Comparison of Collection Tree Protocols with Gossip Algorithms for Data Aggregation in Wireless Sensor Networks

Bassel Zuhair Hakoura

Department of Electrical & Computer Engineering
McGill University
Montreal, Canada

September 2011

A thesis submitted to McGill University in partial fulfillment of the requirements for the degree of Master of Engineering.

© 2011 Bassel Zuhair Hakoura

Abstract

Wireless sensor networks are used in many different applications due to their favorable properties, such as low cost, low power consumption and ease in deployment. One such area where they have gained popularity is data collection. Here, there are many readings being collected by sensor nodes that need to be aggregated for further processing. Several algorithms exist for decentralized monitoring of aggregates, but the two that stand out amongst the rest are tree-based protocols and gossip-based protocols. In this thesis, we compare the performance of the Collection Tree Protocol (CTP) to two different gossip algorithms: pairwise randomized gossip and broadcast gossip. Performance is measured in terms of the total number of messages transmitted because this corresponds to the amount of power consumed for aggregation. CTP is a routing protocol that is used in real WSN deployments and is implemented in the TinyOS operating system. One of the main questions of interest in this thesis is what is the cost of setting up and maintaining a spanning tree using CTP in environments where links are lossy or they fail. In order for CTP to form the spanning tree or for randomized gossip to exchange messages, nodes need to know their neighbors. Therefore we look into neighborhood discovery techniques to account for the number of messages transmitted for this purpose. Furthermore, we employ a lossy wireless link model. Simulations show that CTP for data aggregation outperforms pairwise randomized gossip in all simulation settings for different network sizes and different link failure probabilities. Broadcast gossip and CTP perform quite similarly in small networks under certain node initializations, but broadcast gossip is much better than CTP and transmits a smaller number of messages in large networks.

Sommaire

Les réseaux de capteurs sans fil sont utilisés dans de nombreuses applications en raison de leurs abondants atouts, tels que leur faible coût, leur faible consommation d'énergie ainsi que la facilité de leur déploiement. Le domaine des applications de collecte de données est probablement celui oú les réseaux de capteur sans fil ont gagné le plus en popularité lors de la dernière décennie. Dans le domaine des applications de collecte de données, il sagit de recueillir de nombreuses lectures par les capteurs puis ultèrieurement les regrouper et les analyser. Plusieurs algorithmes existent pour le suivi décentralisé des données, mais les deux qui se démarquent du reste sont des protocoles fondés sur les arborescences et les protocoles décentralisés de passage de messages. Dans cette thèse, nous comparons les performances du protocole basé sur les arborescences (CTP) à deux protocoles décentralisés de passage de messages différents: le premier concerne le passage décentralisé des informations par des communications entre des paires de senseurs, le deuxième est celui de passage décentralisé de linformation par diffusion au niveau du réseau. La performance est mesurée en termes du nombre total de messages transmis, car cela correspond á la quantité d'énergie consommée pour regrouper linformation des senseurs. CTP (de langlais Collection Tree Protocol) ou protocole basé sur des arborescences est un protocole de routage qui est utilisé dans les déploiements des réseaux de senseur sans fil et est implémentée dans le système d'exploitation TinyOS. Une des principales questions d'intrêt dans cette thèse est quel est le coût de la mise en place et le maintien d'un arbre couvrant le réseau dans des environnements où des liens sont à perte. Afin de former l'arbre de recouvrement ou pour effectuer le passage décentralisé des informations par paires de senseurs, les noeuds ont besoin de connaître leurs voisins. Par conséquent, nous nous penchons sur les techniques de découverte de voisinage afin de pouvoir savoir le nombre des messages transmis à cet

effet. En outre, nous employons un modèle de réseau sans fil avec perte. Les simulations montrent que pour la collecte et les regroupements des données, la technique basée sur une arborescence surpasse celle de passage décentralisée des informations par paires de senseurs dans tous les scénarios de simulation, pour différentes tailles de réseaux et différentes probabilités de défaillance des liens. La comparaison entre la technique de passage décentralisée de linformation par diffusion et la technique basée sur une arborescence donne des résultats assez similaires dans les réseaux de petite taille et pour certains types dinitialisation. Par contre, lavantage des algorithmes de passage décentralisés de linformation par diffusion est le petit nombre de messages transmis lorsque la taille des réseaux devient plus importante.

