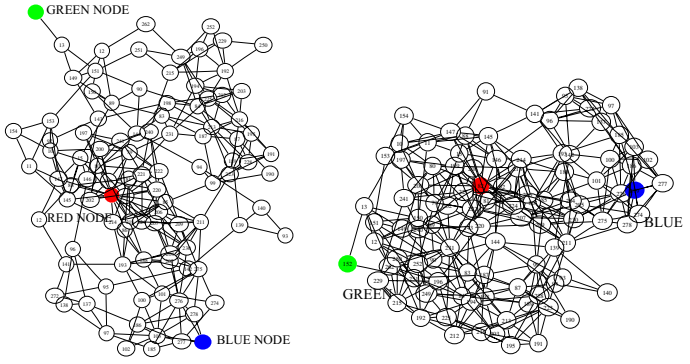


Fig. 2. Communication Graph of TWIST. (a) shows links with *transmission probabilities* greater than 0.9 and (b) greater than 0.7. Three nodes were selected to inject the random walks (green, red, blue).

Hence, for Pull-Only querying, cover time and hitting time translate to the worst-case and average-case querying scenarios[2]. Another important property of random walks is load balancing. Given the limited energy resources of WSN, it is desirable that the walk visits the network evenly without over-stressing some nodes by visiting them more frequently. For a starting node u, we measure the load balancing as the difference between $B_u^{\max}(G_n, s)$ and $B_u^{\min}(G_n, s)$, the expected maximum and minimum (respectively) number of visits observed by nodes in the network after s steps. Denoting $V_i(s)$ as the number of visits on node i after s steps, formally:

$$B_u^{\max}(G_n, s) = E\left[\max_{i \in G_n}\{V_i(s)\}\right] \tag{2}$$

$$B_u^{\min}(G_n, s) = E\left[\min_{i \in G_n}\{V_i(s)\}\right] s.t.\ V_i(s) > 0 \tag{3}$$

3 Experimental Results: Medium-Scale Networks

3.1 Testbed and Experiment Setup

The simple and non-revisiting random walks were implemented in TinyOS 2.0.2 and evaluated on TWIST [3]. TWIST is a remote wireless sensor network test-bed deployed on a building and it has 102 tmotes. The nodes are not mobile, hence, the dynamics observed on the links are due to the surrounding environment.

[2] Additional important measure is the *maximum hitting time* which is the maximum over all h_{uw}, it will be considered in future work.

We utilized the lowest output power available on tmotes (-25 dBm)[3]. Figure 2 shows the communication graph of the network for (a) links with transmission probability above 0.9 and (b) above 0.7. The location of the nodes in the graph is not represented by their actual physical coordinates, but rather, by virtual coordinates obtained with Graphviz [1] based on the connectivity matrix.

We selected three nodes as the starting points for the walks (green, red and blue nodes). These nodes were selected to capture approximately the diameter and radius of the graph. For the remainder of the paper we denote these nodes by g, r and b, respectively. On each one of these three nodes we injected 10 simple random walks and 10 non-revisiting random walks, that is, a total of 60 walks were performed. Each walk was assigned a different random seed and it performed 1000 steps.

First, we present results concerning the temporal variance of the neighborhoods caused by link dynamics. Then, we present results for Pull-Only and Push-Pull querying.

3.2 Link Dynamics

Theoretical studies of random walks do not capture the impact of temporal dynamics on the total transmission costs incurred by the network. Most of these studies are done under ideal conditions that assume a constant neighborhood for all nodes throughout the network lifetime. Unfortunately, node failures, channel multi-path, dynamic environments and other factors lead to highly dynamic neighborhoods in WSN. In order to filter out links affected by these temporal dynamics, our implementation polls a node's neighborhood immediately before transferring the token (Neighborhood Discovery phase in Section 2.1)[4].

In this subsection, we show that *link asymmetry*and *neighborhood variance* are important challenges faced by random walks in WSN. We also show that the Neighborhood Discovery phase is a simple yet robust and efficient mechanism to cope with these dynamics.

