Exploiting random walks for robust, scalable, structure-free data aggregation and routing in mobile ad-hoc networks (MANETs)

Masahiro Nakagawa

Exploiting random walks for robust, scalable, structure-free data aggregation and routing in mobile ad-hoc networks (MANETs)

Masahiro Nakagawa

Dissertation submitted to the
Benjamin M. Statler College of Engineering and Mineral Resources
at West Virginia University
in partial fulfillment of the requirements
for the degree of

Doctor of Philosophy
in
Computer Science and Information Science

Lane Department of Computer Science and Electrical Engineering

Morgantown, West Virginia
2016

Keywords: Random walk, data aggregation, node visitation, self-repelling random walk

Abstract

Exploiting random walks for robust, scalable, structure-free data aggregation and routing in mobile ad-hoc networks (MANETs)

Masahiro Nakagawa

The focus of this thesis is on the design of scalable data aggregation protocols for Mobile Ad-hoc Networks (MANETs). Data aggregation Protocols that rely on network structures such as trees or backbones are not well suited for MANETs because the underlying topology of MANETs is constantly changing. On the other hand, unstructured techniques such as flooding and gossiping have a high messaging overhead and take a long time to finish. Therefore, in this thesis, we explore the use of random walks as a structure-free alternative for data aggregation in MANETs.

The basic idea is to introduce one or more tokens that successively visit each node in a MANET by executing a random walk and compute the aggregate state. While random walks are simple, robust and overhead-free, plain random walks tend to be slow in visiting all nodes because the token can get stuck in regions of already visited nodes. Therefore, we first introduce self-repelling random walks (SRRW) in which at each step, the token chooses a neighbor that has been visited the least number of times. While SRRW significantly speeds up random walks in the initial stages, towards the end a slowdown is observed when a significant fraction of nodes are already visited. To address this shortcoming, we then develop two complementary strategies that speed up data aggregation.

First, we introduce gradient biased random walks (a pull-based strategy) where short temporary multi-hop gradients are used to pull the tokens toward unvisited node. We prove that gradient biased random walks achieve a cover time of $O(N)$ and message overhead of $O(NlogN)$ where N is the number of nodes in the network. Next, we introduce a push-based strategy in which self-repelling random walks are complemented by a single step push phase before the random walk phase, in which each node broadcasts its information to its neighbors. We show that this small push goes a long way in speeding up data aggregation. Push based random walks finish data aggregation in $O(N)$ message and time. Finally, we describe hierarchical extension of the push-based protocol which can produce multi-resolution aggregates at each node using only $O(NlogN)$ messages.

All our results are validated using simulations in ns-3 in networks ranging from 100 to 4000 nodes under different network densities, node speed and mobility models.

Acknowledgements

First, I want to thank my committee chair and advisor, Dr. Vinod K. Kulathumani, for guiding me in the research and providing me the opportunity to work with him and his other graduate students. This thesis work has been made possible with his constant support and guidance.

I also want to thank Dr. Katerina Goseva-Popstojanova, Dr.Yasser P. Fallah, Dr.YanFang Ye, and Dr.Ashish Nimbarte for being a part of the my committee. I got variable comments/advice from them.

I would also like to thank Dr. Anish Arora for providing valuable feedback on the random walk based protocols, which improved the outcome of this thesis.

Last but not the least, I want to express my gratitude to my family. My parents have been very encouraging on my decision to go purse my study. Their support has been relentless and a constant motivation to my desire of pursuing Computer Science in school.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

The focus of this thesis is the design of scalable and robust data aggregation protocols for MANETs. A MANET (Mobile Ad-hoc NETwork) is a type of wireless network that consists of a collection of small moving devices called nodes, each equipped with a radio and a processor. A node can communicate with all nodes within its communication range. Also, a node in MANETs moves according to an underlying motion model that is application-specific. Due to the underlying motion, the topology of MANETs changes continuously. These characteristics make the protocol design for MANETs very challenging.

Data aggregation protocols aim to collect summary information from all the nodes in the network. An aggregated value can be some statistical measure such as average, standard deviation of the sensor state or it could be the number of nodes meeting a specific predicate. For example, the density of vehicles in a given area of city can be monitored using data aggregation. A query can be sent to all the vehicles in a given area. (This can be done using an infrastructure in the area). The vehicles in the area can aggregate information and send only the aggregate value back to the infrastructure. This way, individual vehicles do not have to send data back to the infrastructure. (The same approach can be extended to monitor average speed or weather information if a vehicle has sensors that can detect weather change).

There are two different types of data aggregation: duplicate-sensitive data aggregation

Table 1.1: Mapping between speed and link change per node per second

Size	3m/s	9m/s	15m/s	21m/s
100	1	5	7	9
200	2	6	9	12
300	2	7	10	14
500	3	8	12	16
1000	3	9	14	18
2000	3.8	10	16	20
4000	4	12	18	23

and duplicate-insensitive data aggregation. If the aggregated value is sensitive to duplicate count of the data from the same node, the aggregation is called duplicate-sensitive aggregate. An average, a mode, and a standard deviation are examples of duplicate-sensitive data aggregation. On the other hand, finding the maximum value or the minimum value are examples of duplicate-insensitive data aggregation (also called ODI aggregation) because counting data from the same node multiple times does not change the aggregated value.

1.2 Background and problem statement

Significant research has been done on the design of data aggregation for static sensor networks and networks with mostly stable links. These protocols typically achieve data routing by forming a tree or a network backbone structure over the network topology. Once such a structure is created, data aggregation can be performed by routing along such a fixed structure to collect data. We refer to these protocols as structure-based protocols and examples of structure-based protocols can be found in [1, 2, 3, 4]. However, these structure-based protocols are known to have a scalability issue if the underlying structure is not stable. This is especially true in MANETs, where due to the movement of the nodes, the links between nodes change very frequently. In Table1.1, we state the average number of link change per node per second over different node speeds at the different network sizes and deployment densities that we consider in this thesis. As seen in Table 1.1, the number of link changes per second are quite significant. Note that in the presence of such high link dynamics, the structure-based protocols need to detect a broken link as soon as it occurs

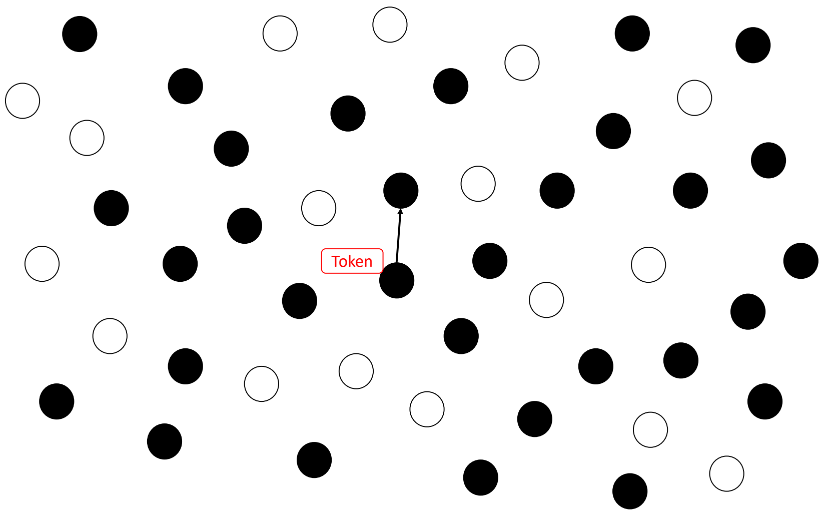


Figure 1.1: The random walk may pass a token to visited node even unvisited nodes are present

and then fix it in a timely fashion. This incurs a very high message overhead and causes a scalability issue [5].

1.2.1 Random walks

To avoid the scalability issue posed by MANETs, in this thesis we explore the use of random walks as a simple, unstructured tool for data aggregation in MANETs. The idea is to circulate one or more tokens in the network that traverse the network using a random walk strategy for moving from one node to the next. When a node is visited for the first time, the node state is aggregated into the token. Thus, when all nodes have been visited at least once, the aggregate is computed and the resulting aggregate can be simply flooded back to all the nodes. To ensure that data is not duplicated in the aggregate, each node can keep track of whether it has been visited. This way, a node's data will be added into the aggregate only if it has not been visited before. Note that this random walk based idea is inherently stable in the presence of network dynamics [6]. Moreover, there are no critical points of failure. There is also no memory overhead because the token only carries the overall aggregate and not the individual data items. This results in a simple, robust protocol. However, plain random walks are quite slow and require a long time to cover all the nodes because they can

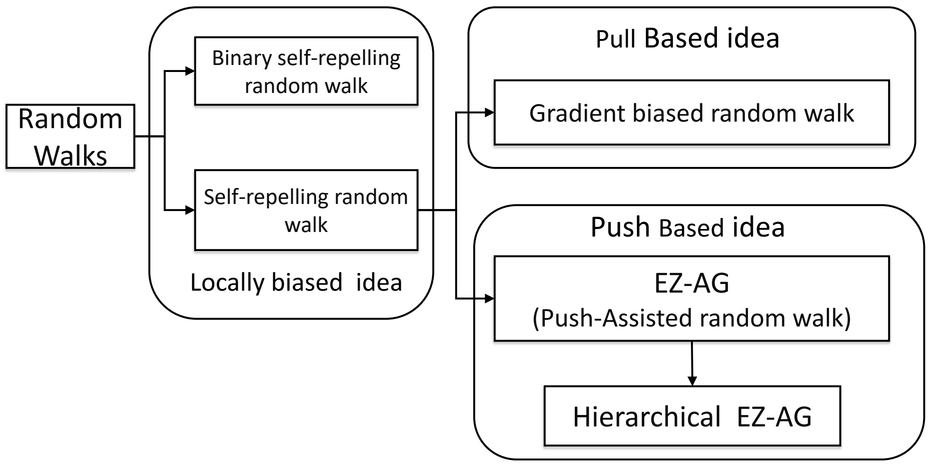

Random Walks
Binary self-repelling random walk
Self-repelling random walk
Locally biased idea
Pull Based idea
Gradient biased random walk
Push Based idea
EZ-AG
(Push-Assisted random walk)
Hierarchical EZ-AG

Figure 1.2: Relationship between each variation of random walks in this thesis.Locally biased idea significantly decreases a cover time but slows down after covering certain fraction of the network. Push idea and Pull idea is used to cope with this slowdown

be stuck in regions of already visited nodes and *finding* the next unvisited node may take a large number of wasted token traversals (see Fig.1.1).

The objective of this thesis to design strategies that speed up random walks for data aggregation, while still retaining their simplicity and robustness.

1.3 Thesis overview

An overview of the different variations of random walks that we explore in this thesis is shown in Fig. 1.2. We now summarize these ideas.

1.3.1 Local biasing techniques

In order to avoid slow down of the pure random walk, we first explore some local biasing strategies for random walks. One such idea is a self-repelling random walk [7] in which the token is passed to the neighbor that has been visited least number of times (with ties broken uniformly at random). If a neighbor is an unvisited node, the number of times this node has been visited is zero. Thus, self-repelling random walks prioritize the unvisited nodes over the previously visited nodes. When a token holder has no unvisited neighbor, the self-repelling random walk selects a node that has been visited the least number of times.

The self-repelling random walk is known to traverse a two dimensional lattice uniformly [8]. We empirically show that this uniformity characteristic during traversal of nodes holds even in dynamic networks. This property is important because it shows that a self-repelling random walk is likely to explore unvisited regions in the network as opposed to getting stuck in regions of already visited nodes. Due to this uniformity property in traversal, self-repelling random walks significantly reduce the cover time. We analytically show that self-repelling random walks achieve a cover time of $O(Nlog(N))$. Moreover, we prove that by using self-repelling random walks, a significant portion of the network can be covered much faster than the pure random walk.

However, self-repelling random walks start to slow down when the fraction of already visited nodes in the network exceeds a certain threshold. This is because self-repelling random walk (giving higher precedence to an unvisited node) helps only when a node has at least one unvisited neighbor. If all neighbors are already visited, the self-repelling random walk loses its advantage. Until a token moves to a node with unvisited neighbor, it keeps visiting already visited nodes. While the order of convergence with respect to node size N remains $O(Nlog(N))$, the slow down creates a long tail in the convergence.

One of the consideration when implementing self-repelling random walks is the number of requests generated for a token during each successive iteration. If all neighbors request a token by specifying the number of times they have been visited, the number of token requests will grow with the network density. To avoid this, we seed the request generation such that nodes with fewer number of visits generate a token request earlier. Also, if a node overhears a token request message sent from a node having higher precedence, it suppresses its own token request message. These two actions ensure that the number of token requests stay fairly constant and small irrespective of the network size and density.

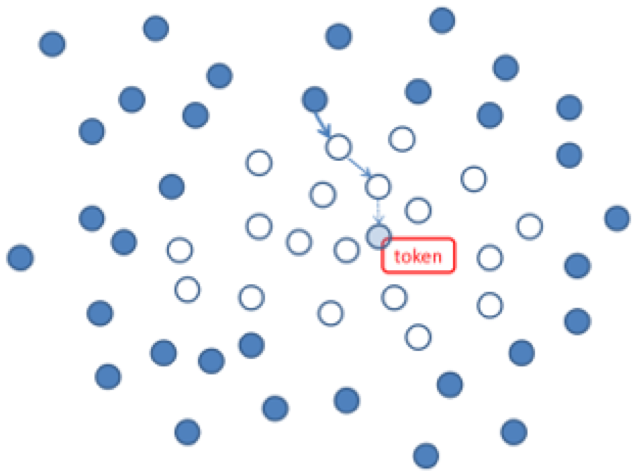


Figure 1.3: During token passing, a token may be surrounded by an island of visited nodes (white circles), i.e., all neighboring nodes have already been visited. Nodes that have not yet been visited (indicated by dark circles) periodically set up a gradient using the set of visited nodes to attract the token towards them.

1.3.2 Speeding up self-repelling random walks using short gradients

In the previous subsection, we saw that the pure random walk can be made faster to an extent by using self-repelling random walks. However, when most nodes are visited, self-repelling random walks exhibit a slow down. To address this slow down, we first use a *pull-based* idea where the tokens are pulled towards unvisited nodes using short multi-hop gradients.

To prevent a token from staying in the region of visited nodes while there are still unvisited nodes to be explored, we set up short, temporary multi-hop gradients to pull a token towards an unvisited node. (See Fig. 1.3) We analytically show that this gradient-based strategy achieves a cover time of $O(N)$. Thus, the order of convergence improves by a factor of $log(N)$. In doing so, the gradient biased random walk introduces a gradient message overhead of $O(Nlog(N))$ to pull tokens towards unvisited nodes. Nevertheless, this overhead is compensated by a reduction in the required number of token transfers. In fact, our simulations show that the ratio of token transfers to that of node size, (i.e., the exploration overhead of gradient biased random walk) remains constant (irrespective of the network size) and close to 1.

1.3.3 EZ-AG: (Push assisted self-repelling random walk)

In the context of duplicate-insensitive (ODI) aggregation, we study a *push based* idea in which before the self-repelling random walk starts, we introduce a single step push phase. In the push phase, each node advertises its state (data) to its neighbors. We call this technique EZ-AG. EZ-AG is designed for the ODI data aggregation [9, 10, 11, 12]. In an ODI-synopsis, the data from the same node can be aggregated multiple times without affecting the result. Examples of such duplicate-insensitive data aggregation are MAX and MIN. Other statistical count such as the SUM and AVERAGE can be also implemented with the ODI synopsis using probabilistic techniques. [10, 13].

To speed up random walks, EZ-AG uses a complementary *push* phase that speeds up the convergence. The *push* phase consists of simple one-hop broadcast from each node in the network. Before the self-repelling random walk is started, each node broadcasts its own state (some information obtained at the node such as sensor reading). Thus, before the self-repelling random walk is started, each node now carries information about all its neighbors. As a result, random walk can finish aggregation without visiting all nodes. We show that with the addition of the *push* phase, the aggregation can finish *before* the self-repelling random walk starts to slow down. This significantly improves the required time and message: EZ-AG requires only $O(N)$ message and time to complete the aggregation. We compare our results with structure-free techniques for the ODI data aggregation such as gossiping and show a $log(N)$ factor improvement in messages compared to existing gossip based techniques.

1.3.4 Hierarchical EZ-AG

When a network is quite large, providing a single aggregate value to the entire network may not be sufficient. It may be more useful to provide each node with a distance sensitive multi-resolution aggregate, i.e., to provide aggregate information about neighborhoods of increasing sizes around itself. EZ-AG can be extended to provide such multi-resolution synopsis of nodes in the network with only $O(Nlog(N))$ messages.

Existing techniques for such hierarchical aggregation require $O(Nlog^{5.4}(N))$ messages [11]. Thus, EZ-AG offers a poly-logarithmic factor improvement in terms of the number of messages for hierarchical aggregation. Moreover, EZ-AG can also be used to generate hierarchical aggregates that are distance-sensitive in refresh rate, where aggregates of nearby regions are supplied at a faster rate than farther neighborhoods.

1.4 Summary of contributions

- We introduce the idea of applying several variants of random walks for robust and structure-free data aggregation in MANETs.
- We analytically show that the self-repelling random walk has a cover time of $O(Nlog(N))$ in a mobile network that is modeled as a time-varying random geometric graph.
- We empirically characterize the uniformity with which nodes are visited by a self-repelling random walk in a mobile network. We highlight the importance of this uniformity characteristic for speeding up data aggregation in MANETs. Note that in [8], authors have highlighted the uniformity of the self-repelling random walk on a 2-d lattice. Here, we have extended this study and empirically shown that this uniformity holds for MANETs under different mobility models.
- We describe a pull-based idea for speeding up self-repelling random walks and achieving duplicate sensitive data aggregation in MANETs. By adding short multi-hop temporary gradient to pull tokens, we show that the aggregation time and the number of messages are bounded in gradient biased self-repelling random walk by $O(N)$ and $O(Nlog(N))$ respectively. The gradient formation adds message overhead but we empirically show that the message overhead is compensated by the faster convergence time.

- We describe a novel push-based idea for speeding up self-repelling random walks and achieving duplicate insensitive data aggregation in MANETs. By adding this push phase, we show that both the aggregation time and the number of messages are bounded in EZ-AG by $O(N)$. In fact, we show that aggregation is completed in significantly less than N token transfers. The protocol is thus extremely simple, requires very little state maintenance (each node only remembers the number of times it has been visited), requires no network structures nor clustering. We compare our results with structure-free techniques for the ODI data aggregation such as gossiping and show a $log(N)$ factor improvement in messages compared to existing gossip based techniques.