Acknowledgments

" God gave you a gift of 86,400 seconds today. Have you used one to say "thank you?" "
William Arthur Ward

I thank the Lord for all His blessings bestowed upon me, some of which are the very people whose names are written here. A page or two is a very limited amount of space to thank all the people who helped me complete this thesis, so for each and every person mentioned here, please know that you deserve much more than the few words or sentences thanking you.

First and foremost, I would like to thank my supervisor, Professor Michael G. Rabbat, for all his support throughout the whole program. Without his guidance, discussions, suggestions, and most importantly, patience, this thesis would not be what it is today. I can't express my gratitude to all the time he dedicated, despite his busy schedule, to make sure this thesis achieves the goals and results we set when we first discussed this topic. One could not ask for a friendlier supervisor.

I would also like to thank both my parents, Falak Almadbak and Zuhair Hakoura, for all their support and faith in my abilities. They always encouraged me to ask questions, think out of the box, and aim for perfect results. It all started when they taught me simple addition, and the rest is history. Their love was, and will always be, the driving force in my life. It is to them that I dedicate this thesis.

I would also like to thank all my fellow Masters and PhD students in the TSP (Telecommunications and Signal Processing) lab in McGill university. Your discussions made my days brighter, and forgive me for wasting so much of your time in useless conversations. Special thanks to Ali, Deniz, Eric, Selim, Andrea, Konstantinos, Milad, Oscar, Niloufar, Rizwan, Haani, Xi, Tao, Rodrigo and Santosh.

I am blessed with having many friends who made this journey bearable. Nadine, I am lucky to have you as one of my best friends, and without you in my life, I would have died of starvation writing this thesis! You were always there for me when the others were busy. Thank you. Yara, you pushed me to work harder, and when things weren't going the way I wanted, you were there to listen to my complaints. If it wasn't for your encouragement, I would still be writing the introduction section. Your random arguments always made me think. Thank you. Stephanie, talking to you kept me sane, and your smile always brightened my day and helped me forget my simulations. Thank you. Nasser, the noise from your apartment at all hours of the day meant I could never sleep too long, and had to escape and work on my research. This has definitely saved me several months of work! Thank you for all the random and funny moments. Bilal and Ramy, we started our undergraduate degrees together, and many years after I know you guys are still here for me. Thank you for all the good times over all those years. Finally, thanks to Ahmed, Yousef and Borhan for pretending you have a lot of work just to come and study with me.

To my parents

Contents

List of Figures

List of Tables

List of Acronyms

CTP	Collection Tree Protocol
WSN	Wireless Sensor Network
TAG	Tiny Aggregation
P2P	Peer To Peer
RGG	Random Geometric Graph
GGE	Greedy Gossip with Eavesdropping
RG	Randomized Gossip
BG	Broadcast Gossip
DTMC	Discrete Time Markov Chain
GB	Gaussian Bumps
Sk	Spike
Sl	Slope
BFS	Breadth First Search
DRR	Distributed Random Ranking

Chapter 1

Introduction

1.1 Motivation

A wireless sensor network (WSN) is a collection of low-cost, autonomous sensor nodes, each of which possesses simple sensing, computational and communication abilities [1]. They are an example of energy-constrained devices. Nevertheless, they are gaining popularity due to their ease in deployment, low power consumption and low cost. They tend to combine characteristics from both networked systems and embedded control systems by having a distributed nature yet stringent energy constraints [2]. A typical use for a WSN involves data collection [3]. Popular examples of a data collection application would be measuring a physical environmental quantity, such as temperature, humidity, light intensity, amongst others. Those applications usually involve a large number of sensor nodes being deployed throughout a region, which introduces many limitations. For example, there is a lack of a central entity providing synchronization and facilitating communication, because the scale of the network would entail large energy consumption to transmit data to a fusion center, and may introduce unacceptable delay [4].