Asymmetric Links. Link asymmetry refers to the phenomena where a node A can communicate with node B, but node B can not communicate with node A. Several works [28,10,9] have shown that asymmetric links are pervasive in WSN. Link asymmetry presents a serious inconvenience for random walks because bidirectional links are required to transfer the token at each step. In order to capture link asymmetry, at each neighborhood poll, we evaluated the difference between the number of nodes receiving the `NEIGH_DISC` packet and the number of `NEIGH_PRESENT` messages received at the sender. A neighborhood poll has 0-degree asymmetry if it reports bidirectional links with all neighbors, i.e. the token-holder receives `NEIGH_PRESENT` packets from all neighbors that received

[3] Utilizing higher output powers leads to graphs with high densities and short diameters. Graphs with these characteristics are not challenging querying scenarios.

[4] A different approach would be to poll neighbors when the network start functioning [4] or on a periodic basis. However, the neighborhood information of these mechanisms could quickly become inaccurate due to the high temporal variance of links.

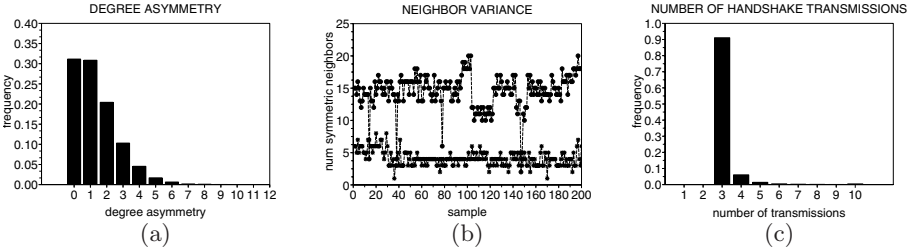


Fig. 3. Impact of link dynamics on (a) Degree Asymmetry and (b) Neighborhood Variance. Utilizing the Neighborhood Discovery phase limits the packet losses during the transfer of the token, as shown on (c).

the `NEIGH_DISC` packet. A neighborhood poll has x-degree asymmetry ($x > 0$) if it observes x asymmetric links in its neighborhood, i.e. there are x neighbors that received the `NEIGH_DISC` packet but their `NEIGH_PRESENT` replies were lost.

Figure 3 (a) depicts the results for 30000 neighborhood polls (approximately 300 polls performed by each node). We observe that only 30% of neighborhood polls observe purely symmetric links. Had the Neighborhood Discovery phase been performed only once (at the beginning of the process), some of the asymmetric neighbors would have been used in futile attempts to transfer the token.

Neighborhood Variance. Filtering asymmetric links is a necessary but insufficient step to cope with link dynamics. Symmetric links also have high temporal dynamics. Effects such as node failures and movements in the surrounding environment lead to intermittent links. These intermittent links affect significantly the neighborhoods observed by the nodes. We denote this intermittent phenomena as Neighborhood Variance. Figure 3 (b) captures the dynamics of the topology. This figure shows the number of *bidirectional* neighbors observed by two nodes at different instants of time (samples), one node with a high average degree and the other with low average degree. Clearly, the temporal dynamics observed by the nodes is significant – similar dynamics are observed for all nodes in the network. By providing an accurate representation of the available bidirectional neighbors, random walks can conduct a more-informed selection of the next step.

Number of Handshake Transmissions. The unreliable nature of WSN links requires a 3-way handshake mechanism to transfer the token reliably at each step. In order to minimize communication costs, it is desirable to use as few transmissions as possible at each step. Figure 3 (c) demonstrates the value of the Neighborhood Discovery phase. By filtering asymmetric links and intermittent bidirectional links, we avoid a potentially large number of packet losses during the transfer of the token. 90% of transfers utilize the minimum number of transmissions required (3). Furthermore, most transfers (>99%) are achieved with 6 transmissions or less (at most three packet losses during the handshake process). These reliable 3-way transmissions are obtained due to the temporal correlation in link quality [10,24]. A good link at time t is likely to still have a good quality at time $t + \delta$, but no accurate link quality estimation can be made