- We provide an extension to the EZ-AG which supplies multi-resolution aggregates to each node. In the networks that are quite large, providing each node with only a single aggregate may not be sufficient. Hierarchical EZ-AG addresses this issue by providing each node with multiple aggregates of neighborhoods of increasing size around itself using only $O(Nlog(N))$ messages. Moreover, we also show that aggregates of nearby regions can be obtained at a progressively faster rate than farther regions. The Hierarchical EZ-AG outperforms existing techniques for multi-resolution data aggregation by a factor of $log^{4.4}(N)$.

- We evaluate all our protocols using simulations in ns-3[14] on networks ranging from 100 to 4000 nodes under various mobility models and node speeds. We also evaluate and compare our protocol with a prototype tree-based technique for data aggregation and show that our protocol is better suited for MANETs and remains scalable even under the high mobility. In fact, *the performance of random walk based protocols is seen to improve as node mobility increases.*

1.5 Outline of this thesis

This thesis consists of 11 chapters. Chapter 2 discusses related work on random walks and data aggregation. Chapter 3 discusses the system model, network model, the mobility model and evaluation metrics. Chapter 4 introduces locally biased random walks. It explains the self-repelling random walks and provides an analytical argument for self-repelling random walks. Chapter 5 introduces detailed protocol description for gradient biased random walk and provides an analytical argument for the gradient biased random walk. Chapter 6 introduces EZ-AG. It provides a detailed protocol description and an analytical argument. In Chapter 7, we talk about hierarchical EZ-AG and explain how hierarchical EZ-AG operates. An analytical argument and a proof for its cover time and message overhead are also provided in this chapter. In chapter 8, we compare and contrast different types of random walks (pure random walk, self-repelling random walk, gradient biased random walk, and EZ-AG) using ns-3 simulation. In chapter 9, we compare and contrast the EZ-AG with the prototype tree-based approach known as Aggregate tree protocol. In chapter 10, we discuss some implementation issues. In Chapter 11, we provide a conclusion.

Chapter 2

Background work and existing literature

In this chapter, we describe related work. First, we discuss structure-based protocols that can be used for data aggregation in static sensor networks. In static sensor networks, the nodes are stationary and do not move. Thus, structure-based protocols can form either a tree structure or a network backbone for routing data. However, when mobility is introduced, these approaches lose the scalability as the mobility breaks the link and requires significant amount of message for fixing the tree structure or backbone. Then, we discuss some structure-free protocols that have been used for data aggregation. Existing structure-free protocols do not have scalability issue. However, they are slow and require more messages to obtain an aggregate compared to random walk based approaches. Finally, we provide related work on random walks.

2.1 Structure-based protocols for static sensor networks

The problem of data aggregation and one-shot querying has been well studied in the context of static sensor networks. It has been shown that in-network aggregation techniques using spanning trees and network backbones are efficient and reliable solutions for the problem [1, 2, 3, 4]. However, in the context of a mobile network, such fixed routing structures are likely to be unstable and could potentially incur a high communication overhead for maintenance [5]. In this paper, we have systematically compared EZ-AG with a prototype tree-based technique for data aggregation and have shown that it outperforms the tree-based idea in mobile networks. We notice that the improvement gets progressively more significant as the average node speed increases.

In this subsection, we discuss several structure-based protocols designed for sensor networks. The structure-based protocols we discuss in this subsection are designed to form a tree structure first and use this tree structure to forward/route information to a designated node. All structure-based protocols use some type of heartbeat message broadcasting. One such example discussed is a query message. This approach to fixing a broken link has its trade off. To reduce time required to fix a broken link, the rate of the heartbeat message needs to be increased. This trade off results in either a lower packet arrival rate (without having correct topology, information cannot be routed to a designated node) or increased total messages. Usually, nodes in sensor networks are battery powered with quite a long expected lifetime. This constrains the total messages required to be as small as possible. If heartbeat rate is kept low to save energy, data cannot be delivered properly. On the other hand, if this rate is increased, the number of total messages increases and cannot meet the energy requirement.

2.1.1 Tiny AGgregation

Tiny Aggregate(TAG) [2] is an example of a structure-based, duplicate-sensitive, data aggregation protocol. TAG also creates a tree-based structure and utilizes this tree-structure to forward data from all nodes to a data sink. TAG tries to reduce the total number of messages by processing aggregates at each node. TAG tries to cope with node mobility and unstable links in a very similar manner as other structure-based protocols. If a node does not transmit data for several rounds (i.e., the link to its parent is broken), a node broadcasts a heartbeat message to advertise itself to its neighbors. The neighbor receiving this heartbeat message will reply to it. The node sending a heartbeat can fix its broken link using this reply by making the responder its parent node. It is obvious how fast TAG can detect broken link and fix topology depending on how quickly heartbeat messages are sent. If this message is sent too often, it will consume more energy.

In the data aggregation, losing some intermediate nodes might have a very strong impact on the resulting aggregate. Authors evaluated the effect of a single loss (a single node in the tree loses its parent). It turns out the impact of the single loss depends on the type of aggregate and level of this node. COUNT is shown to be more sensitive to loss than others. If a node moves out from its parent's communication range is close to the root of a tree, significant amounts of data are lost and the resulting aggregate will not be accurate. Such issue will not happen in our random walk based method. In our method, even the network is partitioned, a token circulates until the network is joined again.

It is possible to give redundancy to TAG by allowing TAG to use multi-path routing. By having multiple paths to the root node, the single loss may not break the protocol completely. Even if some links are broken, there may be still valid path(s) to the root node. This approach essentially allows more time before all routes to the root node fail. As long as there is a single valid path to the root node, aggregation can be successfully performed. However, with multi-path routing, a root node may receive data sent from one node multiple times as data is routed to the root using all available paths. This duplicated count may cause an issue with some types of data aggregation.

2.1.2 Synopsis diffusion

In [10], Nath et al. presents a solution to this problem. Nath proposed synopsis diffusion, a way to utilize multi-path routing while avoiding duplicate counting of identical information. Synopsis diffusion decouples aggregation and routing by using order and duplicate insensitive (ODI) synopses. To achieve this, the synopsis, synopsis generation function, synopsis fusion function, and synopsis evaluation function need to be designed for each type of aggregation. Some examples on how to design such synopsis functions (generation, fusion, and evaluation) and synopsis can be found in [10, 15].

2.1.3 Directed Diffusion

Direct Diffusion [4] is another example of a structure-based approach. Direct Diffusion tries to route information efficiently to the specified root node. First, a node wanting to find out something about a network will send a query. The query specifies the data rate and expiration time of the query. Each node needing to respond query will generate results at every period specified in the query until this query expires. This query is broadcast from a root node and all nodes receiving this query will repeatedly rebroadcast it (i.e., flood query). The received query is registered at each node and a duplicated query will be ignored. After the initial query, the root node keeps rebroadcasting the same query with a faster data rate. The duration of this query broadcasting determines how fast direct diffusion can cope with topology changes.

As direct diffusion utilizes rebroadcasting of messages to fix broken links, it suffers the same issue as other structure-based protocols. Once a node starts to move, the rate of link change starts to increase. If we do not reduce the duration at which a root node rebroadcasts query, the data reception rate will drop significantly. On the other hand, if the duration at which root node rebroadcast gets smaller, the total number of query messages sent will drastically increase. At certain speeds or network size, direct diffusion is expected to have very low packet delivery rate. The number of query replies reaching root drops significantly.

2.1.4 CTP:Collection Tree Protocol

Collection Tree Protocol (CTP)[3] is one example of a tree-based structure approach which tries to provide reliable, robust and efficiency data collection. CTP tries to route data from node to closest root node. (CTP works with multiple root nodes.) CTP achieves this by creating and maintaining a tree-based structure and forwards all data using this tree structure. CTP maintains tree-structure by periodically broadcasting maintenance packet. To be reliable, CTP needs to be able to detect broken links and fix these as soon as possible. If a node has a broken link, the node can update its parent to the sender of maintenance packet. Thus, how quickly CTP can fix broken links depends on how fast a root node broadcasts a maintenance packet. This contradicts with efficiency. If there is no broken link, there is no need for maintenance. To obtain better performance, CTP dynamically adjusts the timer sending the maintenance packet. When CTP detects some link change around it, it resets the timer to its smallest value. Then it doubles the timer value up to its max value. This adaptive timer helps CTP to break the trade-off between power consumption and packet delivery rate.

2.2 Structure-free protocols

Flooding, neighborhood gossip, and spatial gossip are three structure-free techniques that can be used for data aggregation. Note that flooding data from all nodes to every other node has a messaging cost of $O(N^2)$. Alternatively, one could use multiple rounds of neighborhood gossip where in each round a node averages the current state of all its neighbors and this procedure is repeated until convergence [16, 17]. However, this method requires several iterations and has also been shown to have a communication cost and completion time of $O(N^2)$ for convergence in grids or random geometric graphs, where connectivity is based on locality [18].

In [11] and [12], a spatial gossip technique is described where each node chooses another node in the network (not just neighbors) at random and gossips its state. When this is repeated $O(log^{1+\epsilon} N)$ times, all nodes in the network learn about the aggregate state. Note that this scheme requires $O(N.polylog(N))$ messages. Our random walk based protocol, EZ-AG, requires only $O(N)$ messages. Note also that while all this prior work is on static networks, we demonstrate our results on MANETs.

Table 2.1: Cover time and Hitting time of graphs

Type of graph	Hitting time	Cover time
Lolipop	$O(N^3)$	$O(N^3)$
Path	$O(N^2)$	$O(N^2)$
Complete Graph	$O(N)$	$O(NlogN)$

2.3 Random walks

There have been several articles on random walks to bound its cover time for different types of graphs. For instance, a complete graph has a cover time bound of $O(Nlog(N))$ and a lollipop graph has a cover time of $O(N^3)$ [19, 20, 21]. (See Table 2.1) Typically, dense and highly connected graphs have a lower cover time that tends to increase as connectivity decreases [6]. A speed-up by a factor of k has been shown when k independent random walks are utilized in the graph [22, 23, 24]. In [25] , the authors study cover times of random walks and lazy random walks on temporal graphs governed by adversarial strategies.

Self-repelling random walks are those in which at each step, a random walk moves to a neighbor that has been visited the least number of times (with ties broken using a random choice)[7]. The uniformity with which such random walks explore a graph has been analyzed in [8]. The analysis [8] shows that the variance in the number of visits at each node is tightly bounded, resulting in a uniform distribution of visited nodes across the network. In this thesis, we have exploited this observation to show that the cover time of self-repelling random walks in mobile networks is $O(Nlog(N))$. We further show that by using short temporary gradients to guide the self-repelling random walks, the cover time can be improved to $O(N)$. Cover time for self-repelling random walks in time-varying graphs (relevant for mobile networks), has not been previously characterized, to the best of our knowledge.

For *static* mesh networks modeled as geometric graphs with a uniform degree of connectivity, the expected cover time is known to be $O(Nlog^2(N))$. In this thesis, we have shown that the cover time can be improved to $O(N)$ with gradient biasing. We also note that although the cover time for self-repelling random walks has the same upper bound as pure random walks on geometric graphs, the property of quickly covering a large fraction of nodes without much exploration is critical for the scalability and cover time bounds of the gradient biased random walk. In other words, applying the idea of gradients to regular random walks will not yield improvement in cover time.

We also note that in previous work there has been some *empirical* evidence of obtaining an $O(Nlog(N))$ cover time for *static 2-d grids* by exploiting some form of choice in random walks, where preference is given to less visited nodes at each step [6]. There is also empirical evidence from previous studies that shows that locally biased random walks visit a significant fraction of the network without much wasted exploration. In [26], it is empirically observed that about 80% of the network can be covered in less than N steps, where N is the number of nodes in the network. These empirical observations further verify the claims made in this thesis.

Chapter 3

System Model

In this chapter, we discuss the system model and mobility models used in this thesis. Also, we explain the metrics we used in our simulation.

3.1 Network Model

We consider a mobile network of N nodes modeled as a geometric Markovian evolving graph [27]. Each node has a communication range R. We assume that the N nodes are independently and uniformly deployed over a square region of sides $\sqrt{A}$ resulting in a homogeneous network density $\rho = \frac{N}{A}$ of the deployed nodes. Consider the region to be divided into square cells of sides $\frac{R}{\sqrt{2}}$. Thus, the diagonal of each such cell is the communication range R. Let $R^2 > \frac{2clog(N)}{\rho}$. It has been shown that there exists a constant $c > 1$ such that each such cell has $\theta(logN)$ nodes with high probability, i.e., the degree of each node is $\theta(logN)$ with high probability. Such graphs have been referred to as geo-dense geometric graphs [6]. Denote $d = \theta(logN)$ as the degree of connectivity.

3.2 Mobility Models

We consider three different mobility models for our evaluations.

- The first is a random direction mobility model (with reflection) [28, 29] for the nodes. This is a special case of the random walk mobility model [30]. In this mobility model, at each interval a node picks a random direction uniformly in the range $[0, 2\pi]$ and moves with a constant speed that is randomly chosen in the range $[v_l, v_h]$. At the end of each interval, a new direction and speed are calculated. If the node hits a boundary, the direction is reversed. The motion of the nodes is independent of each other. An important characteristic of this mobility model is that it preserves the uniformity of node distribution: given that at time $t = 0$ the position and orientation of users are independent and uniform, they remain uniformly distributed for all times $t > 0$ provided the users move independently of each other [29, 31].

- The second is random waypoint mobility model. Here, each mobile node randomly selects one location in the simulation area and then travels towards this destination with constant velocity chosen randomly from $[v_l, v_h]$ [30]. Upon reaching the destination, the node stops for a duration defined by the *pause time*. After this duration, it again chooses another random destination and the process is repeated. We set the pause time to 2 seconds between successive changes.

- The third is Gauss Markov mobility model. In this model, the velocity of mobile node is assumed to be correlated over time and modeled as a Gauss-Markov stochastic process [30]. We set the temporal dependence parameter $\alpha = 0.75$. Velocity and direction are changed every 1 second in the Gauss Markov Model.

3.3 Metrics

A key metric that we are interested in is the number of times the token is transferred to already visited nodes. We present this in the form of *exploration overhead* which is defined as the ratio of the number of token transfers to the number of unique nodes whose data has been aggregated into the token. We compute exploration overhead at different stages of coverage as the random walk progresses.

We note that since we study random walks on mobile networks, the notion of time is also related to node speed. Moreover, when dealing with wireless networks, time also involves messaging delays. Therefore, during empirical evaluation we separately characterize the actual convergence time (in seconds) along with the number of steps (i.e., number of token transfers).

Chapter 4

Local-biasing to speed up random walks

In this chapter, we discuss the idea of utilizing local information to speed up random walks. As we mentioned, a random walk is agnostic to a node movement but its cover time is very long. As a random walk picks one of its neighbor randomly to pass a token, the token will be passed to same node multiple times before it can cover all the nodes in the network. To redress this shortcoming, we utilize self-repelling random walks [32]. Self-repelling random walks were introduced in the 1980s and have been studied extensively in the physics literature. Self-repelling random walks bias pure random walk using the number of times each node has been visited so far. Self-repelling random walks give priority to the node with the least number of visit. (Tie is broken randomly). By doing so, self-repelling random walks can minimize the total time required to visits all the nodes at least once.

4.1 Self-repelling random walks:Protocol description

To achieve self-repelling random walks, each node stores two variables, *holder*, and *visited.* The variable *visited* tracks the number of times that a node has been visited by any of the tokens; *visited* is initially 0 at all nodes. When a token first arrives at a node, *visited* is set to 1. Tokens are assumed to be initiated at a random set of nodes. All nodes in which a token is initiated are marked as visited by default and the token value is initialized to the data at the corresponding node. The variable *holder* is used to identify nodes that currently hold a token.

4.1.1 Token passing with self-repelling random walks

For self-repelling random walks, a token holder announces that it has a token. Nodes that hear this message start a timer to send a request at a random slot within a chosen interval. The number of intervals is implementation specific. In our simulation, we set the number of intervals to be 5: $[0, ..., \frac{T_r}{5}]$,$[\frac{T_r}{5}, ..., \frac{T_r}{5} * 2]$, ... ,$[\frac{T_r}{5} * (4), ..., T_r]$. Nodes that have not been visited start a timer to send a request at a random slot within the first interval $[0, ..., \frac{T_r}{5}]$. Nodes that hear this message and have already been visited once, start a timer to send a request at a random slot within the second interval $[\frac{T_r}{5}, ..., \frac{T_r}{5} * 2]$. Finally, nodes visited more than four times, start a timer to send a request at a random slot with in the last interval $[\frac{T_r}{5} * (4), ..., T_r]$. This ensures that unvisited nodes get a chance to transmit before visited nodes. Also, if a node hears another request being sent with a value of *visited* that is lower than its own, it suppresses its own request. This ensures that the number of requests being sent remains low and fairly constant irrespective of the network density.

A timer T_r is started at the token holder to accept requests for the token. The token holder picks a random unvisited node if at least one unvisited node sends a request. Otherwise, the token holder picks the node that has been least visited. The token is transferred to the chosen node. The node that receives the token marks itself as visited if it was unvisited so far. If the token is used for data aggregation, an already visited node may not add its information again to a token. This concludes the procedure for token passing using self-repelling random walks. The token is continued to pass iteratively using this procedure.

4.2 Analysis of self-repelling random walks

To begin with, let us prove a lemma that characterizes the expected geometric distance between unvisited tokens over time. First, we state the following claim regarding the uniformity in the distribution of visited nodes during the progression of a self-repelling random walk.

Proposition 4.2.1. *The distribution of visited nodes (and unvisited nodes) remains spatially uniform during the progression of a self-repelling random walk.*

Argument: Our claim is based on the analysis of uniformity in coverage of self-repelling random walks in [8] and in [33]. In [8], the variance in the number of visits per node of self-repelling random walks is shown to be tightly bounded, resulting in a uniform distribution of visited nodes across the network. More precisely, let $n_i(t, x)$ be the number of times a node i has been visited, starting from a node x. The quantity studied in [8] is the variance $\frac{1}{N}(\sum_i (n_i(t, x) - \mu)^2)$, where $\mu = \frac{1}{N}(\sum_i n_i(t, x))$. It is seen that this variance is bounded by values less than 1 even in lattices of dimensions 2048×2048. A detailed extension of this analysis for *mobile networks* is presented in Section 4.3 ,which shows the uniformity with which nodes are visited during a self-repelling random walk. We use this to infer that even after the walk started, the distribution of visited nodes (and by that token, unvisited nodes) remains uniform. The result shows that the self-repelling random walk is not stuck in regions of already visited nodes - instead, it spreads towards unvisited areas. □

We have empirically characterized the remarkable uniformity with which self-repelling random walks visit nodes in a network in Section 4.3.