The popularity of those data collection applications demands developing distributed, fault-tolerant algorithms that can operate under those constraints. Those algorithms should carry out required computations as quickly and as efficiently as possible, with minimal overhead. In those applications there can be a large number of nodes in the network such that computations are distributed and the global aggregate value may vary over time. This can be due to node values changing over time, links being added or removed, or nodes leaving or joining the network over time [5].

In the last few years, many algorithms and protocols have been developed and proposed to be used for decentralized monitoring of aggregates. In particular, tree-based protocols and gossip-based protocols are two classes of protocols that are widely used. Tree-based algorithms create and maintain a spanning tree, which is used to collect local data variables from each node, and then aggregate the data towards the root, or sink, node [6]. Thus the only node that has the global aggregate from the whole network is the root node. The communication costs are less than that of a centralized approach [4]. On the other hand, gossip-based protocols do not create or maintain a spanning tree, which is the main reason why they tend to be simpler than their tree-based counterparts. Instead, each node wakes up randomly based on a Poisson process and communicates with one or more of its' neighbors, depending on the class of gossip algorithm [7]. Thus, each node has an estimate of the global aggregate [4]. Gossip-based algorithms usually have the extra advantage that they generate a rough approximation of the aggregate in question in a very short time, which can be an important property for certain applications. Thus it is important to choose the protocol to be implemented in the WSN based on the application at hand.

1.2 Problem Statement

Previous studies rarely focused on a comparison between data aggregation using a tree-based algorithm vs. a gossip-based algorithms. A few recent examples are [4, 8]. Most of the literature of tree-based aggregation algorithms is concerned with the formation of the optimal aggregation tree in terms of energy efficiency [9, 10]. When it comes to gossip algorithm, the main issue was proving the convergence of the algorithms on random geometric graphs and 2-dimensional grids [11], and accelerating the convergence of the algorithm to achieve a certain level of error [7, 12–15].

An important factor affecting the amount of energy consumed by a WSN is the number of messages transmitted, which in turn depends on the underlying protocol. This is because the energy required to transmit messages is much larger than that required for sensing or for carrying out computations. The main question this thesis attempts to answer is how much overhead does a tree-based algorithm, which is the Collection Tree Protocol (CTP) [16], require compared to two gossip-based algorithm. The first one is pairwise, randomized gossip, proposed by Boyd et al. in [7] and the second one is broadcast gossip proposed by Aysal et al. [14]. Pair-wise randomized gossip on a random geometric graph takes $O(n^2 \log(1/\epsilon))$ messages to converge to accuracy ϵ [7], where n is the number of nodes in the network. Broadcast gossip was shown to perform better than randomized gossip since it does not preserve the average value [14]. The average computation on a spanning tree requires $O(n)$ messages when the spanning tree is already established in the network, so one of the main questions of interest in this thesis is what is the cost of setting up and maintaining a spanning tree using CTP in environments where links are lossy or they fail.

The simulations are carried out under different network conditions. Overhead is defined as the total number of messages transmitted by the protocol, from the time the WSN is

deployed up to the computation of the global aggregate and making it available at each node. The network conditions investigated include different number of nodes, and different probabilities of link failure. Ultimately, this will help WSN designers and deployment specialists decide which aggregation method to use in a specific situation to reduce the number of messages transmitted, which reduces the battery usage in each sensor, which in turn prolongs the WSN life.