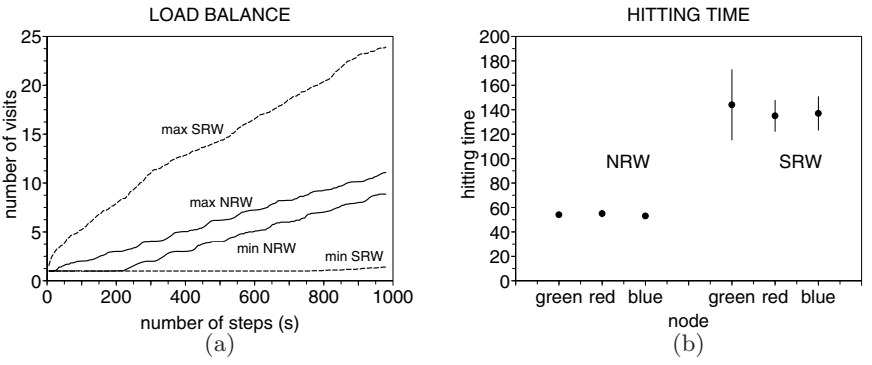


Fig. 4. (a) Load Balance and (b) Hitting Time of NRW and SRW. NRW outperforms SRW on both metrics.

for $t + \Delta$ (where $\Delta \gg \delta$). Hence, identifying reliable bidirectional links during the Neighborhood Discovery phase guarantees to a large extent the stability of the links during the token transfer.

3.3 Pull-Only Querying

In this subsection, we present results for our properties of interest in a Pull-Only querying scenario.

Load Balance. Due to the limited energy resources of WSN, it is important to distribute the energy consumption evenly across the network. Denoting $B_r^{\max}(s)$, $B_g^{\max}(s)$ and $B_b^{\max}(s)$ as the average of number of visits to the most visited node during the 10 SRWs of length s started at the red, green and blue nodes, we computed the average visits to the most visited node $B_{\text{srw}}^{\max}(s) = (B_r^{\max}(s) + B_g^{\max}(s) + B_b^{\max}(s))/3$ at each step $s = 1, \ldots, 1000$. The average number of visits to the least visited node in the SRW $B_{\text{srw}}^{\min}(s)$, and $B_{\text{nrw}}^{\max}(s), B_{\text{nrw}}^{\min}(s)$ for NRWs were computed in a similar way. As a measure of load balancing we consider the difference between the most and least visited nodes. Figure 4 (a) presents the visits to the most and least visited nodes in SRW and NRW. For example, when the number of steps $s = 400$, the least visited node on SRWs has on average 1 visit while the most visited node has on average 13 visits. For NRWs, the min and max averages are 3 and 5 respectively. This implies that NRWs do a significantly better job in distributing the use of energy resources. Furthermore, as the number of steps increase, SRWs continues to degrade, while NRWs keep the maximum and minimum number of visits within linear bounds (even distribution of load).

Hitting Time. In Pull-Only querying, the hitting time represents the expected time required to find an event that appears uniformly at random in any node of the network. For each node r, b, g, we computed the average hitting time and standard deviation for the 10 SRWs and 10 NRWs started at these nodes.