Lemma 4.2.2. *There exists at least one unvisited node within h-hops of a token holder in a self-repelling random walk with probability p as long as the fraction of unvisited nodes in the network ϕ_u satisfies $\phi_u > \frac{1}{h^2 c}$.*

Proof. Recall that there are d nodes in each cell of size $\frac{R^2}{2}$ with high probability. There are $\frac{2A}{R^2}$ such cells. Given that a token is in any of these cells, there is one unvisited neighbors as long as each of the cells has at least one unvisited node. Let $n_u(t)$ denote the number of unvisited nodes at any time t. To find a lower bound on $n_u(t)$, we use the analogous coupon collector problem which studies the expected number of coupons to be drawn from B categories so that at least 1 coupon is drawn from each category[34, 35]. Using the coupon collector result[34, 35], if $n_u(t) > (\frac{2A}{R^2})(log(\frac{2A}{R^2}) + \gamma)$, then each cell has at least one unvisited node with high probability. In this case $\gamma = 0.5772$ is the Euler-Mascheroni constant. First, we note the followings:

$$\begin{aligned}
log(N) - log(\frac{2A}{R^2}) &= log(\frac{NR^2}{2A}) \\
&\geq log(\frac{2\rho N clog(N)}{2\rho N}) \ since \ R^2 = \frac{2clog(N)}{\rho} \ and \ \rho = \frac{N}{A} \\
&\geq log(clog(N)) \\
&> log(log(N)) \qquad since \ c \ > \ 1 \\
&> \gamma \qquad since \ N \ \gg e^{e^{\gamma}} \ for \ the \ scale \ of \ networks \ we \ consider
\end{aligned}$$

Therefore, having $(\frac{2A}{R^2})(logN)$ unvisited nodes can only increase the chance of having an unvisited node in each cell. Hence, the required number of unvisited nodes in the network can be written as:

$$n_u(t) > \frac{2Alog(N)}{R^2}$$

Recall $R^2 \geq \frac{2clog(N)}{\rho}$. Thus the required number of unvisited nodes can be written as:

$$n_u(t) > \frac{2N}{2c}$$

Dividing by N, the fraction of unvisited nodes required so that each node has one unvisited neighbor can be written as:

$$\phi_u > \frac{1}{c}$$

To generalize this result to the availability of an unvisited neighbor within h hops of a token holder, we consider cells of sides $\frac{hR}{\sqrt{2}}$ which have a diagonal of length hR. Thus, the required fraction of unvisited nodes so that each node has one unvisited node within h communication hops can be written as follows, where $c > 1$.

$$\phi_u > \frac{1}{h^2 c}$$

$\square$

Significance: When $h = 1$ and $c = 2$, $\frac{1}{(h^2c)} = 0.5$. Thus, as long as less than 50% of the nodes are visited, self-repelling random walks are expected to find an unvisited node within one hop with high probability. With $h = 2$, we see that until about 88% coverage, an unvisited node can be expected within a 2 hop neighborhood of a token. This highlights the extent to which bounded self-repelling random walks can cover a significant portion of the network can be without much wasted exploration and without any supporting network structures. Note also that when c increases, we expect the degree d (i.e., the number of nodes in each cell) to increase and hence progressively larger portions of the network can be visited without much wasted exploration.

Theorem 4.2.3. *Both expected cover time and the expected number of token transfer for a self-repelling random walk in a connected, mobile network of N nodes with uniform stationary distribution of node locations and a single token are $O(N(log(N)))$.*

Proof. From Lemma 4.2.2, we note that the expected number of unvisited neighbors remains greater than 1 as long as $\phi_u > \frac{1}{c}$. Thus for a fraction $(1 - \frac{1}{c})$ of the nodes, the expected steps taken by a token is 1. $\square$

Once the fraction of visited nodes exceeds $(1 - \frac{1}{c})$, a slowdown is expected because the token might be randomly traversing an area of already visited nodes. But note that for a fraction $\frac{1}{c} - \frac{1}{4c}$ nodes, there exists an unvisited node within 2 hops of a token holder and in general for a fraction $\frac{1}{(h-1)^2 c} - \frac{1}{h^2 c}$ nodes, there exists an unvisited node within h hops of the token holder with high probability. Thus, for a fraction $\frac{1}{(h-1)^2 c} - \frac{1}{h^2 c}$ nodes, an expected number of $O(h^2)$ steps is needed to find an unvisited node. Continuing up to a maximum of H hops, where $H = \sqrt{N}$, the total number of steps traversed by a token before visiting all nodes and the expected time for complete coverage is given by the following expression:

$$O\left(\left(\mathrm{N} - \frac{\mathrm{N}}{\mathrm{c}}\right) + \left(\frac{N}{c} - \frac{N}{4c}\right) * 4 + \ldots + \left(\frac{N}{(H-1)^2 c} - \frac{N}{H^2 c}\right) H^2\right)$$

$$= \mathrm{O}\left(N - \frac{N}{c} + N\left(3 + \frac{5}{4} + \ldots + \frac{(2H+1)}{H^2}\right)\right)$$

$$= \mathrm{O}\left(N - \frac{N}{c} + N\sum_{i=1}^{H} \frac{2i+1}{i^2}\right)$$

$$= \mathrm{O}(N + \mathrm{N}\sum_{i=1}^{H} \frac{2}{i} + \mathrm{N}\sum_{i=1}^{H} \frac{1}{i^2}$$

$$= \mathrm{O}(N(1 + log(H))) \qquad \{\textit{Euler harmonic series approximiation.}\}$$

$$= \mathrm{O}(N(1 + log(N)))$$

$$= \mathrm{O}(N(log(N)))$$

Now consider the region to be divided into k equi-sized areas with one token used to visit node in each area. Thus, each token is responsible for an area of $\frac{N}{k}$. Using this, we state the following corollary.

Corollary 4.2.4. *Both the expected convergence time and the average number of transfer per token in random walks in a connected, mobile network of N nodes with uniform stationary distribution of nodes and k tokens are* $O(\frac{N}{k}log(N))$.

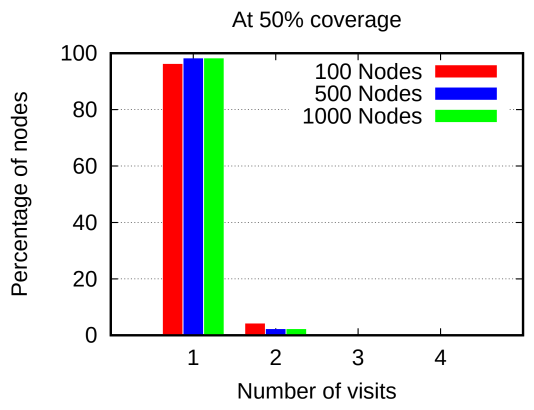
During performance evaluation, we relax the requirement that each of the k tokens are restricted to stay within a unique area. We initialize the k tokens uniformly at random across the network and validate empirically that the above bounds hold.

Using the Proposition, we observe that with $\sqrt{N}$ tokens, the expected cover time is $O(\sqrt{N}(log(N)))$. When $log(N)$ tokens are used, the expected convergence time is $O(N)$.

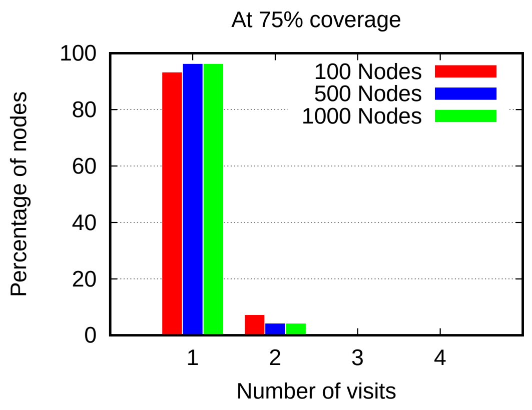
4.3 Uniformity of self-repelling random walk

First, in Fig. 4.1a, Fig. 4.1b and Fig. 4.1c,we show the number of times each node is visited when the self-repelling random walk has finished visiting 50% of the nodes, 75% of the nodes and 85% of the nodes. We observe that most of the nodes are just visited once and this result holds even at 1000 nodes. These graphs *highlight the uniformity with which nodes are visited as self-repelling random walks progress.* The self-repelling random walk is not stuck in regions of already visited nodes - instead, it spreads towards unvisited areas. Otherwise, one would have observed more duplicate visits to the previously visited nodes. In Fig. 4.1d, we analyze the distribution of number of visits at each node when 100% coverage is attained. Here, we see that most nodes are visited 2 or 3 times and the distribution falls off rapidly after that.

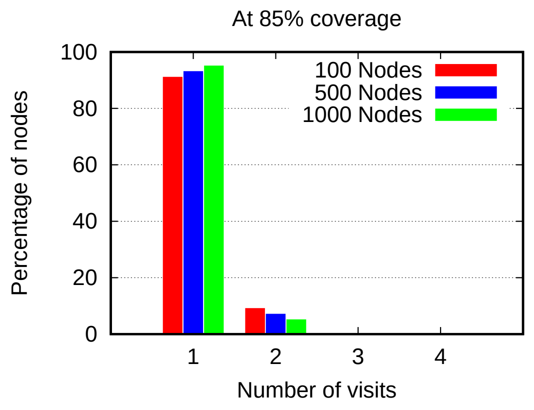
We then compare the uniformity in coverage with that of pure random walks. In Fig. 4.2a, we plot the number of visits to each node until all nodes are visited at least once for a 500



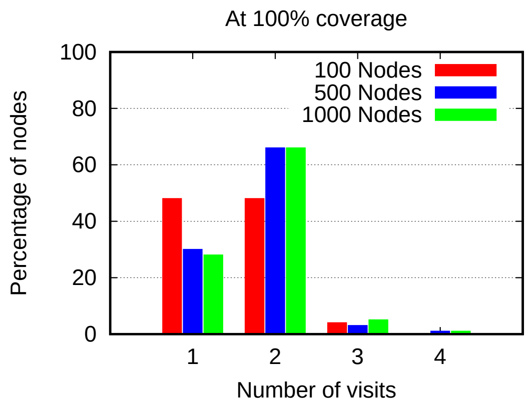
(a) Distribution of number of visits at 50% coverage



(b) Distribution of number of visits at 75% coverage

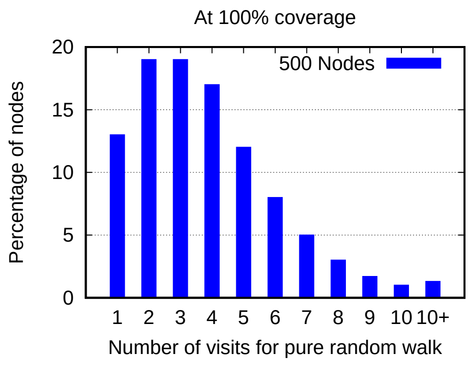


(c) Distribution of number of visits at 85% coverage

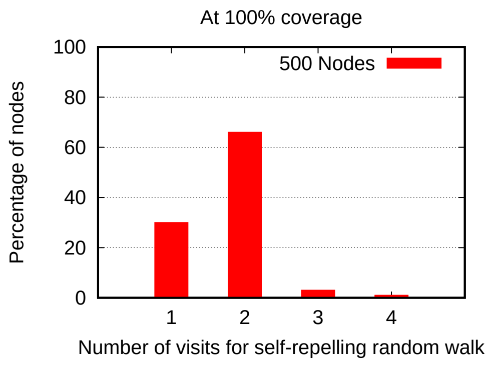


(d) Distribution of number of visits at 100% coverage

Figure 4.1: Distribution of number of visits at each node at different stages of exploration of a self-repelling random walk (network size 100,500 and 1000 nodes)



(a) Distribution of number of visits to each node in a pure random walk



(b) Distribution of number of visits to each node in a self-repelling random walk

Figure 4.2: Comparison of coverage uniformity with pure random walks

nodes network. In comparison with Fig. 4.2b, we observe that the tail of the distribution is much longer and the number of duplicate visits is much higher for pure random walks.

4.4 Variant of self-repelling random walks

There is a potential variation of self-repelling random walks: binary self-repelling random walks. In binary self-repelling random walks, each node only stores if the node has been visited so far or not. Thus, when all neighbors are visited, it reduces to pure random walks. In our experiments, we show that binary self-repelling random walks do speed up random walks but they are slower than self-repelling random walks by constant factor.

Chapter 5

Gradient Biased Random Walks

In this chapter, we introduce a pull based idea to overcome long tail observed in self-repelling random walks and provide an analysis of convergence time and total messages. Self-repelling random walks successfully speeds up random walks but the token can stuck in the regions of visited nodes while there are still unvisited nodes to be explored. To prevent this, we set up short, temporary multi-hop gradients to pull the token towards unvisited nodes. We show analytically that this yields a cover time of O(N). Thus, the order of convergence improves by a factor of log(N). In doing so, the self-repelling random walk introduces a gradient message overhead of O(Nlog(N)), to pull tokens towards unvisited nodes. Nevertheless, this overhead is compensated by a reduction in the required number of token transfers. In fact, our simulations show that the ratio of token transfers to that of node size, i.e., the exploration overhead of gradient biased random walk remains constant (irrespective of network size) and close to 1.

5.1 Gradient Biased Random Walks: Protocol description

Gradient Biased Random Walks consist of two components: (i) token passing, and (ii) gradient setup. To realize these components, each node stores three variables, *visited*, *holder* and *level*. The variable *visited* is a boolean value that tracks whether a node has been visited by any of the tokens; *visited* is initially *false* at all nodes. When a token first arrives at a node, *visited* is set to *true*. Tokens are assumed to be initiated at a random set of nodes. All nodes in which a token is initiated are marked as visited by default and the token value is initialized to the data at the corresponding node. The variable *holder* is used to identify nodes that currently hold a token. When a gradient bias is used, each node also participates in a gradient setup process to attract tokens towards unvisited nodes. To do so, each node uses the state variable *level* where $0 \leq level \leq 1$. Nodes that are unvisited are at $level = 1$. Nodes that hold a token set *level* to 0 as soon as they receive a token.

Token passing with gradient bias

For random walks with a gradient bias, a token holder announces that it has a token. Nodes that hear this message send a request at a random slot within a chosen interval T_r. A timer T_r is started at the token holder to accept requests for the token. All nodes with $level > 0$ randomize their response time and reply to the token announcement along with their current level. Nodes with $level > 0$ are nodes that have either not been visited ($level = 1$) or nodes that have been visited and are now part of a gradient ($0 < level < 1$). If a node hears another request being sent with a *level* greater than itself, it suppresses its own request. The token holder stores all requests received during time T_r. The replies are sorted based on the level of the requesters and the token is sent to the node with the highest level. When multiple requesters exist with the same level, the token recipient is chosen randomly among that set. Thus, if any unvisited node requests a token, the token will be sent to that node. If all nodes that have currently requested the token have been visited, the token is sent to the node with the highest value of level, which is expected to be the node that is closest to an unvisited node. As soon as a token reply has been sent, the node resets holder to 0. The following section describes how short multi-hop gradients are setup to attract tokens towards unvisited nodes.

Gradient setup

During the operation of gradient biased random walk, a token can get stuck inside a region where all its neighbors have already been visited. To recover from such a scenario, a gradient is setup in the network to attract tokens towards unvisited nodes, i.e., nodes with *level* = 1 (See Fig.1.3). This is done as follows. Nodes with *level* = 1, for which none of their neighbors currently hold a token and have at least one neighboring node with *level* = 0, initiate a gradient setup by broadcasting a gradient message. Nodes with *level* = 0 that receive a gradient message update their level to half of sender's level and rebroadcast the gradient message. Thus, gradient broadcasts propagate only till the region where nodes with non-zero level are present, filling up the gap between an unvisited node and other nodes with non-zero levels.

Gradient refresh

To account for node mobility, gradients have to be periodically refreshed. To do so, when a node updates its level from zero to some non-zero value < 1, it starts a timer proportionate to the new level and when the timer expires it resets its level back to 0. Thus, nodes with higher values of level are refreshed slower than smaller values. This heuristic is based on two reasons. (1) Gradients should preferably not be refreshed before a token is able to climb up a gradient and reach an unvisited node. By refreshing at a rate proportional to the value of level, a token gets more time to reach closer to the source of the gradient. (2) Nodes that are far away from an unvisited node (closer to the bottom of the gradient) should prevent blocking of gradient setup from unvisited nodes that are nearby, for extended periods of time.

5.2 Analysis of gradient biased random walks

.

Theorem 5.2.1. *Both expected cover time and the expected number of token transfer with gradient bias in a connected, mobile network of N nodes with uniform stationary distribution of nodes and a single token are $O(N)$*

Proof. Similar to the analysis in Theorem 4.2.3, for a fraction $(1 - \frac{1}{c})$ of the nodes, the expected distance traveled by a token is 1. However, once the fraction of visited nodes exceeds $\frac{1}{c}$, the gradients will be used to pull the token towards unvisited nodes. Now note that a fraction $\frac{1}{(h-1)^2c} - \frac{1}{h^2c}$ nodes, the expected distance traveled by a token is h.

Continuing up to a maximum distance of $H = \sqrt{N}$, the total average distance traversed by a token before visiting all nodes and the average time for complete coverage is given by the following expression:

$$O\left(\left(\mathrm{N} - \frac{\mathrm{N}}{\mathrm{c}}\right) * 1 + \left(\frac{N}{c} - \frac{N}{4c}\right) * 2 + \ldots + \left(\frac{N}{\left(\sqrt{N}-1\right)^2 c} - \frac{N}{Nc}\right)\sqrt{N}\right)$$

$$= \mathrm{O}\left(N + \frac{N}{c}\left(1 + \frac{1}{4} + \frac{1}{9} + \ldots + \frac{1}{N}\right)\right)$$

$$= \mathrm{O}\left(N + \frac{N}{c}\sum_{i=1}^{\sqrt{N}} \frac{1}{i^2}\right)$$

$$= \mathrm{O}\left(N(1 + \frac{1}{c})\right)$$

$$= \mathrm{O}(N)$$

□

In comparison to Theorem 4.2.3, we note a speed up by a factor of log(N).