1.3 Thesis Contribution and Organization

The main contribution of this thesis is a quantitative comparison between CTP for data aggregation and two different gossip algorithms; pairwise randomized gossip and broadcast gossip, in terms of the number of messages transmitted. Chapter 2 provides a comprehensive review of the research done in areas of data aggregation, tree-based protocols, gossip-based protocols, neighbor discovery in a WSN and wireless link quality measurements; all of which are areas directly affecting this thesis. Chapter 3 gives a brief overview of CTP along with its main properties and tasks. Since CTP is used for routing packets in a WSN and not for data aggregation, Chapter 4 describes the modifications we introduced to CTP to implement it. Chapter 5 describes pairwise, randomized gossip algorithms, broadcast gossip algorithms and our implementation of them. Chapter 6 contains simulation results and discussions. Finally, Chapter 7 contains our conclusion and our thoughts on the possible future work to extend on this thesis.

Chapter 2

Literature review

In this chapter, we shall look at the previous work done in several areas which are directly related to this thesis. Section 2.1 gives a general overview of data aggregation. Section 2.2 discusses different properties and examples of tree-based aggregation algorithms, while Section 2.3 discusses properties and examples of gossip-based algorithms. There are several different methods for neighbor discovery, some of which are discussed in Section 2.4. Finally, Section 2.5 provides an overview of different studies that were conducted to try and understand how varying link qualities in a WSN are related to different properties in the network itself, such as the distance between the transmitting node and the receiving node, and the power at the transmitter, amongst others.

2.1 Data Aggregation

A WSN is typically deployed in remote or dangerous areas and is used in a wide range of applications such as environmental monitoring (e.g., tracking of animals [17]), healthcare applications (e.g., MoteTrack [18]), and many others. This implies that battery life

is a crucial factor for the sensor node's life, and in turn to the useful life of the WSN. Data aggregation is a process performed by intermediate nodes in the WSN, by combining several data packets into one, based on their content and the aggregation function in the network [19]. In addition to being one of the applications of a WSN, data aggregation is one method which can be used to decrease the overall number of messages transmitted, thus saving energy and reducing bandwidth usage.

Data aggregation mechanisms might require a certain level of synchronization among the nodes in the network. Usually, this means that each node waits for a certain amount of time to transmit its data, rather than transmit it once it is available. In [19], the authors compared three timing policies in data aggregation networks; periodic simple aggregation, periodic per-hop aggregation, and periodic per-hop adjusted aggregation. In the first policy, each node waits a pre-defined amount of time to aggregate received data then it transmits. The second method is very similar to the first, except that a node will transmit its data once it receives data from all of its children, implying that every node knows all of its children. In the third policy, each node has a different timeout value depending on the node's position in the tree.

In-network aggregation can either lead to size reduction or to no size reduction [19]. Data aggregation with size reduction occurs when the data from different sensors is combined to send a smaller amount of information over the network. An example can be when a node receives several packets about the local temperature measured, one from each sensor, it can just choose the maximum value and transmit it in one packet. Data aggregation without size reduction can occur for example when a node measures two different variables and forwards them without processing them.

In [5], Keshav discusses the importance of computing approximate global state, which is a typical example of data aggregation. He starts by discussing examples of practical

setting where this arises, one of which is sensor networks. He provides the network model, and then describes how the model can be organized based on three factors; the type of function (e.g., extremal values, count queries, etc.), the network topology (e.g., clique, random graph of degree k, etc.) and the state change model (e.g., change in node state, change in links etc.). Five solutions, along with their advantages and disadvantages are considered; centralization, tree-based solutions, flooding (and randomized flooding), random walk-based solutions, and randomized gossip-based solutions. Data aggregation mechanisms are usually linked to the sensors' data gathering techniques and to packets routing in the network [20]. This can affect energy consumption and network efficiency. Akkaya, Demirbas and Aygun [21] described the impact of data aggregation on network-related performance metrics, such as latency, accuracy, etc. The end-to-end latency of packets can increase when data aggregation is used, and this may not be acceptable for certain applications. Accuracy depends on the number of nodes used to collect the data that is received at the base-station, where a larger number of nodes used results in higher accuracy. Fault tolerance of the data aggregation mechanism is important under unreliable wireless links. Akkaya, Demirbas and Aygun also survey several protocols that deal with each of those issues, amongst others, and describe several open problems that sill need to be considered and researched in the future.