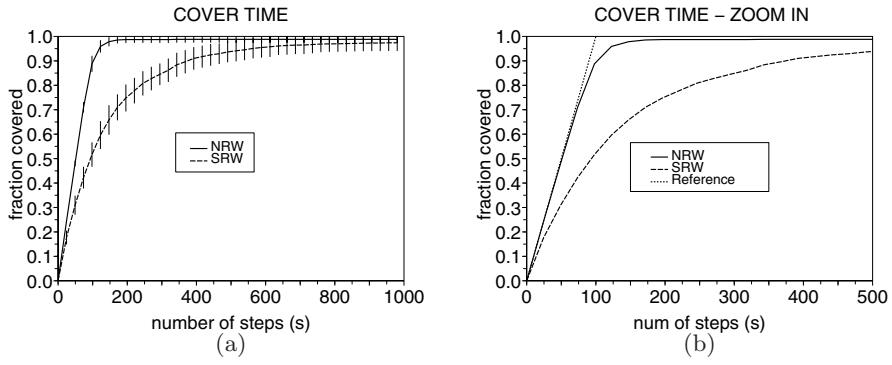


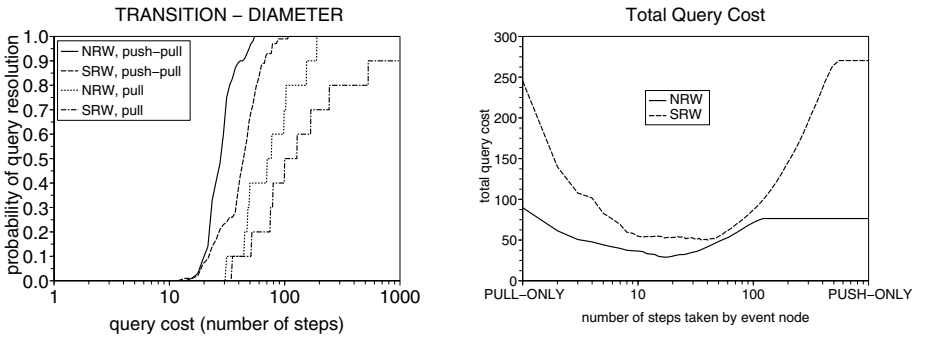
Fig. 5. Cover time of NRW and SRW. NRWs have faster cover times and less variance. NRWs also have linear partial cover times (up to approximately 70%).

Figure 4 (b) shows the results. The first three points represent NRWs and the next three points represent SRWs. There are two important observations to highlight. First, NRWs take approximately three times less steps than SRWs to solve the average query. Second, the variance among the 10 NRWs is almost negligible, but it is significant in SRW. Hence, NRWs are not only a faster and more energy efficient querying mechanism, but also provide less uncertainty.

Cover time. Several WSN scenarios require the estimation of the worst-case querying cost. When the data is of vital importance and it is not duplicated, or the query computes a function of all nodes, it may be necessary to visit (cover) all nodes in the network. Figure 5 presents the test-bed results for cover time. The SRW and NRW curves represent the average and standard deviation of 30 walks each (10 walks for each r, b and g node). In Figure 5 (a) we observe that NRW has two important advantages over SRW. First, NRW covers the network significantly faster than SRW. For instance, when $s = 100$, NRW covers 90% of the network while SRW covers 50% of the network. Second, the standard deviation of NRW is significantly lower than SRW, which leads to less uncertainty in the result of the querying process. Figure 5 (b) is a zoom-in of Figure 5 (a) and it shows that the partial cover time is linear for up to about 80% of the network (the dashed line has slope 1). The linear partial cover time indicate that most queries can be solved in linear time for NRWs.

3.4 Push-Pull Querying

The results presented in the previous subsection assumed that the events are not published (pull-only). However, several works in WSN have shown that push-pull querying mechanisms [7,14] can perform significantly better than pull-only querying. In push-pull querying, both, the sink and event inject walks and the query is solved when the two walks cross. In this subsection, we evaluate the performance of NRW in push-pull querying scenarios. The basic idea of gaining from a push-pull scheme is based on the following property.



(a) (b)

Fig. 6. (a) Probability of query resolution vs. query cost. Push-Pull has a significantly better performance than Pull-Only. (b) Total query cost. Both SRW and NRW have an optimal Push-Pull performance in-between the Push and Pull extremes. In general NRW with Push-Pull provides the best performance.

Property 1. *A sufficient condition for two random walks to intersect on a graph is that each walk visits at least* $\lceil \frac{n}{2} \rceil + 1$ *different nodes, where* n *is the number of nodes in the graph.*

Moreover, based on what it is known as the *birthday paradox*, it can be shown that two walks can intersect with high probability even in a sub-linear time:

Property 2 ([14]). *Two random walks on a graph will cross with high probability when each walk visits a uniform sample of* $O(\sqrt{n})$ *nodes, where* n *is the number of nodes in the graph*[5]*.*

Considering the above properties and the observation that the partial cover time of NRWs is linear up to a fraction well-beyond 50% of the network (Figure 5); then, by starting NRWs at the sink and the event nodes with a maximum number of steps s_{max} around $0.5n$, there is a high likelihood that the walks will cross and solve the query.

Query Resolution Transition. In order to evaluate the performance of push-pull querying, we obtain crossing-times from the walks collected in our experiments. Considering that each node injected 10 NRWs, we evaluated the 10×10 possible combinations of walk-pairs.

The first scenario we considered is the following. Each walk was set to perform a maximum of s_{max} steps, where s_{max} takes discrete values between 1 and 1000. The event-node starts a push (publish) walk and runs until it takes s_{max} steps or stops earlier if the sink-node is found. If the sink is not found, the event-trace

[5] The time to visit a uniform sample of $O(\sqrt{n})$ nodes depends on the *mixing time* [16] of the random walk which we don't study here.

remains alive. At a later time, the sink-node starts a pull (query) walk[6] and stops when it hits the trace left by the event-walk, otherwise, the sink-walk runs until completing s_{max} steps. Let s_{sink} and s_{event} be the number of steps taken by the sink and event walks, and $s_{total} = s_{sink} + s_{event}$ be total number of steps required to solve a query (the query cost).