Similar to Corollary 4.2.4, we note that with k tokens, the cover time of random walk with gradient bias is $O(\frac{N}{k})$. Thus, with $\sqrt{N}$ tokens, the expected cover time is $O(\sqrt{N})$. When log(N) tokens are used, the expected cover time is $O(\frac{N}{log(N)})$. We also verify these results empirically.

Theorem 5.2.2. *The expected gradient message overhead in random walk with gradient bias in a connected, mobile network of N nodes and k tokens is* $O(N\frac{log(N)}{k})$.

Proof. Following the lines of Theorem 4.2.3, we note that for a fraction $(\frac{1}{c} - \frac{1}{4})$ of the nodes, the average gradient set up cost will be $4c$. And, for a fraction $(\frac{1}{((\sqrt{p}-1)^2c)-\frac{1}{pc}})$ of the nodes, the gradient setup cost will be pc, where $pc = \frac{N}{k}$. The result then follows from summing up the series as shown in the proof of Theorem 4.2.3. □

Thus, we note that when using gradient bias, there is an extra overhead to pull the tokens towards unvisited nodes, but this is compensated by reduction in the number of required token transfers and reduction in convergence time. Moreover, the gradient message overhead decreases linearly with number of tokens.

Chapter 6

EZ-AG:Push assisted self-repelling walk

In this chapter, we introduce a push-based self-repelling random walks (EZ-AG) to avoid long tail observed in self-repelling random walks. EZ-AG achieves this by effectively reducing the total number of nodes to be visited to complete aggregate. The key idea in EZ-AG is *single step push phase at each node.* By allowing a node to hold multiple information, self-repelling random walks can collect all information (complete data aggregation) without visiting all the nodes in the network.In our experiment, it is observed that EZ-AG finishes data aggregation before self-repelling random walks start to slow down. EZ-AG can be used for duplicate-insensitive data aggregation in MANETs and our analysis shows it outperforms existing gossip based protocols.

6.1 ODI (order and duplicate insensitive) synopsis

EZ-AG is designed for duplicate-insensitive data aggregation. However, EZ-AG can be used for duplicate-sensitive data aggregation by utilizing ODI synopsis. ODI synopsis is designed so that the resulted aggregate is insensitive to the duplication of one item [10]. For example, MIN and MAX are already in the ODI synopsis as the final result is the same irregardless of the order of aggregation or amount of duplicates present. However, Sum and Average are not naturally in the ODI synopsis. Therefore, the ODI synopsis for each statistical function needs to be defined. To define ODI synopsis, three distinct functions (the synopsis generator, synopsis fusion and synopsis evaluation) and the form of synopsis need to be defined. All nodes use the synopsis generation function to convert their data into synopsis. All nodes route the synopsis to another node(s). The node receiving a synopsis from other node(s) uses the synopsis fusion function to combine its own synopsis and received synopsis. Then, it transfers the combined synopsis. This operation is repeated till all synopsis reach to the root node. The root node uses the synopsis fusion function to combine its own synopsis and received synopsis. Then, the synopsis evaluation function is applied to this combined synopsis to obtain the final aggregate value. In [10], several examples on how to define the synopsis, synopsis fusion function, synopsis generation function, and synopsis evaluation function for different statistical functions can be found. In the rest of this subsection, we briefly summarize algorithms found in [10] for generating the ODI synopsis for some statistical functions.

6.1.1 Count

The goal of Count is to find the total number of distinct items in the network. The item could be a node that satisfies a pre-defined predicate or some other values obtained from sensors. To achieve counting, we can utilize the approximate counting algorithm[36]. This algorithm allows us to approximate the number of distinct items in the multi-set.

This algorithm works as follows. Each element in the set selects one random number r between [1,k]. (k is the maximum value of the distinct item). Each element calls the CT function with the selected random number r. The CT(r) function repeatedly flips coins r times and returns the number of coin flips required to see the head of the coin or $r - 1$ if no head appears in r trials. Then, it sets CT(r) bit of the synopsis vector to 1. (The synopsis vector for count is the log(k) bit vector). This operation is repeated for all elements in the network. In EZ-AG, nodes initiating a token store its synopsis vector in the token and forward the token to one of its neighbor. The neighbor received tokens, then updates stored synopsis using the synopsis fusion function. (This combines its synopsis vector and the synopsis vector in the token using bitwise OR operation.) To obtain the approximate count, each node applies the fusion evaluation function to the synopsis. (When EZ-AG terminates self-repelling random walk, each token holder floods its synopsis vector. Thus, all nodes in the network obtain synopsis vectors from all token holders.) The fusion evaluation function searches the most significant bit i of synopsis vector which has been set to 1. Then returns the value $(\frac{2^{i-1}}{0.77351})$.

6.1.2 Sum

The sum can be found by extending the approximate count algorithm. Finding the sum can be turned into a counting problem by replacing the node with the value k with k nodes having the value of 1. However, this approach may require lots of energy. For node n with value k, the approximate counting algorithm needs to be applied k times. To avoid this problem, 2^j can be used in place of $\frac{2^{i-1}}{0.77351}$ where j is the highest order bit set and i is lowest order bit set in the synopsis vector [37]. The synopsis for the sum is a vector whose length is $loglog(NK)$; N is the network size and K is the largest k possible. By using the highest order bit, the synopsis generation function can be modified. Now, the synopsis generation function needs to select a random number r and calculate $\left\lceil -log_2(1 - r^{\frac{1}{k}}) \right\rceil$ which is essentially finding the probability of having the jth bit as the highest order bit. This also affects the synopsis fusion function. The value we need is j which is the index of the highest order set. Thus, the fusion function needs to compare two synopsis vectors and selects one that yields larger value. Please note that this algorithm to find the sum can be used to find other statistics such as average.[38].

6.1.3 Uniform sample of sensor reading

This aggregate tries to find the uniform sample of X nodes in the network ($X < N$), randomly sampling a value that each node holds by picking X elements uniformly random. The components of this algorithm are defined as follows. The synopsis will be a tuple. The synopsis generation function outputs ($value$,r, id) where $value$ is the value stored at the node specified by the id and r is a random number [0,1] obtained from the uniform distribution. The synopsis fusion function takes union of two synopsis. After taking union of two tuples, only X tuples are selected using the r value. (Sort X tuples in decreasing order using r values and take X tuples first). If the union resulted in less than X tuples, everything is kept. Finally, the synopsis evaluation function creates a set of values contained in X tuples.

6.2 EZ-AG: Protocol description

The node requesting the aggregate first initiates a flood in the network to notify all nodes about the interest in the aggregate. Note that each node broadcasts this flood message exactly once. This results in N messages.

Once a node receives this request, it *pushes* its state to its neighbors. Each node uses the data received from its neighbors to compute an aggregate of the state of all its neighbors. Note that the push phase also requires exactly N messages.

Soon after the initiator sends out an aggregate request, it also initiates a token to perform a self-repelling random walk. A node that has the token broadcasts an *announce* message. Nodes that receive the announce message reply back with a token *request* message and include the number of times they have been visited by the token in this request. The node that holds the token selects the requesting node which has been visited least number of times (with ties broken randomly) and transfers the token to that node. This token transfer is repeated successively. Note that nodes which hear a token announcement schedule a token request at a random time t_r within a bounded interval, where t_r is proportional to the number of times that they have been visited. Thus, nodes that have not been visited or visited fewer times send a request message earlier. When a node hears a request from a node that has been visited fewer or same number of times, it suppresses its request. Thus, the number of requests received for a token announcement remains fairly constant and irrespective of network density.

In the following section, we prove analytically that the aggregate can be computed from all nodes in the network with high probability in $O(N)$ token transfers. In the empirical evaluation, we show that the median number of token transfers is actually only kn, where $0 < k < 1$, and k is unaffected by network size. Thus, the median exploration overhead is less than 1. One can use this observation to terminate the self-repelling random walk after exactly N steps and with high probability one can expect that data from all the nodes has been aggregated.

Once the aggregate has been computed, the result can simply be flooded back to all the nodes. This requires $O(N)$ messages. Another potential solution (when aggregate is only required at a base station) is to transmit the aggregated tokens using a long distance transmission link (such as cellular or satellite links) in hybrid MANETs where the *long links* are used for infrequent, high priority data.

The protocol is thus extremely simple, requires very little state maintenance (each node only remembers the number of times it has been visited), requires no network structures or clustering.

6.3 Analysis of EZ-AG

In this section, we first show that the aggregation time and message overhead for push assisted self-repelling random walks is $O(N)$. We consider a static network for our analysis. In section 8.4.3, we evaluate the protocol under different mobility models and verify that the results hold even in the presence of mobility.

Theorem 6.3.1. *The required number of messages for data aggregation by EZ-AG in a connected, static network of N nodes with uniform distribution of node locations is $O(N)$.*

Proof. We note that the aggregation request flood and the result dissemination flood require $O(N)$ messages. During the push phase, each node broadcasts its state once and this also requires only N messages. Now, we analyze the self-repelling random walk phase.

Consider the region to be divided into square cells of sides $\frac{R}{\sqrt{2}}$ (see Fig 6.1). Thus the diagonal of each such cell is the communication range R. Recall from our system model that each such cell has $\theta(logN)$ nodes with high probability at all times and there are $O(\frac{N}{log(N)})$ such cells. Therefore, at the end of the push phase, each node has aggregated information about its $\theta(logN)$ cell neighbors. Also note that the network can be divided into $\theta(\frac{N}{log(N)})$ sets of nodes that each contain information about $\theta(log(N))$ nodes within their cell. Therefore, the self-repelling random walk has to visit at least one node in each cell to finish aggregating

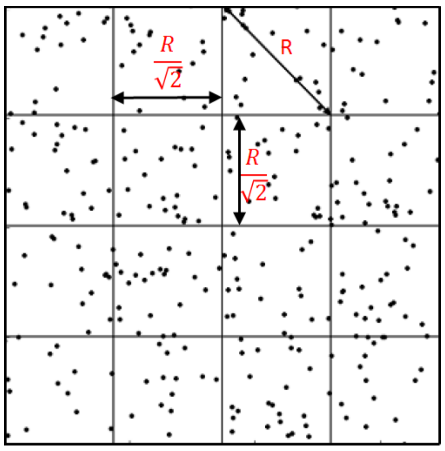


Figure 6.1: Proof synopsis: Consider the region divided into square cells with diagonal size R. At the end of single step push phase, each node has information about all nodes in its cell. So it is sufficient for the token (performing a self-repelling random walk) to visit one node in each cell to finish aggregation.

information from all nodes.

To analyze the number of token transfers required to visit at least one node in each cell, we use the analogous coupon collector problem (also known as the double dixie cup problem) which studies the expected number of coupons to be drawn from B categories so that at least 1 coupon is drawn from each category [35]. To ensure that at least 1 coupon is drawn from each category with high probability, the required number of draws is $O(BlogB)$. Using this result and the fact that a self-repelling random walk traverses a network uniformly, we infer that $O((\frac{N}{log(N)}) * log(\frac{N}{log(N)}))$ token transfers are needed to visit at least 1 node in each of the $\theta(\frac{N}{log(N)})$ cells.

Note that $log(N) > log(\frac{N}{log(N)})$. Hence, the required number of messages for the push assisted self-repelling random walk based aggregation protocol is $O(\frac{N}{log(N)} * log(N))$, i.e., $O(N)$. □

Note that in the presence of mobility, the node locations with respect to cells may not be preserved during the push phase. Therefore the generation of $\theta(\frac{N}{log(N)})$ identical partitions of network state as described in the above analysis may not exactly hold. However, in section 8.4.3 we empirically ascertain that kN token transfers (where $k < 1$) are still sufficient to aggregate data from all nodes even in the presence of mobility. In fact, we observe that *the required token transfers actually decrease with increasing speed*, indicating that data aggregation using self-repelling random walks is *actually helped by mobility*.

It follows from the above result that the total time for aggregation is also $O(N)$. The impact of network effects such as collisions on the message overhead and aggregation time (if any) will be evaluated in Section 8.4.

Chapter 7

Hierarchical aggregation

When a network is quite large, providing each node with only a single aggregate for the entire network may not be sufficient. It may be more useful to provide each node with a distance-sensitive multi-resolution aggregate, i.e., to provide aggregate information about neighborhoods of increasing sizes around itself. In this chapter, we describe how EZ-AG can be extended to provide such multi-resolution synopsis of nodes in a network with only $O(Nlog(N))$ messages.

Existing techniques for such hierarchical aggregation require $O(Nlog^{5.4}(N))$ messages [11]. Thus, EZ-AG offers a poly-logarithmic factor improvement in terms of number of messages for hierarchical aggregation. Moreover, EZ-AG can also be used to generate hierarchical aggregates that are distance-sensitive in refresh rate, where aggregates of nearby regions are supplied at a faster rate than farther neighborhoods.

7.1 Protocol description

We divide the network into square cells at different levels (0, 1, ... P) of exponentially increasing sizes (shown in Fig. 7.1). At the lowest level (level 0), each cell is of sides $\frac{R}{\sqrt{2}}$. Recall from our system model that each such cell has $\theta(log(N))$ nodes with high probability. For simplicity, let us denote $\theta(log(N))$ by the symbol δ. Thus, there are $\frac{N}{\delta}$ cells at level 0. Note that 4 adjoining cells of level i constitute a cell of level $i+1$. Thus, each cell at level j has $\delta 4^j$ nodes with high probability. At the highest level P, there is only one cell with all the N nodes. Note that $P = log_4(\frac{N}{\delta})$. At any given time, a node belongs to one cell at each level.

To deliver multi-resolution aggregates, we introduce a token and execute EZ-AG at each cell at every level. A token for a given cell is only transferred to nodes within that cell and floods its aggregate to nodes within that cell. Thus, there are $\frac{N}{\delta}$ instances of EZ-AG at level 0 and each instance computes aggregates for δ nodes, i.e., $\theta(logN)$ nodes.

The computation and dissemination of aggregates by different instances of EZ-AG are not synchronized. Thus, a node may receive aggregates of different levels at different times. Also, since the nodes are mobile, an aggregate at level l received by a node at any given time corresponds to the cell of the same level l in which it resides at that instant.

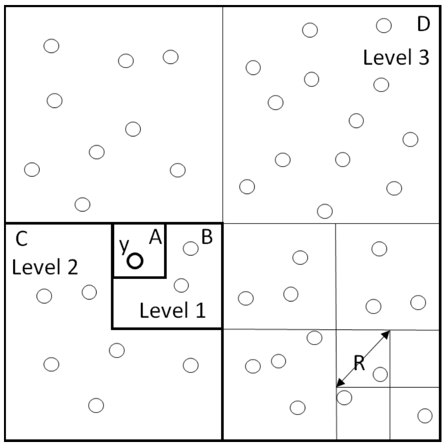


Figure 7.1: Extension of EZ-Ag to deliver multi-resolution aggregates: The network is partitioned into cells of increasing hierarchy where the cell at smallest level is of diagonal R. The node y shown in the figure would receive an aggregate corresponding to one cell at each level that it belongs to. In this case, it would receive aggregates for cells A, B, C and D. The largest cell D consists of the entire network.

7.2 Analysis of Hierarchical EZ-AG

Theorem 7.2.1. *An ODI aggregate at level j can be computed using hierarchical EZ-AG in $O(4^j\delta)$ time and messages.*

Proof. Note that each cell at level j contains $\theta(4^j\delta)$ nodes with high probability. Therefore, using Theorem 6.3.1, EZ-AG only requires $O(4^j\delta)$ time and messages to compute aggregate within the cell. □

We note from the above theorem that aggregates at level 0 can be published every $O(\delta)$ time, aggregates at level 1 can be published every $O(4\delta)$ time and so on. Thus, aggregates for cells at smaller levels can be published exponentially faster than those for larger cells. Thus, if the tokens repeatedly compute an aggregate and disseminate within their respective cells, EZ-AG can generate hierarchical aggregates that are distance-sensitive in refresh rate, where aggregates of nearby regions are supplied at a faster rate than farther neighborhoods.

Theorem 7.2.2. *Hierarchical EZ-AG can compute an ODI aggregate for all cells at all levels using $O(Nlog(N))$ messages.*

Proof. Note that a cell at level 0 contains δ nodes and there are $\frac{N}{\delta}$ such cells. The aggregate for cells at level 0 can be computed using $O(\delta)$ messages.

In general, there are $\frac{N}{4^j\delta}$ cells at level j and aggregates for these cells can be computed using $O(4^j\delta)$ messages. Summing up from levels 0 to P, the total aggregation message cost (M) for hierarchical EZ-AG can be computed as follows.

$$M = O\left(4^0 * \frac{N}{4^0\delta} + 4^1\delta * \frac{N}{4^1\delta} + ... + 4^{log_4 N}\delta * \frac{N}{4^{log_4 N}\delta}\right)$$

$$= \mathrm{O}\left(\sum_{j=0}^{P}\left(4^j\delta\frac{N}{4^j\delta}\right)\right)$$

$$= \mathrm{O}\sum_{j=0}^{P}(N)$$

$$= \mathrm{O}\left(Nlog(N)\right) \quad since, P = log_4(\frac{N}{\delta})$$

□

Thus, hierarchical EZ-AG can compute an ODI aggregate for all cells at all levels using $O(Nlog(N))$ messages.

Chapter 8

Performance Evaluation of different types of Random Walks

In this chapter, we systematically evaluate the performance of different types of random walks using simulation in ns-3. We set up MANETs ranging from 100 to 4000 nodes using the network models described in section 3.1. Nodes are deployed uniformly in the network with a deployment area and communication range such that $R^2 = \frac{4log(N)}{\rho}$. Thus, the network is geo-dense with c=2, i.e., each node has on average $2log(N)$ neighbors with high probability. We test such networks in our simulations with the following mobility models: 2-d random walk, random waypoint, and Gauss-Markov (described in 3.2). The average node speeds range from 3 to 21 *m/sec.*

8.1 Comparing random walks: Pure vs self-repelling vs gradient biased self-repelling random walks

In this section, we compare all three random walks pure random walks, self-repelling random walks, and gradient biased self-repelling random walks under 2-d random walk mobility model. The difference in convergence characteristics is illustrated, where we show the convergence pattern in a single randomly chosen trial of 500 nodes. In this particular trial, we observe that until around the 80% mark, the self-repelling random walk proceeds on par with gradient biased self-repelling random walk, but it slows down slightly after this point.