Figure 6 (a) presents the cumulative distribution function *cdf* of the query resolution cost for SRW and NRW. For completeness, we also provide the *cdf* for pull-only querying[7]. The curves for push-pull are actually *lower bounds* for the probabilities of query resolution (since we use $2s_{max}$ as the query cost). In practice, the total query cost is much smaller than $2s_{max}$ (as we will show later).

In general, Push-Pull querying provides an order of magnitude better performance than Pull-Only querying for SRW and NRW. Figure 6 (a) shows the *cdf* for the diameter of the network (green and blue nodes) - approximately 8 hops[8]. For example, we observe that for a query cost of 60, the SRW pull-only solves the query with probability 0.2 and the NRW pull-only with probability 0.4. On the other hand, for the push-pull the SRW solves the query with probability about 0.85 and the NRW with probability 1.0. In Section 4, we will observe that as the size (diameter) of the network increases, NRWs increase their comparative performance with respect to SRWs.

In order to complete the test-bed evaluation of push-pull querying, we consider a second scenario. In this case the event-node issues a push walk of increasing lengths s_{event}. For a given event-node walk of length s, if it didn't reach the sink-node, *the sink-node issue a pull walk that continues to step until it crosses the event walk*. The length of the sink walk is denoted s_{sink}. The total query cost is $s_{total} = s_{event} + s_{sink}$ and we then evaluate the average total query cost for each s. Note that when $s = 0$, the query is pull-only and when s is very large the query is push-only[9]; for other values of s the query is push-pull.

Figure 6 (b) shows the average query cost for SRW and NRW for increasing push walk lengths. Our two main observations are validated again here. First the NRW solves the query in less steps than the SRW. Second the push-pull query resolution is more efficient in terms of number of steps than the pull-only or push-only queries. The data shows that NRW optimal query cost is about 29 steps when s_{event} is 17 steps, while the SRW cost is about 52 for 17 steps and about 50 at the optimum when s_{event} is 40. More generally, the optimum query cost seems to be when s_{event} and s_{sink} are about the same size.

4 Simulation Results: Large-Scale Networks

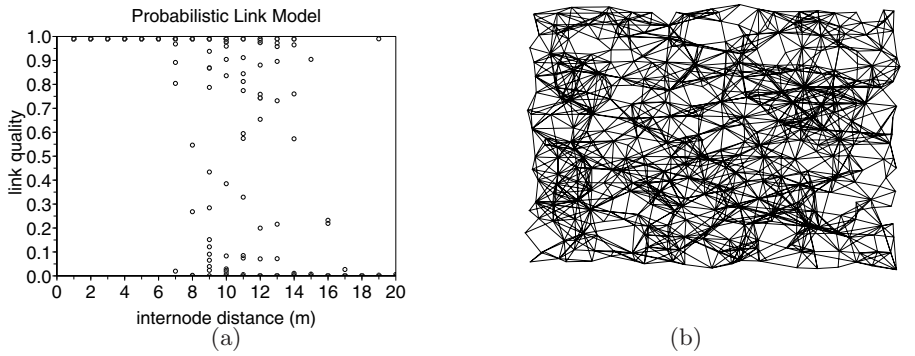
The test-bed results provide interesting empirical observations, but these results are confined to the particular size (102 motes) and characteristics of the TWIST

[6] The focus of this section is on crossing-times, and hence, we assume that the query walk is started within the lifetime of the event-trace.

[7] For pull-only queries, we utilize the 10 empirical walks available at each node. Due to these limited number of walks, the curves show the staircase form.

[8] A deterministic calculation of the diameter is not possible due to link dynamics.

[9] These costs are not necessarily equal since hitting times are symmetric.



(a) (b)

Fig. 7. Link Probability Model. (a) samples of link quality vs. distance. (b) sample of a network with 400 nodes and output power -10 dBm.

network . In order to validate the results for larger networks we perform simulations on WSN topologies that include link unreliability and link asymmetry. It is important to remark that these simulations capture some degree-heterogeneity due to multi-path channels and hardware variance, but they do not capture temporal variance. Hence, the main motivation of the simulations is to observe if the partial cover time of NRWs remain linear for larger networks.

4.1 Simulation Environment

We performed simulations using Scilab [2], an open-source alternative to Matlab.

Topology. Various network sizes were tested (100, 400, 900 and 1600). The network followed a normal-random topology, where nodes are initially deployed on a regular grid layout with an internode distance of $d = 5$ meters. Then, a 2-D normal r.v. is used to introduce a perturbation on the x and y coordinates of each node. The idea of a more uniformly distributed topology, compared to a pure random deployment, was borrowed from Glomosim [26].

Communication Model. The link quality among nodes was calculated based on the probabilistic model presented in [28]. This model captures unreliable and asymmetric links and it is given by:

$$p(d) = (1 - \frac{1}{2}\exp^{-\frac{\gamma(d)}{2}\frac{1}{0.64}})^{8f} \tag{4}$$

$p(d)$ is the link quality for an internode distance d. f is the number of bits transmitted and $\gamma(d)$ is the signal to noise ratio, which includes the output power and channel parameters. In our simulations f=160 bits and the channel parameters are 3.0 for both, the path loss exponent and the shadowing variance [23,28]. Figure 7 (a) shows samples of link quality for various internode distances; as we observe, the model resembles the behavior of empirical studies [27,25,28].

Simulations Run. We utilized an output power of -10 dBm. The output power and channel parameters presented above aim to recreate, to some extent, graph

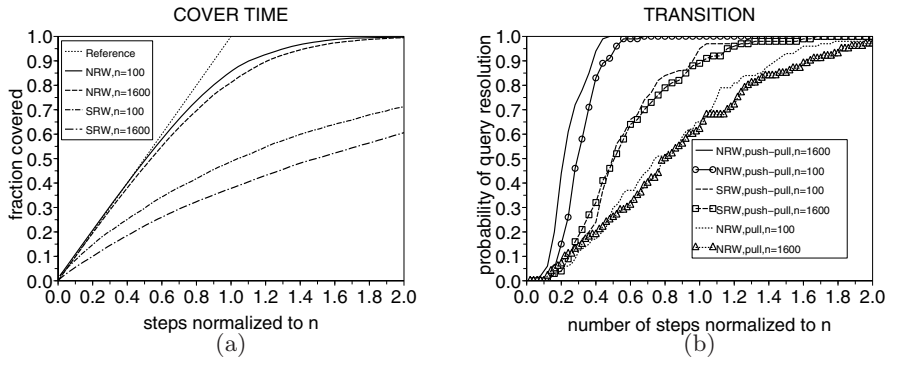


Fig. 8. Simulation Results (a) Cover time: partial cover time presents linear behavior. (b) Probability of query resolution: NRW with Push-Pull provide the best performance.

characteristics of TWIST such as degree distribution. Figure 7 (b) shows a sample topology with 400 nodes (only links with link quality > 0.7 are shown). The sink is assumed to be the node at the bottom-left corner of the graph. For Pull-Only querying, we performed 100 SRWs and 100 NRWs on each network size n. For the Push-Pull scenario, the event node was located at the top-right corner of the graph and we also run 100 SRWs and 100 NRWs starting at this node.

4.2 Simulation Results

Pull-Only. Due to space constraints, we focus on the results for cover time. The hitting time shows the same trend as the empirical results: NRW performs significantly better than SRW and the difference in performance increases in favor of NRW as the network size increases. Figure 8 (a) shows the cover time normalized to the size of the network n. In the interest of clarity, we plot results only for n=100 and n=1600 (n=400 and n=900 are in-between these curves).

The most important observation is that the partial cover times of the empirical and simulation results have the same trend: an initial long linear behavior Furthermore, once normalized, there is not a significant difference among partial cover times of NRWs for networks with different sizes. These results indicate that for larger networks NRW are also expected to solve most Pull-Only queries in linear time. For SRW, the partial cover times remain significantly longer than NRW, and the cover time degrades as the size of the network increases.