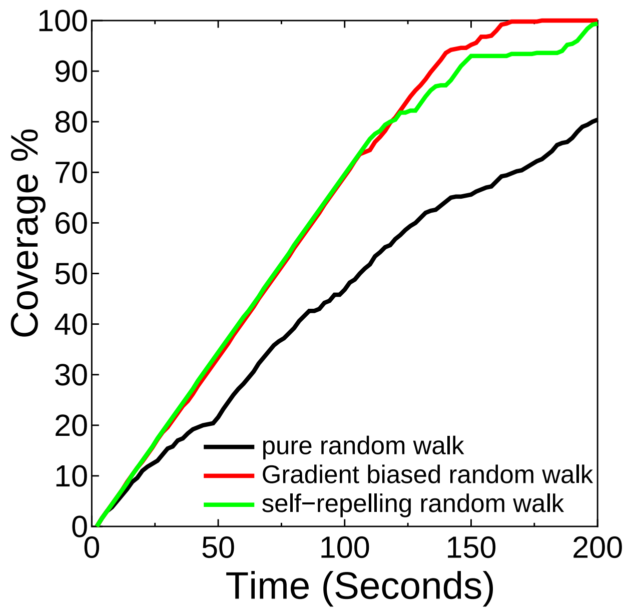
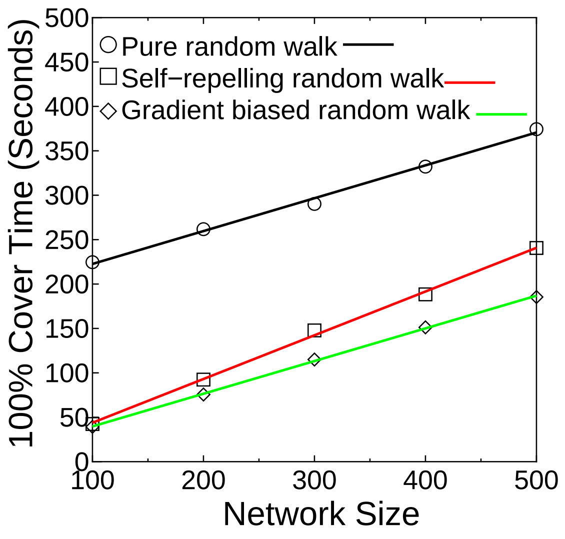


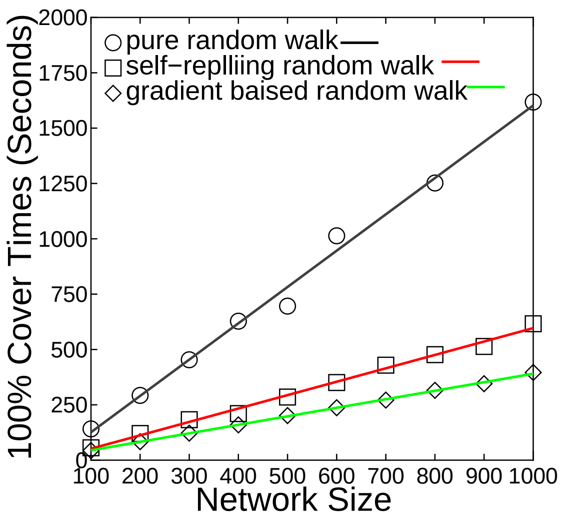
Figure 8.1: Convergence pattern on network of 500 nodes for a given trial with and without bias

This is because, until this point the self-repelling random walk enables a token to find an unvisited node directly (within 1 hop) and there are very few wasted explorations. A more pronounced slowdown for self-repelling random walk is noticed around 90% , whereas a pure random walk is slow throughout. After this point, there are wasted explorations in finding unvisited nodes when only self-repelling is used, where as the self-repelling random walk with gradient proceeds at a uniform rate throughout.

The average and 95% percentile of 100% cover time for three types of random walks is shown Fig. 8.2. As we can see, self-repelling random walks and gradient biased self-repelling random walks successfully speeds up the random walks. Also, the gradient biased self-repelling random walks achieves 100% coverage faster. This shows that the self-repelling random walks successfully pulled tokens toward the unvisited nodes.



(a) Convergence time vs network-size (average)



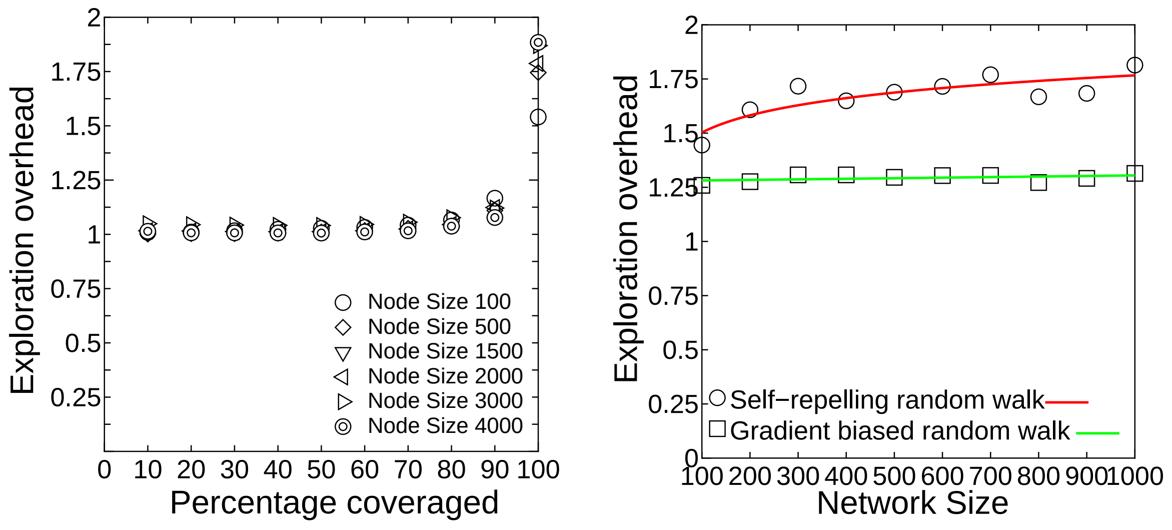
(b) Convergence time vs network-size (95% percentile)

Figure 8.2: Analysis of exploration overhead

8.2 Analysis of Self-repelling random walk vs Gradient biased self-repelling random walk

In this section, we compare self-repelling random walks and gradient biased self-repelling random walks using exploration overhead and cover time (time required to visit all nodes in the network) as our metrics. The cover time gives us more direct understanding of how long each protocol requires to compute total aggregation. However, the cover time is quite implementation specific and incorporates messaging latency in the wireless network. For instance, in our implementation each transaction (i.e., each iteration of token announcement, token requests and token passing) took on average 250 *msec*. But this number could be much smaller using methods such as [39] that use collaborative communication for estimating neighborhood sizes that satisfy given predicate. Thus, we also use the exploration overhead which indicates the number of token passing transactions required for coverage.

The 100% cover time is shown in Fig. 8.2a and Fig. 8.2b.



(a) Exploration overhead as a function of percentage of nodes visits for self-repelling random walks

(b) Exploration overhead at 100% coverage as a function of network size

Figure 8.3: Analysis of exploration overhead

8.2.1 Convergence characteristics

We analyze exploration overhead for self-repelling random walks in more detail in Fig. 8.3a which plots the ratio of token transfers to the number of visited nodes at different stages of coverage for self-repelling random walks (averaged over multiple trials). This ratio captures the wasted explorations where a token is repeatedly passed to already visited nodes. We see that with self-repelling random walks, the ratio stays low and close to 1 until about 80% mark and then starts rising. The observation matches our result in Lemma 4.2.2. But the token passing overhead for gradient biased self-repelling random walks remains close to 1 throughout without any sharp rise at all network sizes as seen in Fig. 8.3b. In Fig. 8.3b, we compare the exploration overhead at different network sizes with 100% coverage. We observe that for self-repelling random walks, this ratio grows at $log(N)$, while for the gradient bias it is almost flat, matching our results in Theorem 4.2.3 and Theorem 5.2.1. Redundant token passes are very low with gradient bias.

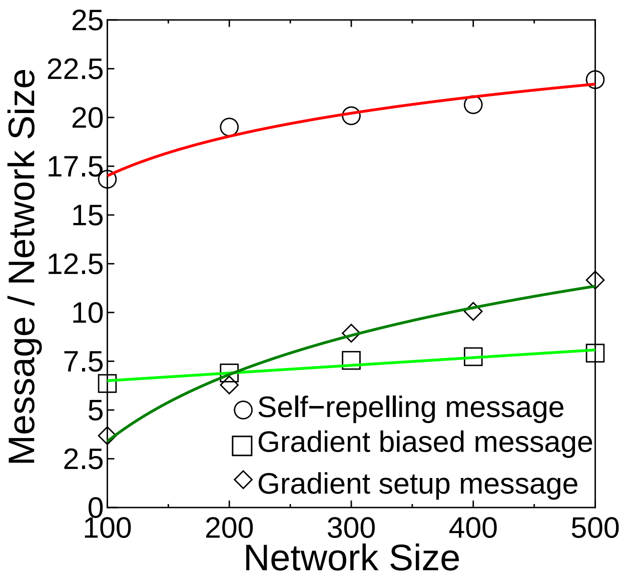
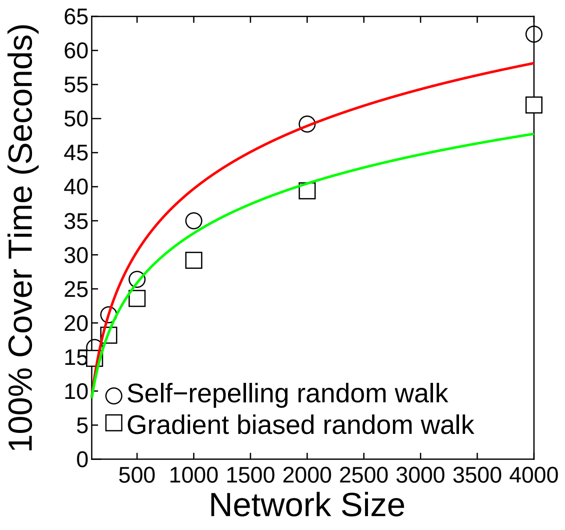


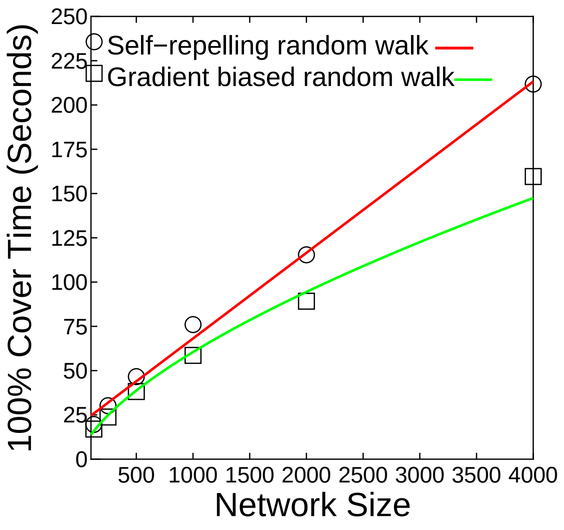
Figure 8.4: Analysis of message overhead for self-repelling random walk with and without gradient

8.2.2 Message overhead

In Fig. 8.4, we compare the message overhead of self-repelling random walks and gradient biased self-repelling random walks. To highlight the log factor overhead, we show messages normalized by network size on y-axis and network size on x-axis. The values for token message with gradient bias stay constant indicating that the token messages grow linearly with network size as the values are normalized. The curve for self-repelling token messages and the gradient setup messages grows at $log(N)$, indicating a $Nlog(N)$ messaging overhead. Note that the gradient setup cost in gradient biased random walk is compensated by a significant reduction in the required token messages compared to the self-repelling scheme. These results show that gradient biased self-repelling random walks achieve superior performance both in terms of cover time and message overhead over that of self-repelling random walks, despite the use of short multi hop gradients.



(a) Cover time vs network-size ($\sqrt{N} tokens$)



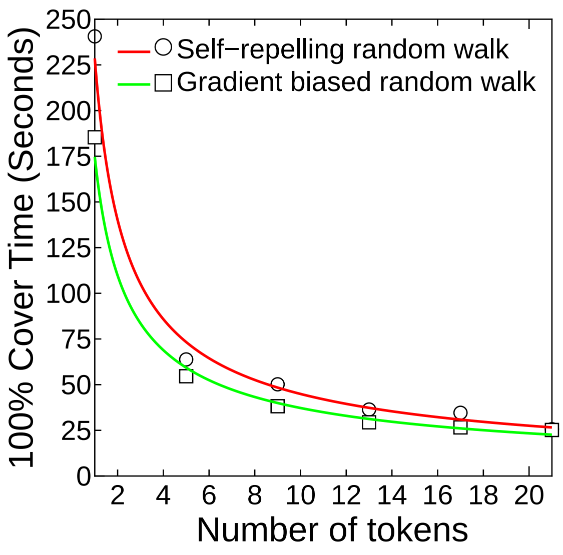
(b) Cover time vs network-size ($\log_2 N tokens$)

Figure 8.5: Analysis of cover time with multiple tokens

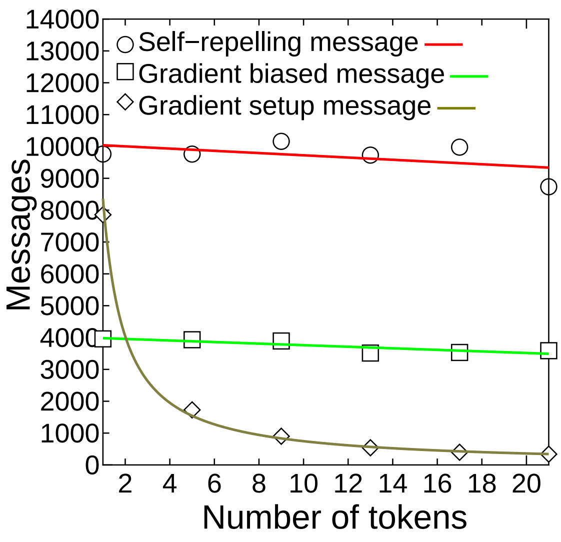
8.2.3 Multiple tokens

First, we quantify the impact of using $\sqrt{N}$ tokens and $log_2(N)$ tokens. Fig. 8.5a shows the cover time with $\sqrt{N}$ tokens in the network for self-repelling random walks and gradient biased self-repelling random walks. The network sizes that we simulate are 125, 250, 500, 1000, 2000, and 4000 nodes. The corresponding number of tokens used in the network is 11, 15, 22, 31, 42, and 63 respectively. We observe that the coverage time grows only at $O(\sqrt{N})$, matching our analysis. In Fig. 8.5b, we show the impact of using $log_2(N)$ tokens. The network sizes that we simulated are 125, 250, 500, 1000, 2000, and 4000. The corresponding number of tokens used in the network is 7, 8, 9, 10, 11, and 12 respectively. Here, the trend is observed to be linear.

Next, in Fig. 8.6a, we analyze the cover time as a function of number of tokens. The network size is 500 and the number of tokens is varied from 1 to 22. We notice that the cover time decreases linearly with the number of tokens, matching our analysis. In Fig. 8.6b, we compare the messaging overhead for the same scenario. For gradient biased self-repelling random walks, we see that the gradient message overhead decreases linearly with the number of tokens. The token passing overhead for gradient bias stays roughly constant, because there is very little overhead to begin with. The token passing overhead for self-repelling random walks decreases linearly with number of tokens.



(a) Cover time vs number of tokens (N = 500)

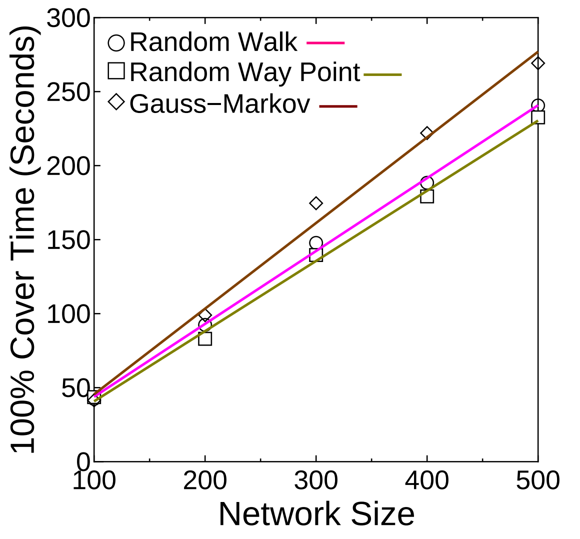


(b) Messages vs number of tokens (N = 500)

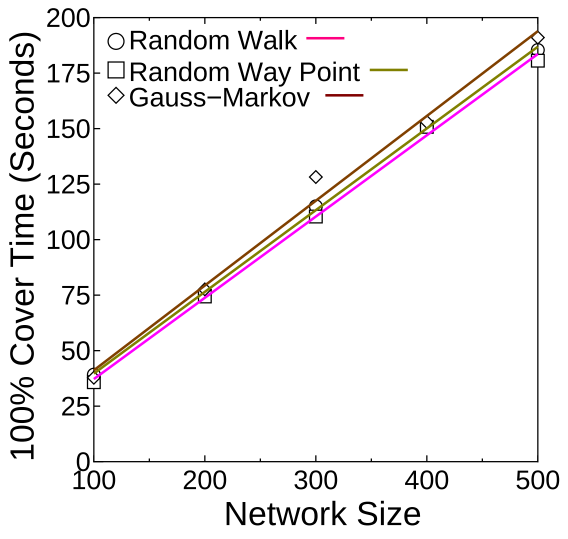
Figure 8.6: Analysis of cover time and message overhead for N = 500

8.2.4 Impact of mobility model

We simulated gradient biased random walks, and self-repelling random walks under other mobility models, random way and Gauss Markov. The node speeds remain in the range of 2 - 4 *m/sec*. For a random way point, the pause time is set to 2 seconds between successive changes. In the Gauss Markov model, where motion characteristics are correlated with time, tuned with a parameter α. (We set $\alpha = 0.75$). Velocity and direction are changed every 1 second in the Gauss Markov Model. For the random walk mobility model, it is known that the stationary distribution of the nodes is almost uniform [29]. However, for the random waypoint model on a 2-d network, it is known that the uniformity distribution of nodes is not maintained (nodes tend to cluster towards the center). The uniformity properties of Gauss Markov model are unknown. From Fig. 8.7a and 8.7b, we observe that the cover time indicating robustness with respect to mobility model especially for gradient biased self-repelling random walks.