Push-Pull. In order to evaluate the effectiveness of the push-pull mechanism, the sink and event nodes were located at opposite extremes of the topology (diameter of graph). First, the node at the top right corner (event) started the walk for s steps, and then, the node at the bottom left corner (sink) started the walk. For completeness, the figure also shows the pull-only performance of NRWs. Figure 8 (b) depicts the *cdf* of query resolution for $n = 100$ and $n = 1600$. We observe that the Push-Pull with NRWs provides the best performance, followed by Push-Pull with SRWs and finally Pull-Only with NRWs. The results for

Pull-Only with SRWs is not shown but the probability of solving the query after $s = 2n$ was less than 0.4 for all networks' size. Also, the difference in performance between Push-Pull NRWs and Push-Pull SRWs increase for larger networks, however this improvement is not clearly observed in the figure.

5 Related Work

The research work on querying can be classified in two main groups: location-less and location-based. In location-less deployments, nodes have information only about their neighbors presence. The most notable querying paradigms are: flooding, expanding ring searches (controlled floods) and random walks. In location-based deployments, nodes also have location information about their neighbors. This information is very useful for geographic routing and geographic hash tables. In this work we focus on random walks on location-less scenarios.

Random walks on graphs have been studied mathematically, and there is a growing body of theoretical literature on the subject [8,16]. In the context of location-less wireless sensor networks, different variants of random-walk-based protocols have been proposed and analyzed. In one of the earlier works, Servetto and Barrenechea [20] proposed and analyzed the use of constrained random walks on a grid to improve the load-balanced routing between two known nodes. In [5], the authors argue that even simple random walks can be used for efficient and robust querying because their partial cover times show good scaling behavior. The ACQUIRE protocol [19] combines random walks with controlled floods and show that this hybrid mechanism can outperform flooding and even expanding-ring-based approaches in the presence of replicated data.

The evaluation of push-pull mechanisms was inspired by important related work. Rumor routing [7] advocates the use of multiple random walks from the events as well as the sinks, so that their intersection points can be used to provide a rendezvous point. On the same line of work, Shakkottai [21] analyzed different variants of random-walk-based query mechanisms and concludes that source and sink-driven sticky-searches (similar to rumor routing) provide a rapid increase of query success probability with the number of steps. Friedman *et. al.* [14] offered and evaluated via simulation probabilistic quorum systems that use different push-pull mechanism including simple and self-avoiding random walks. Contrary to the studies mentioned above, we consider the number of visits as an important parameter to *guide* the random walk.

Our work on NRWs is mainly motivated by [6,22]. These studies show in different ways that simple random walks lead to energy wastage due to their *blind* (re)visiting mechanism. In [6], instead of selecting only one node at random, the authors propose to select two (or more) nodes at random and select the one with the minimum number of visits as the next hop. In [22], the authors use homophyly and degree information to navigate the network, and the walk "ignores visited neighbors if there is at least one unvisited neighbor".

Based upon notable contributions on random-walk-based querying, we propose and analyze Non-Revisiting Random Walks with Push-Pull querying; a simple and efficient querying paradigm for practical WSN deployments.

6 Conclusions

In this work we evaluated the performance of Non-Revisiting Random Walks (NRW). Contrary to the blind selection performed by simple random walks, NRWs select the neighbor with the minimum number of visits. This mechanism increases the likelihood of encountering unvisited nodes, and as a consequence, provides a faster coverage.

We evaluated NRWs on (i) a test-bed consisting of 102 motes and (ii) with simulations on topologies consisting of unreliable and asymmetric links. Our results provide two important contributions. First, polling the neighborhood at each step of the walk is an efficient mechanism to cope with temporal link dynamics. This polling mechanism permits an accurate representation of the neighborhood, which allows a robust token-transfer and a well-informed selection of the next steps. Second, NRWs together with a simple push-pull mechanism are an efficient querying mechanism. NRWs maintain the elegance and simplicity of simple random walks, while at the same time can provide querying costs that are liner or sub-linear (depending on the size of the network).

In this work we considered only the cost of finding the data of interest (query), but not the cost required to transfer the information back to the sink. In future work we will evaluate the total cost (query + reply). Also, we plan to investigate the impact of non-TDMA MAC protocols on SRWs and NRWs.

Acknowledgement. This work has been funded by an IRCSET Postdoctoral Grant PD200857, SFI Grant No. SFI08-CE-I1380 and CONET, the Cooperating Objects Network of Excellence, EU FP7-2007-2-224053. The authors are thankful to Jan Hauer and Vlado Handziski for their support on the TWIST testbed.

References

1. `http://www.graphviz.org`
2. `http://www.scilab.org`
3. `http://www.twist.tu-berlin.de/wiki`
4. Ahn, J., Kapadia, S., Pattem, S., Sridharan, A., Zuniga, M., Jun, J., Avin, C., Krishnamachari, B.: Empirical evaluation of querying mechanisms for unstructured wireless sensor networks. SIGCOMM CCR 38(3), 17–26 (2008)
5. Avin, C., Brito, C.: Efficient and robust query processing in dynamic environments using random walk techniques. In: IPSN 2004 (2004)
6. Avin, C., Krishnamachari, B.: The power of choice in random walks: An empirical study. Computer Networks 52, 1 (2008)
7. Braginsky, D., Estrin, D.: Rumor routing algorthim for sensor networks. In: WSNA 2002 (2002)
8. Burioni, R., Cassi, D.: Random walks on graphs: ideas, techniques and results. J. Phys. A: Math. Gen. 38, R45–R78 (2005)
9. Cerpa, A., Wong, J.L., Kuang, L., Potkonjak, M., Estrin, D.: Statistical model of lossy links in wireless sensor networks. In: IPSN 2005 (2005)

10. Cerpa, A., Wong, J.L., Potkonjak, M., Estrin, D.: Temporal properties of low power wireless links: modeling and implications on multi-hop routing. In: MobiHoc 2005. ACM, New York (2005)
11. Chang, N., Liu, M.: Revisiting the ttl-based controlled flooding search: optimality and randomization. In: MobiCom 2004. ACM, New York (2004)
12. Chang, N., Liu, M.: Controlled flooding search in a large network. IEEE/ACM Transactions on Networking (TON) 15(2), 449 (2007)
13. Cheng, Z., Heinzelman, W.: Flooding strategy for target discovery in wireless networks. Wireless Networks 11(5), 607–618 (2005)
14. Friedman, R., Kliot, G., Avin, C.: Probabilistic quorum systems in wireless ad hoc networks. In: IEEE DSN 2008, June 2008, pp. 277–286 (2008)
15. Lawler, G.F.: A self-avoiding random walk. Duke Math. J. 47(3), 655–693 (1980)
16. Lovasz, L.: Random walks on graphs: A survey. Combinatorics, Paul Erdos is Eighty 2(1), 1–46 (1993)
17. Ni, S., Tseng, Y., Chen, Y., Sheu, J.: The broadcast storm problem in a mobile ad hoc network. In: MOBICOM 1999, p. 162. ACM, New York (1999)
18. Pemantle, R.: Vertex-reinforced random walk. Probability Theory and Related Fields 92(1), 117–136 (1992)
19. Sadagopan, N., Krishnamachari, B., Helmy, A.: Active query forwarding in sensor networks. Ad Hoc Networks 3(1), 91–113 (2005)
20. Servetto, S., Barrenechea, G.: Constrained random walks on random graphs: Routing algorithms for large scale wireless sensor networks. In: WSNA 2002 (2002)
21. Shakkottai, S.: Asymptotics of query strategies over a sensor network. In: IEEE INFOCOM, Citeseer, vol. 1, pp. 548–557 (2004)
22. Simsek, O., Jensen, D.: Navigating networks by using homophily and degree. PNAS 105, 35 (2008)
23. Sohrabi, K., Manriquez, B., Pottie, G.: Near-ground wideband channel measurements. In: IEEE Proc. VTC, New York (1999)
24. Srinivasan, K., Kazandjieva, M., Agarwal, S., Levis, P.: The beta-factor: measuring wireless link burstiness. In: SenSys 2008 (2008)
25. Woo, A., Tong, T., Culler, D.: Taming the underlying challenges of reliable multihop routing in sensor networks. In: SenSys 2003 (2003)
26. Zeng, X., Bagrodia, R., Gerla, M.: GloMoSim: a library for parallel simulation of large-scale wireless networks. ACM SIGSIM Simulation Digest (1998)
27. Zhao, J., Govindan, R.: Understanding packet delivery performance in dense wireless sensor networks. In: SenSys 2003, New York, NY, USA (2003)
28. Zuniga, M., Krishnamachari, B.: An analysis of unreliability and asymmetry in low-power wireless links. ACM Trans. Sen. Netw. 3(2), 7 (2007)