(a) Impact of node mobility (Self-repelling random walks)



(b) Impact of node mobility (Gradient biased self-repelling random walks)

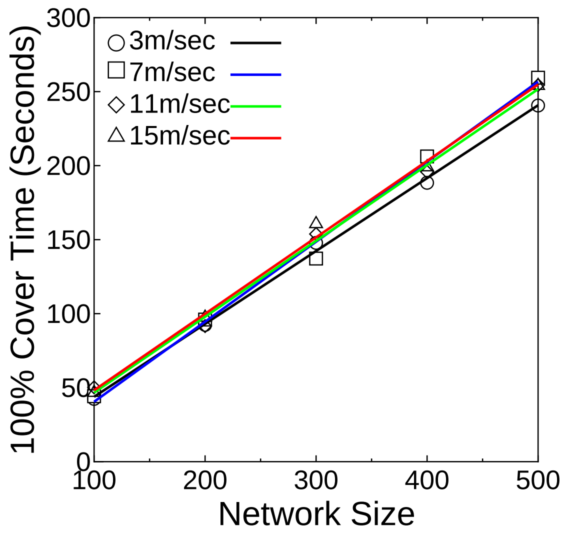
Figure 8.7: Impact of mobility

8.2.5 Impact of node speed

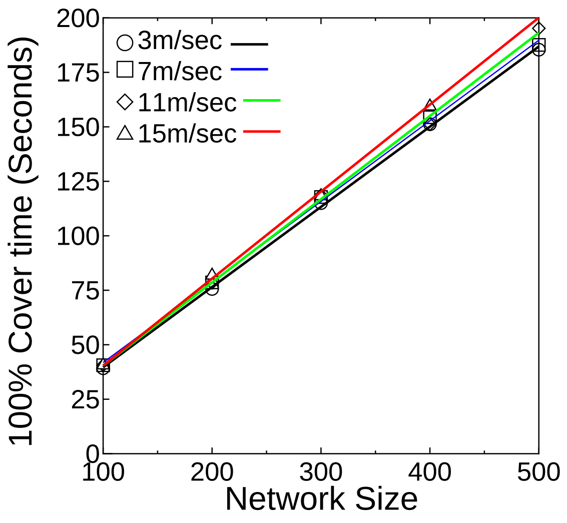
To highlight that gradient biased self-repelling random walks are robust with respect to the rate of mobility induced link changes, we simulated gradient biased self-repelling random walks under different node speeds. As observed in Fig. 8.8, even as the average node speed increases to 15 m/sec, the cover time remains quite steady.

8.2.6 Gradient biased random walk in sparse networks

In this section, we evaluate self-repelling random walks and gradient biased self-repelling random walks when the geo-dense property of network deployment does not hold. In other words, we choose a communication range that is constant (irrespective of network size) and does not meet the connectivity threshold. Thus, the network may be partitioned at times and the node degree may not be uniform at all times. Specifically, we have chosen $R^2 = \frac{4}{\rho}$ in the following results. In Fig. 8.9a, we observe that the number of token transfers exhibits a similar trend to that of geo-dense networks for both self-repelling random walks and gradient biased self-repelling random walks. In comparison with Fig. 8.5a, we observe



(a) Cover time vs number of tokens (N = 500)

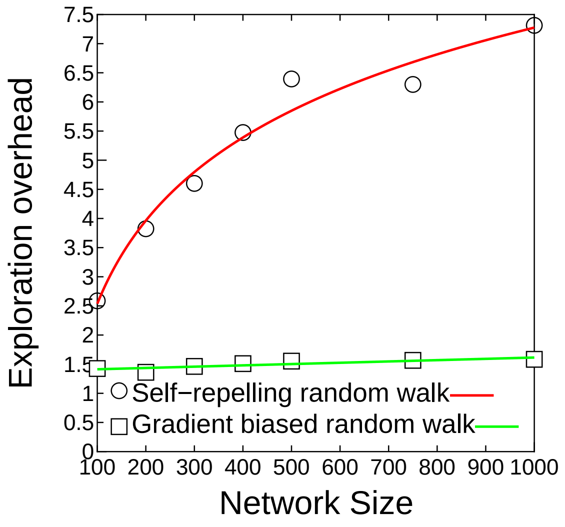


(b) Messages vs number of tokens (N = 500)

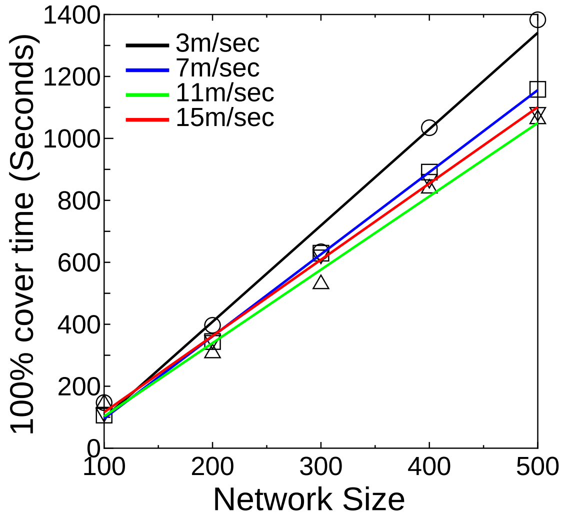
Figure 8.8: Impact of node speed

that the improvement offered by gradient biased self-repelling random walks with gradient biasing over that of self-repelling random walks is far grater in this case. This is because, the use of gradients allows a token to be quickly transferred to unvisited nodes even when the network becomes connected momentarily. Self-repelling random walks spend more time in exploring for unvisited nodes and may not be able to utilize temporary moments when the network is connected.

In Fig. 8.9b, we show the impact of higher network speed on self-repelling random walks in such sparse networks. We observe that the cover time improves with network speed in this case. This is probably explained by the fact that at higher speeds, nodes which are temporarily disconnected from the portion that has a token, tend to converge with the connected component sooner and thus reduce the long tail in cover time.

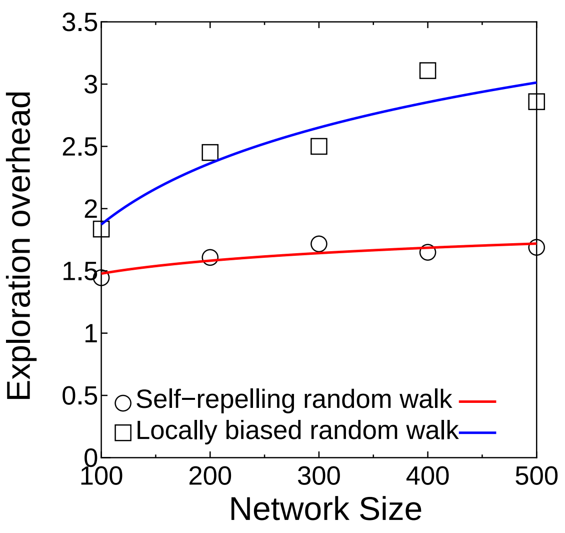


(a) Exploration overhead

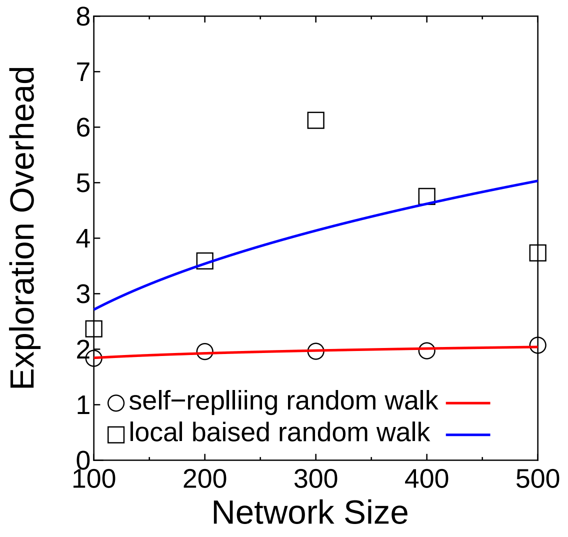


(b) 100% Cover time as a function of Node size

Figure 8.9: Impact of density(Sparse network)



(a) Average



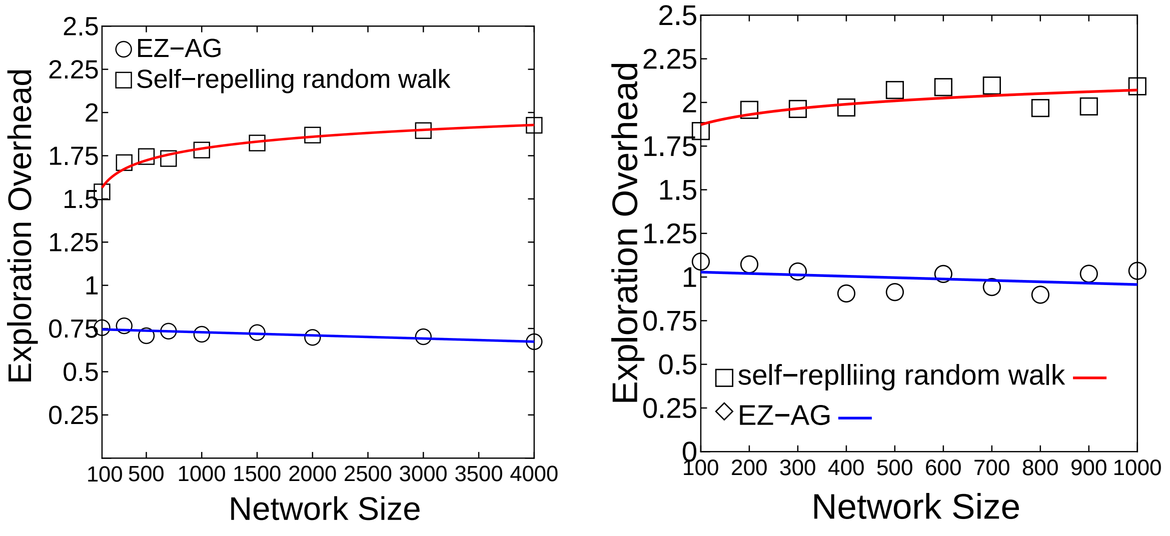
(b) 95%percentile

Figure 8.10: Exploration overhead as a function of network size for locally biased random walks and self-repelling random walks

8.3 Binary self-repelling random walks vs self-repelling random walks

In this section, we compare self-repelling random walks with a special case of the same in which nodes only keep track of whether they are visited and do not keep track of the number of times they have been visited. When all neighboring nodes have been visited, the token is transferred to a visited node chosen uniformly at random among the visited nodes. Thus, until there is at least one neighbor that has not been visited, the action at each node is identical to that of self-repelling random walks. When all neighbors have been visited, the locally biased random walks are reduced to pure random walks. We observe from Fig. 8.10 that the performance of such locally biased random walks is quite similar to the self-repelling random walks and the latter offers a constant factor of improvement. A detailed analysis of such locally biased random walks can be found in [9].

8.4 Self-repelling random walks vs EZ-AG



(a) Average (b) 95%percentile

Figure 8.11: Exploration overhead at 100% coverage as a function of network size for self-repelling random walks and EZ-AG

Here, we compare the convergence characteristics of pure random walks with that of self-repelling random walks and push-assisted self-repelling random walks. To compare these characteristics we used exploration overhead.

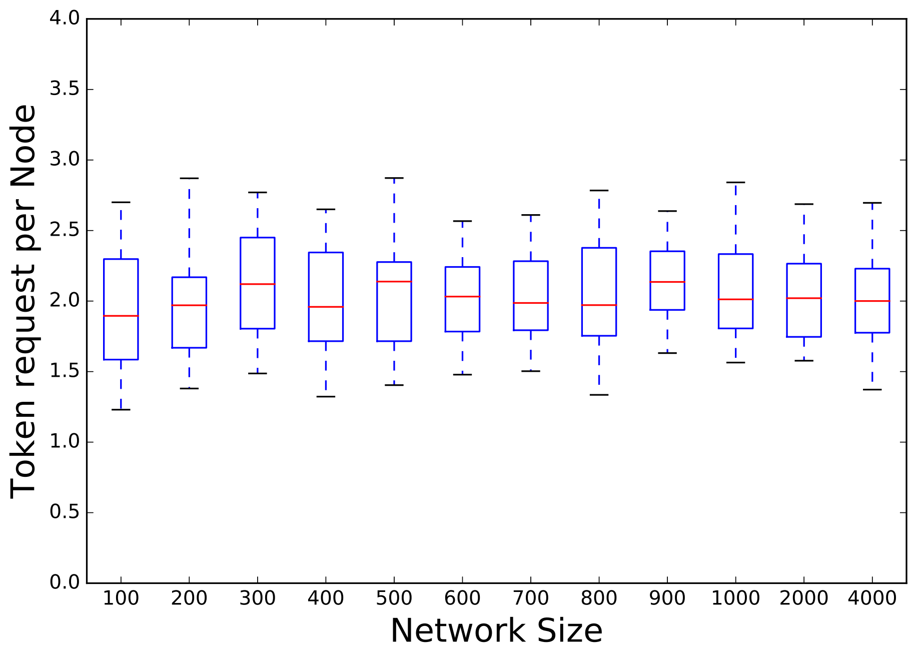
8.4.1 Convergence characteristics

In Fig. 8.3a, we show the exploration overhead of self-repelling random walk during different stages of coverage. As seen in Fig. 8.3a, until about 85% coverage, self-repelling random walks have an exploration overhead of around 1 (irrespective of network size) but then the overhead starts to rise sharply. This is because, until this point self-repelling random walks enable a token to find an unvisited node directly and there are very few wasted explorations. A slowdown for self-repelling random walk is noticed after this point. As a result, the exploration overhead at 100% coverage is close to 2 and moreover it increases with network size. This is what we aim to address using EZ-AG.

The exploration overhead at 100% coverage is shown in Fig. 8.11a and Fig. 8.11b for self-repelling random walks and EZ-AG (i.e., push-assisted self-repelling random walks). As seen in the figure, the exploration overhead for self-repelling random walks grows with a logarithmic trend due to the wasted explorations towards the tail end of the random walk phase when most of the nodes have already been visited. The push assisted self-repelling random walks remove these wasted explorations and as a result the median exploration overhead stays constant at all network sizes and is actually less than 1 (approximately 0.75 as seen in Fig. 8.11a).

8.4.2 Message and Time

In Fig. 8.13a and Fig. 8.13b, we show the total aggregation time and the total number of messages as a function of network size for the aggregation protocol based on push-assisted self-repelling random walks. The total number of messages required to complete the data aggregation includes the push messages, the messages involved in the self-repelling random walk phase, and the messages involved in disseminating the result to all the nodes using a flood. Note that, each token transfer step itself consists of announcement, token request and



(a) Number of token requests generated per token transfer

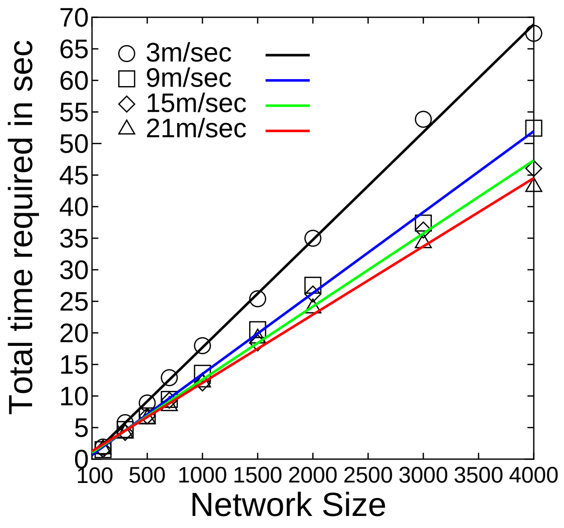
Figure 8.12

token transfer messages. These are all included in Fig. 8.13b ,which shows that the messages grow linearly with network size.

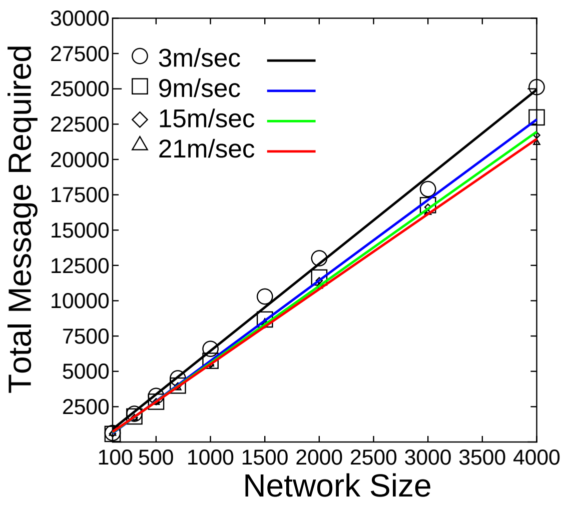
An interesting aspect of the token transfer procedure is the number of requests generated for a token during each iteration. Note that the average number of neighbors increases as $\theta(logN)$ when the network size increases. However, from Fig. 8.12a, the number of token requests per transfer is seen to be independent of the number of neighbors. From the box plot of Fig. 8.12a, we observe that the average number of token requests in each trial is in the range of $1 - 3$. This is because nodes that are visited less often send a request earlier than those that are visited more. And, if a node hears a request from a node that has been visited less often than itself, it suppresses its request. Thus, irrespective of the neighborhood density, the number of token requests per node stays constant.

As seen in Fig. 8.13a, the total aggregation time also exhibits a linear trend. Note that the measurement of time is quite implementation specific and incorporates messaging latency in the wireless network. For instance, in our implementation each transaction (i.e., each iteration of token announcement, token requests and token passing) took on average $25msec$.

But this number could be much smaller using methods such as [39] that use collaborative communication for estimating neighborhood sizes that satisfy given predicates.



(a) EZ-AG total time required



(b) EZ-AG total message required

Figure 8.13: Total time and messages

8.4.3 Impact of mobility model and speed

In Fig. 8.14, we evaluate the impact of node mobility on the exploration overhead of EZ-AG. We observe that even though random waypoint and Gauss Markov models do not preserve the uniform distribution of node locations, the exploration overhead exhibits a similar trend. As seen in Table 1.1, the network structure is rapidly changing at the speeds chosen for evaluation. Despite this, in Fig. 8.15a, we observe that the exploration overhead actually starts decreasing with node speed (this is shown more clearly in Fig. 8.15b for networks with different sizes).

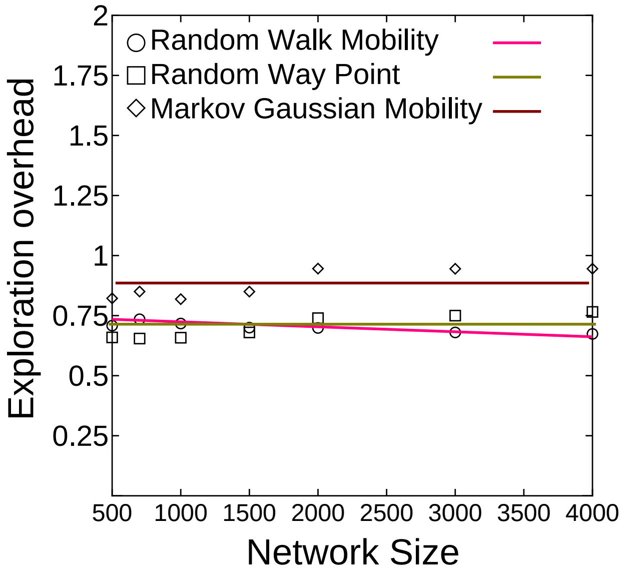
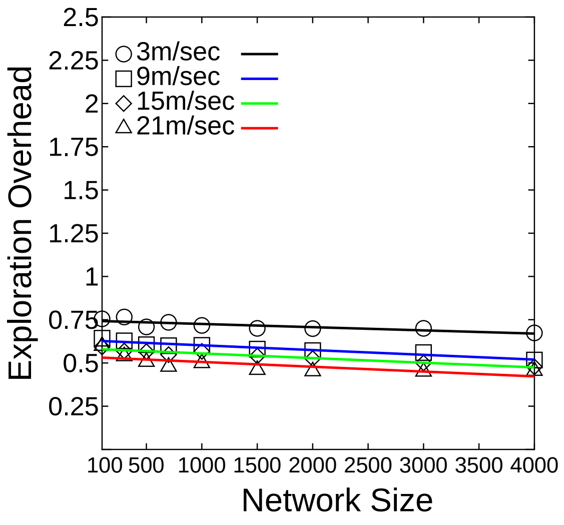
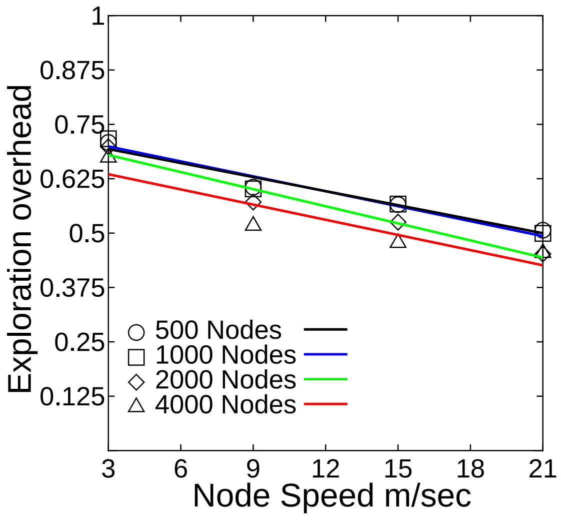


Figure 8.14: Analysis of exploration overhead over different mobility model



(a) Impact of mobility models on exploration overhead of EZ-AG



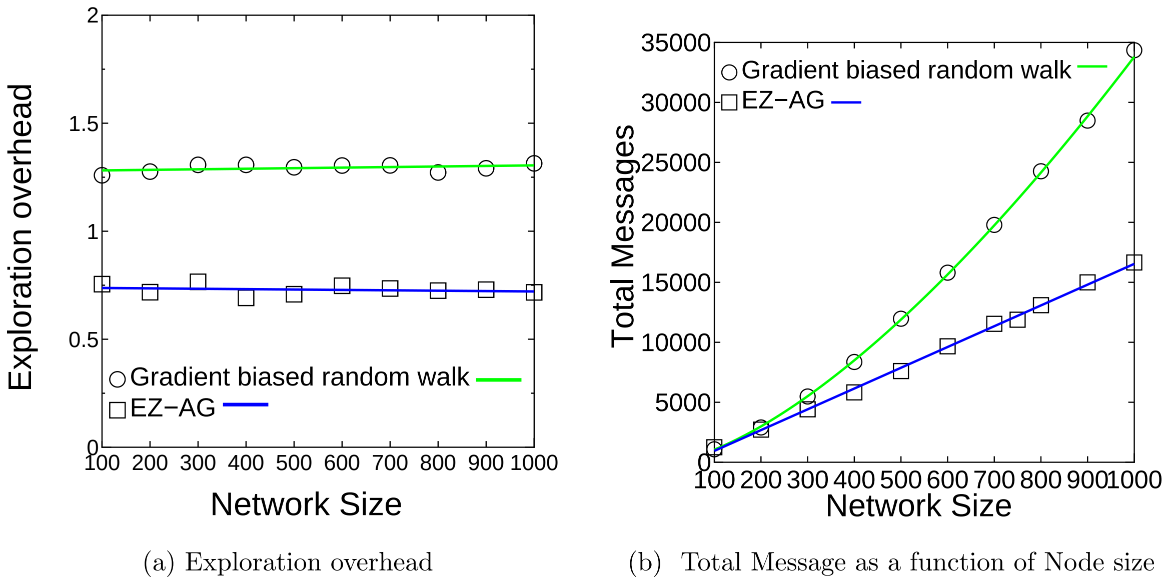
(b) Exploration overhead as a function of node speed (network size = 500)

Figure 8.15: Analysis of exploration overhead of EZ-AG for different speed

8.5 Gradient biased self-repelling random walks vs EZ-AG

In this section, we compare the convergence characteristics of gradient biased self-repelling random walks and EZ-AG. Both protocols are designed for data aggregation and have the same cover time $O(N)$. Note that these two protocols aim for different types of data aggregation: duplicate sensitive and duplicate insensitive, respectively.

There are two reasons to introduce EZ-AG. Firstly, EZ-AG improves required message bound from $O(Nlog(N))$ to $O(N)$. Secondly, EZ-AG is faster than gradient biased self-repelling random walks by some constant factor. In remaining section we will provide a comparison between EZ-AG and gradient biased self-repelling random walks to prove this point.



(a) Exploration overhead (b) Total Message as a function of Node size

Figure 8.16: Comparison between EZ-AG and Gradient biased

In Fig. 8.16a, we show the exploration overhead of gradient biased self-repelling random walks and EZ-AG. The exploration overhead is observed to be constant over all node sizes for both cases. However, EZ-AG shows some constant factor improvement when it is compared to gradient biased self-repelling random walks. (Approximately 0.5 in this case). This shows that if an aggregate is duplicate insensitive, EZ-AG could calculate the aggregate faster than gradient biased self-repelling random walks.

In Fig. 8.16b, we show the total message required to compute aggregate. The total messages for EZ-AG contains two floods and a self-repelling random walk. As seen in Fig. 8.16b, EZ-AG outperforms the gradient based random walks. Also EZ-AG has linear growth which matches with our analysis in chapter 6. On the other hand, the total message for gradient biased self-repelling random walks show nonlinear trends.

As we can see from Fig. 8.16, EZ-AG outperforms the gradient biased self-repelling random walks in terms of both aggregation time and the total messages. Thus, we conclude that it makes sense to apply EZ-AG when the aggregation is duplicate insensitive type.

Chapter 9

Comparison against tree based approach

In this section, we compare the performance of EZ-AG with a structured approach for one-shot duplicate insensitive data aggregation that involves maintaining network structures such as spanning trees. For our comparison, we use a prototype tree-based protocol (aggregate tree protocol) that we describe briefly. Unlike existing data aggregation protocols for static networks [3, 2], aggregation tree protocol periodically refreshes the underlying tree structure to cope with mobility of the node. Other than that, the idea of aggregation tree protocol is very similar to existing data aggregation protocol for static sensor networks. Also, we compare EZ-AG with existing gossip protocol.

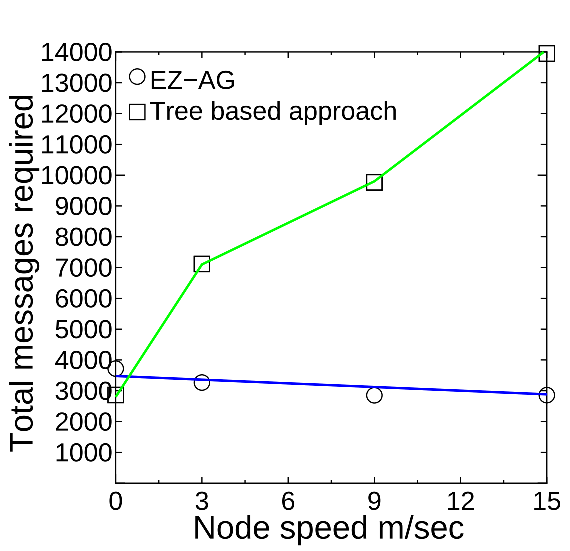
9.1 Aggregate Tree Protocol

First, we discuss the data structure maintained at each node under aggregate tree protocol. Each node creates a fixed size transmission buffer and receiving buffer. The structure of this buffer is implementation dependent. For this comparison, the buffer is implemented as a ring buffer (i,e,. when the buffer gets full, a new item replaces the oldest item). The transmission buffer is used to store the data originated from itself or its children. On the other hand, receiving buffer stores the data originating from its children.

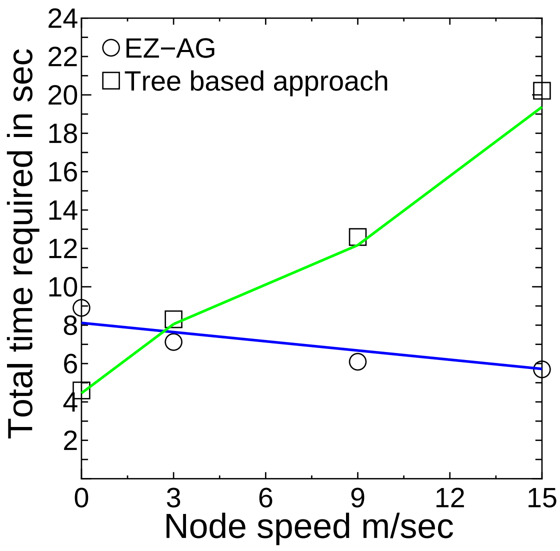
The node wanting to find an aggregate over the network initiates tree creation and tries to maintain it. A node initiates data aggregation by flooding a request message in the network. This request message contains a sequence number and ETX value which specifies the depth of the tree. Thus, each time a request message is flooded, this ETX value is incremented by 1. Nodes receiving request message will first check if this is the first request message they have received. If this is the first message, nodes receiving the message will set its parent as the sender of request message and set its ETX value. After this initial update, there are two things each node does. 1) Schedule a data transmission to its parents. The duration of the timer is randomly picked from an interval of 0 msec to 50 msec. 2) Broadcast a request message with updated ETX value.

A node transmits all the data in the transmission buffer when the scheduled timer expires. The parent receiving this message first searches its receiving buffer for duplication. If an identical message is found, the parent node drops this message. If there is no duplication, it will add the message to the receiving buffer. If there is some message already stored in the transmit buffer, a node opportunistically aggregates message into one packet to reduce the message size.

A message can be lost due to the node mobility or wireless link condition. To handle this, a node keeps transmitting messages until either it receives an acknowledgment or reaches its maximum retry count. However, this approach is not sufficient to handle messages loss due to node mobility. If the loss of acknowledgment is caused by node mobility, a node needs to relinquish its parent and find a new parent. The initiator node will periodically send a request message with the same sequence number until it receives all data (i.e., initiator needs to know the total node count in advance). The key point is this request message has the same sequence number as the original message so that the node can distinguish which request is for the message. The node exhausting maximum retry count will clear its state and wait for a periodic request message to find its new parent node. This way, a node that lost its parent can obtain a new parent and continue to send/forward data.



(a) Total messages as a function of node speed



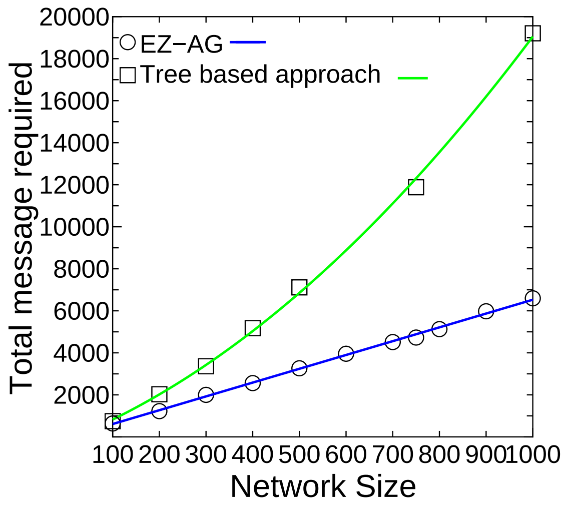
(b) Aggregation time as a function of node speed

Figure 9.1: Comparison of time and messages for EZ-AG and tree-based protocol at different node speeds

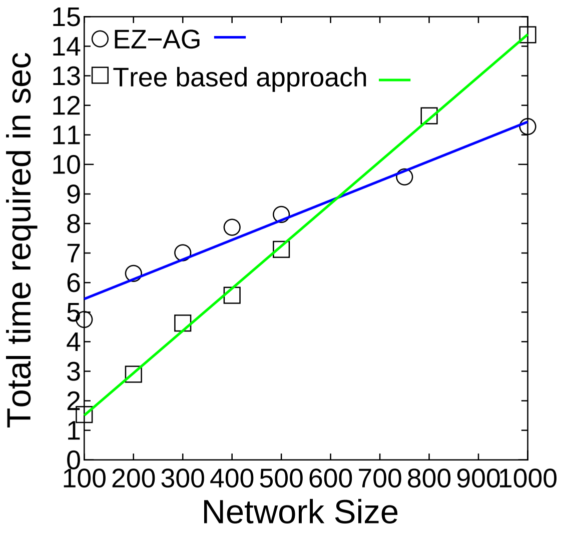
9.2 Total time and message with different node speed

In this subsection, we compare total message and time required by aggregation tree protocol and EZ-AG over different node speeds. For this simulation the node size is fixed to 500 nodes. In Fig. 9.1a, we observe that aggregate tree protocol performs slightly better compared to EZ-AG on static network (node speed of 0 m/sec). This shows that for a static network, having efficient routing information benefits data aggregation. However, as the node speed increases, the total message for EZ-AG starts to decrease while total messages for aggregate tree protocol increases. At node speed of 21 m/sec, aggregate tree protocol requires more than four times as many messages. This shows as the link change gets frequent, EZ-AG quickly catches up with aggregate tree protocol and outperforms it. This is because as the node speed increases, the link change increases and aggregate tree protocol has more wasted messages (message transfer to invalid parent) and message to refresh tree structure. As seen in Fig. 9.1a, aggregate tree protocol quickly loses its advantage of having an efficient route for forwarding data once node starts to move.

In Fig. 9.1b, we plotted total time under the same conditions. As seen in Fig. 9.1b, total time shows very similar trend as message. As the node speed increases, the total time required decreases for EZ-AG and increases for aggregate tree protocol. Again this is due to the rate of link change. As the link change gets more frequent, aggregate tree protocol needs to refresh its underlying tree. The message needs to stay in the transmit buffer till node finds a new parent node.



(a) Total messages as a function of node speed

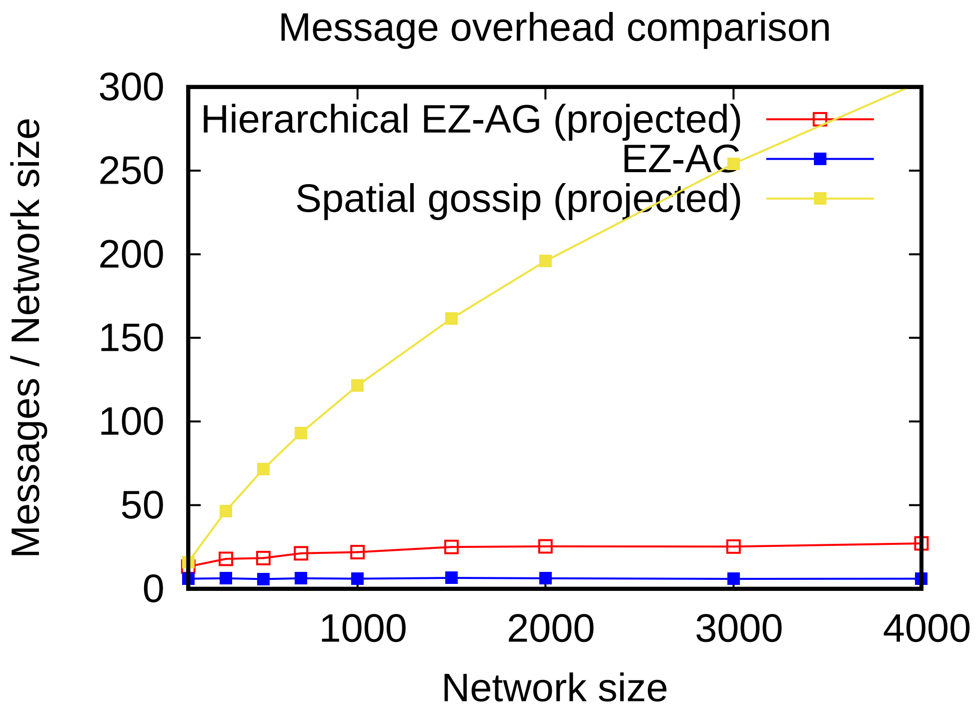


(b) Aggregation time as a function of node speed

Figure 9.2: Comparison of time and messages for EZ-AG and tree-based protocol at different network size

In Fig. 9.2, we fixed average node speed (3 m/sec) and plotted total message and time over different node size. In both cases, we observe that EZ-AG has a linear trend. The total messages grows linearly as network size increases. On the other hand, aggregate tree protocol shows hyper linear trend. Please note that as network size increases the total message for EZ-AG gets smaller than that of aggregate tree protocol. Again, this is due to the rate of link change. As node size increase, the chance of losing a new neighbor increases. This causes the large number of re-transmission for aggregate tree protocol.

Our observations show that as the rate of link change increases (either due to node size or average node speed), aggregate tree protocol starts to spend more time and message to refresh its underlying tree and loses the advantage of having efficient routing.



(a) Total messages as a function of node speed

Figure 9.3: Projected number of messages per node for hierarchical EZ-AG and multi-resolution spatial gossip

9.3 Comparison with gossip techniques

In [11] and [12], a spatial gossip technique is described where each node chooses another node in the network (not just neighbors) at random and gossips its state. When this is repeated $O(log^{1+\epsilon}(N))$ times (where $\epsilon > 1$), all nodes in the network learn about the aggregate state. Note that this scheme requires $O(N.polylog(N))$ messages. EZ-AG requires only $O(N)$ messages.

In [11], an extension to the spatial gossip technique is described which provides a multi-resolution synopsis of the network state at each node. The technique described in [11] requires $O(Nlog^{5.4}(N))$ messages. The hierarchical extension of EZ-AG only requires $O(NlogN)$ messages. The difference is actually quite significant at larger network sizes as seen in Fig. 9.3a, where we show the analytically projected difference in messages transmitted per node for hierarchical EZ-AG and multi-resolution spatial gossip [11].

Chapter 10

Discussion

In this chapter, we discuss some issues and optimization techniques that are not related to the core idea of using random walks for data aggregation, but nevertheless are important in the context of implementing our protocols in a MANET.

10.1 Reliable token transfer

Reliable token transfer is critical for successful operation. If a token is released by a node, but the intended recipient did not receive the token reply message, the token is lost. At the same time, if the sending node relies on acknowledgments to release a token, it is possible that the acknowledgments are lost and duplicate tokens are created. For applications where duplicate counting is not permitted (such as duplicate-sensitive aggregation), this is a problem. This issue can be addressed in practice by using acknowledgments in conjunction with checkpoints. The procedure is described below.

As soon as a token reply has been sent, the sender releases the token (the node resets holder to zero). At the same time, it remains in a waiting state for acknowledgments from the recipient. If an acknowledgment is not received within a time T_a, the token send message is repeated up to a maximum of K re-tries. If the recipient receives the token multiple times, it simply repeats the acknowledgment message. However, if the token sender does not receive the acknowledgment even after K retries, it creates a *checkpoint* for the token: (a) the aggregate computed thus far is appended to the token along with the token id, (b) a fresh token id is created (unique ids can be created by simply assigning a node's id to the token during creation) and (c) the token aggregate is reset. It is possible that the token was actually successfully passed, but even in this case the checkpoint will not create duplicate counting. At the same time, the process ensures that data is not lost.

10.2 Termination detection

Gradient biased random walk

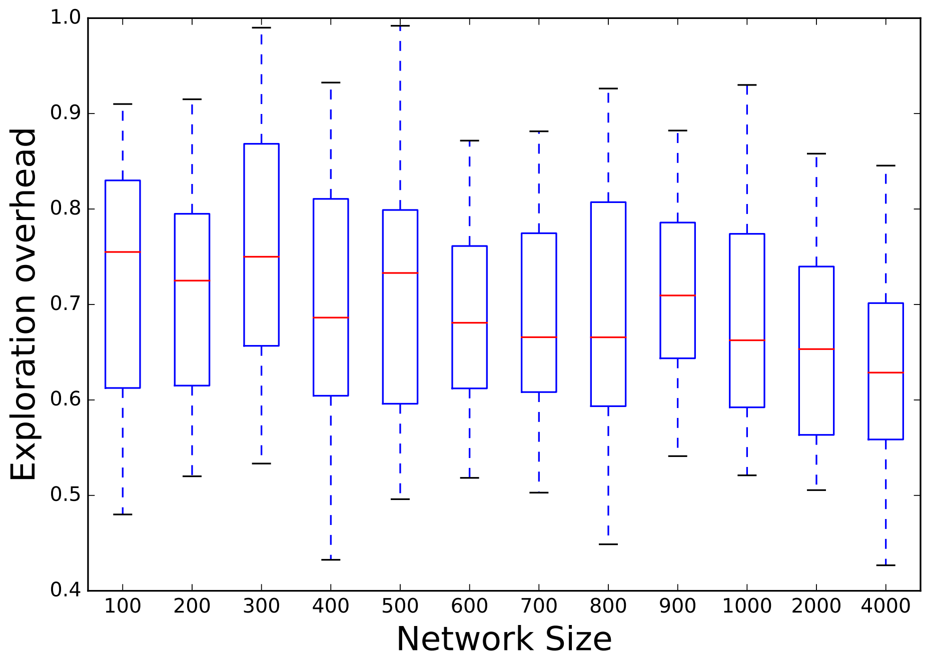
When using gradient biased random walk, termination can be deterministically detected. Note that when all nodes have been visited, the gradient setup will be terminated because the gradient setup is only initiated by nodes that have not been visited. As a result, a node that holds a token will continue to get only a level 0 reply for its token announcement. If a gradient is being setup, there would be at least one neighboring node with a value of level > 0. Therefore, when nodes holding the token get a level 0 reply from all its neighbors over an interval greater than the gradient refresh interval, the holder nodes can conclude that all nodes in the network have been visited.

Self-repelling random walks

In contrast, when using self-repelling random walks, there is no deterministic way to detect termination. However, as the percentage of visited nodes increases, the ratio of token transfers to the visited nodes starts increasing. This ratio may be used to design an approximate threshold for termination detection. Moreover, the result in Fig. 8.3a shows the expected ratio of token transfers to the visited nodes at different levels of coverage with self-repelling random walk. In this figure, we see that until about $80 - 90\%$ coverage, there is very little variance in the token passing overhead ratio across different network sizes. Therefore, these values can be used to determine approximate thresholds for terminating a self-repelling random walk trial at a desired level of coverage, irrespective of network size.

EZ-AG

In Fig. 10.1a, we show the variation in exploration overhead for EZ-AG over 50 different trials at different network sizes. We observe that irrespective of network size, for 97.5% of the trials, the exploration overhead is smaller than 1. We can use this data to design a terminating condition for the random walk phase of the protocol. For example, we could terminate the random walk phase after exactly N steps, and then start the dissemination of the aggregate.



(a) Variance in exploration overhead

Figure 10.1

10.3 Token ex-filtration

Once the initiated tokens have visited all nodes, it is necessary to ex-filtrate the tokens to a given location such as the operating base station or to one or more querying nodes in the network. Instead of using structures to route these aggregates towards querying nodes, a simple solution is to simply flood the aggregate tokens across the network in $O(D)$ time (where D is the network diameter) with an $O(Nk)$ message overhead where k is the number of tokens. This leads to a potential question: why not use flooding or diffusion based approaches all the way? Note that the cost of disseminating data from each node to all other nodes is $O(N^2)$ where N is the number of nodes in the network. By using a fixed number of k tokens to first compute the aggregates and then flooding the aggregates, the message overhead for flooding is only $O(Nk)$. Thus, our bounds on message overhead remain unaffected.

Note that other structure-free solutions are also possible for token ex-filtration. For instance, another potential solution is to transmit the k aggregated tokens using a long distance transmission link (such as cellular or satellite links) in hybrid MANETs where the *long links* are used for infrequent, high priority data.

Chapter 11

Conclusion

11.1 Conclusion

In this thesis, we presented robust, scalable, structure-free protocols for duplicate-sensitive and duplicate-insensitive data aggregation in MANETs. First, we observed that pure random walks have a very high exploration overhead and this causes the cover time to be very high. Then, we introduced self-repelling random walks designed to improve this high exploration overhead. The approach taken by self-repelling random walks is to bias neighbor nodes using the number of time each neighbor has been visited so far. This simple approach guarantees that the token is always passed to the unvisited neighbor if one is available. This very simply approach shows the significant reduction in the exploration overhead and improves the cover time. With this simple approach, the complexity of the cover time is reduced to an order of $O(Nlog(N))$. Also, we empirically show the uniformity of self-repelling random walks on different mobility models.

However, self-repelling random walks show a long tail behavior. Once a self-repelling random walk covers a certain percentage of the nodes, the exploration overhead starts to increase. This cannot be avoided because, as the coverage increases, the chance of having unvisited neighbor(s) decreases. Eventually, all neighbors of token holder become visited nodes. In this case, a token keeps visiting already visited node till it hits the node which has unvisited neighbor. All token transfers performed till the token reaches the node with unvisited neighbor(s) increase the exploration overhead. To mitigate this long tail behavior, we proposed gradient biased random walks to pull tokens towards unvisited nodes. Gradient biased random walks have a cover time of $O(N)$, avoid a long tail and significantly reduce the cover time as well as exploration overhead.

We quantified the performance of gradient biased random walks under different network conditions. Our analysis shows that gradient biased random walks always outperform pure random walks and self-repelling random walks under all of these conditions. We showed that gradient biased random walks have very little state overhead and is naturally resilient to topology changes of MANETs.

Then, we introduced the duplicate-sensitive data aggregation protocol in MANETs that exploits the simplicity and efficiency of self-repelling random walks: EZ-AG.We showed that by complementing self-repelling random walks with a single step push phase, EZ-AG can achieve duplicate-insensitive data aggregation in $O(N)$ time and messages. In terms of message overhead, EZ-AG outperforms existing structure free gossip protocols by a factor of $log(N)$. We quantified the performance of EZ-AG using ns-3 simulations under different mobility modes. We also showed EZ-AG outperforms structure based protocols in mobile networks and the improvement gets increasingly significant as average node speed increases.

We also described a hierarchical extension to EZ-AG that provides multi-resolution aggregates of the network state to each node. It outperforms existing technique by a factor of $O(log^{4.4}N)$ in terms of number of message.

Lastly, we want to point out that our protocols are lightweight in terms of resource requirements and make rather minimal assumptions of the underlying network. In particular, it does not assume knowledge of node addresses or locations, require a neighborhood discovery service or network topology information, or depend upon any particular routing or transport protocols such as TCP/IP. A key implication is that our protocol is suitable for heterogeneous networks (and radio platforms).

References

[1] V. Naik, A. Arora, P. Sinha, and Hongwei Zhang, "Sprinkler: A reliable and energy efficient data dissemination service for extreme scale wireless networks of embedded devices," *Mobile Computing, IEEE Transactions on*, vol. 6, no. 7, pp. 777–789, July 2007.

[2] Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong, "Tag: A tiny aggregation service for ad-hoc sensor networks," *SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 131–146, Dec. 2002.

[3] Omprakash Gnawali, Rodrigo Fonseca, Kyle Jamieson, David Moss, and Philip Levis, "Collection tree protocol," in *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, New York, NY, USA, 2009, SenSys '09, pp. 1–14, ACM.

[4] Chalermek Intanagonwiwat, Ramesh Govindan, Deborah Estrin, John Heidemann, and Fabio Silva, "Directed diffusion for wireless sensor networking," Piscataway, NJ, USA, Feb. 2003, vol. 11, pp. 2–16, IEEE Press.

[5] V. Kulathumani, A. Arora, M. Sridharan, K. Parker, and B. Lemon, "On the repair time scaling wall for manets," *IEEE Communications Letters*, vol. 20, no. 8, pp. 1623–1626, Aug 2016.

[6] Chen Avin and Bhaskar Krishnamachari, "The power of choice in random walks: An empirical study," in *Proceedings of the 9th ACM International Symposium on Modeling Analysis and Simulation of Wireless and Mobile Systems*, New York, NY, USA, 2006, MSWiM '06, pp. 219–228, ACM.

[7] Erwin Bolthausen, "On self-repellent one dimensional random walks," *Probability Theory and Related Fields*, vol. 86, no. 4, pp. 423–441, 1990.

[8] MJAM Brummelhuis and HJ Hilhorst, "How a random walk covers a finite lattice," *Physica A: Statical Mechanics and its Applications*, vol. 85, no. 1, pp. 33–44, 1992.

[9] Vinod Kulathumani, Ken Parker, Mukundan Sridharan, and Anish Arora, "Census: A protocol for visiting all nodes in manets using biased random walks," *CoRR*, vol. abs/1409.7368, 2014.

[10] Suman Nath, Phillip B. Gibbons, Srinivasan Seshan, and Zachary R. Anderson, "Synopsis diffusion for robust aggregation in sensor networks," in *Proceedings of the 2Nd*

International Conference on Embedded Networked Sensor Systems, New York, NY, USA, 2004, SenSys '04, pp. 250–262, ACM.

[11] Rik Sarkar, Xianjin Zhu, and Jie Gao, "Hierarchical spatial gossip for multiresolution representations in sensor networks," *ACM Trans. Sen. Netw.*, vol. 8, no. 1, pp. 4:1–4:24, Aug. 2011.

[12] David Kempe, Jon Kleinberg, and Alan Demers, "Spatial gossip and resource location protocols," *J. ACM*, vol. 51, no. 6, pp. 943–967, Nov. 2004.

[13] Jeffrey Considine, Marios Hadjieleftheriou, Feifei Li, John Byers, and George Kollios, "Robust approximate aggregation in sensor data management systems," *ACM Trans. Database Syst.*, vol. 34, no. 1, pp. 6:1–6:35, Apr. 2009.

[14] The ns-3 network simulator., ," Online - last accessed 2-June-2016.

[15] Yufei Tao and Dimitris Papadias, "Historical spatio-temporal aggregation," *ACM Trans. Inf. Syst.*, vol. 23, no. 1, pp. 61–102, Jan. 2005.

[16] Roy Friedman, Daniela Gavidia, Luis Rodrigues, Aline Carneiro Viana, and Spyros Voulgaris, "Gossiping on manets: The beauty and the beast," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 5, pp. 67–74, Oct. 2007.

[17] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Randomized gossip algorithms," *Information Theory, IEEE Transactions on*, vol. 52, no. 6, pp. 2508–2530, June 2006.

[18] M.G. Rabbat, "On spatial gossip algorithms for average consensus," in *Statistical Signal Processing, 2007. SSP '07. IEEE/SP 14th Workshop on*, Aug 2007, pp. 705–709.

[19] L?szl? Lov?sz, "Random walks on graphs: A survey," 1993.

[20] U. Feige, "A tight lower bound on the cover time for random walks on graphs," Tech. Rep., Jerusalem, Israel, Israel, 1993.

[21] Uriel Feige, "A tight upper bound on the cover time for random walks on graphs," *Random Struct. Algorithms*, vol. 6, no. 1, pp. 51–54, Jan. 1995.

[22] Noga Alon, Chen Avin, Michal Koucky, Gady Kozma, Zvi Lotker, and Mark R. Tuttle, "Many random walks are faster than one," in *Proceedings of the Twentieth Annual Symposium on Parallelism in Algorithms and Architectures*, New York, NY, USA, 2008, SPAA '08, pp. 119–128, ACM.

[23] Colin Cooper and Alan Frieze, "The cover time of random regular graphs," 2003.

[24] Colin Cooper, Alan Frieze, and Tomasz Radzik, "Multiple random walks in random regular graphs," *SIAM J. Discret. Math.*, vol. 23, no. 4, pp. 1738–1761, Nov. 2009.

[25] Chen Avin, Michal Kouck, and Zvi Lotker, "How to Explore a Fast-Changing World (Cover Time of a Simple Random Walk on Evolving Graphs)," in *Proceedings of the 35th International Colloquium on Automata, Languages and Programming, Track A: Algorithms, Automata, Complexity, and Games.* 2008, vol. 5125 of *Lecture Notes in Computer Science*, pp. 121–132, Springer International Publishing.

[26] Chen Avin and Carlos Brito, "Efficient and robust query processing in dynamic environments using random walk techniques," in *Proceedings of the 3rd International Symposium on Information Processing in Sensor Networks*, New York, NY, USA, 2004, IPSN '04, pp. 277–286, ACM.

[27] A. Clementi, A. Monti, F. Pasquale, and R. Silvestri, "Information spreading in stationary markovian evolving graphs," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 9, pp. 1425–1432, Sept 2011.

[28] J.-Y. Le Boudec and M. Vojnovic, "Perfect simulation and stationarity of a class of mobility models," in *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, March 2005, vol. 4, pp. 2743–2754 vol. 4.

[29] P. Nain, D. Towsley, Benyuan Liu, and Zhen Liu, "Properties of random direction models," in *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, March 2005, vol. 3, pp. 1897–1907 vol. 3.

[30] Tracy Camp, Jeff Boleng, and Vanessa Davies, "A survey of mobility models for ad hoc network research," *WIRELESS COMMUNICATIONS and MOBILE COMPUTING (WCMC): SPECIAL ISSUE ON MOBILE AD HOC NETWORKING: RESEARCH, TRENDS AND APPLICATIONS*, vol. 2, pp. 483–502, 2002.

[31] A. Clementi, A. Monti, F. Pasquale, and R. Silvestri, "Information spreading in stationary markovian evolving graphs," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 9, pp. 1425–1432, Sept 2011.

[32] C Byrnes and A J Guttmann, "On self-repelling walks," *Journal of Physics A: Mathematical and General*, vol. 17, no. 17, pp. 3335, 1984.

[33] A.Arora. V.Kulathumani, M.Nakagawa, "Converge characteristics of self-repelling random walks in mobile ad-hoc netowrks. https://www.csee.wvu.edu/ vkkulathumani/srrw.pdf," .

[34] Aristides V. Doumas and Vassilis G. Papanicolaou, "The coupon collector's problem revisited: Asymptotics of the variance," *Advances in Applied Probability*, vol. 44, pp. 166–195, 3 2012.

[35] Donald J. Newman and Lawrence Shepp, "The Double Dixie Cup Problem," *The American Mathematical Monthly*, vol. 67, no. 1, 1960.

[36] Philippe Flajolet and G. Nigel Martin, "Probabilistic counting algorithms for data base applications," *J. Comput. Syst. Sci.*, vol. 31, no. 2, pp. 182–209, Sept. 1985.

[37] Noga Alon, Yossi Matias, and Mario Szegedy, "The space complexity of approximating the frequency moments," in *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing*, New York, NY, USA, 1996, STOC '96, pp. 20–29, ACM.

[38] J. Considine, F. Li, G. Kollios, and J. Byers, "Approximate aggregation techniques for sensor databases," in *Data Engineering, 2004. Proceedings. 20th International Conference on*, March 2004, pp. 449–460.

[39] Wenjie Zeng, Anish Arora, and Kannan Srinivasan, "Low power counting via collaborative wireless communications," in *Proceedings of the 12th International Conference on Information Processing in Sensor Networks*, New York, NY, USA, 2013, IPSN '13, pp. 43–54, ACM.