2.2 Tree-based Aggregation Algorithms

The idea behind tree-based aggregation algorithms is simple. A spanning tree is first constructed, with the root node being the sink node. Each node transmits its value to its own parent. At each non-leaf node, the value of each of its child nodes, in addition to its own value, is processed before transmitting the result up the tree [6].

CTP is a tree-based protocol [16]. It was introduced as a robust and efficient collection protocol in networks deployed to gather data. It was not designed as a data aggregation protocol. However, since it is used in TinyOS and is one of the state of the art protocols, this thesis will investigate its use in a data aggregation application. Chapter 3 describes CTP, while Chapter 4 describes our introduced modifications to CTP.

Many tree-based aggregation protocols appear in the literature. EADAT (Energy-Aware Distributed Aggregation Tree) was introduced in [22], and as the name suggests, it is based on the energy level of different sensor nodes, and it only depends on local knowledge of the network topology. Ding, Cheng and Xue [22] present heuristics to construct and maintain an aggregation tree. The main idea is that non-leaf nodes will remain active, while the leaf nodes have their radios turned off, to increase the network lifetime. Tiny Aggregation Protocol (TAG) [23] is a generic aggregation service for ad-hoc networks of TinyOS motes. It processes the aggregates *in network*. That is, the computations are done as the data is flowing through the sensors, and irrelevant data is discarded. TAG works with a tree-based routing scheme, with one node appointed as the root node, and constant topology maintenance taking place. It consists of a distribution phase and a collection phase. TAG provides many advantages, including reduced communication compared to a centralized approach, tolerance to disconnections and loss, and symmetric power consumption at all nodes.

In [24], Dam and Stadler present GAP (Generic Aggregation Protocol). It is a distributed, asynchronous protocol, that builds and maintains a BFS (Breadth First Search) spanning tree to perform continuous aggregation in a network. Every node in the network maintains a neighborhood table, which contains the node and its immediate one-hop neighbors. The table describes the role (self, parent, child or peer), depth (hops to the root node) and weight (value of its aggregate weight.) The global aggregate is only available

at the root node. The execution of the protocol is based on asynchronous messages driven by events occurring in the network, such as the node discovery or parent change. The advantage of GAP is that it is self-stabilizing, meaning that the tree is reconstructed and the aggregate is recalculated if nodes join or leave the network, or in the event of node failure. MGAP [25] is an extension of GAP, which is used if the aggregate needs to be available at all the nodes in the network, rather than at the root node only.

Other approaches focus on minimizing energy consumption [9,10]. Oceanus is a heuristic-based algorithm used to build trees based on the aggregation algorithm's efficiency [10]. The energy efficiency of data aggregation is a measure of the data reduction achieved, and the optimal location of the aggregation points depends on the aggregation function. For a simple function, e.g., MAX, the aggregation points are located very close to the data sources, while for a more complex, low-efficiency data aggregation function, the data aggregation points are located much closer to the sink. Tethys [9] is a distributed algorithm used to create the aggregation tree by accounting for the aggregation efficiency and the cost associated with it.

2.3 Gossip-based Algorithms

Gossip algorithms are distributed algorithms that *"mimic the the way information spreads when people gossip about some information with each other"* [26]. They are gaining popularity in today's networking technologies, with applications in WSN's, P2P networks, and wireless ad-hoc networks. These applications include continuous monitoring of aggregates [8], detection of global threshold crossings [27] and averaging [28], amongst others. Their use in a wide range of applications stems from the following properties. Nodes are autonomous, and do not require any overhead from formation or maintenance of complex

routes. They are simple and robust because they are not affected by the network conditions or nodes joining and leaving the network at random, and do not possess a single point of failure [29].

A fully decentralized, robust, gossip-based protocol is presented in [30] that is used to compute aggregates. This scheme is a push-pull gossip protocol where a node randomly selects one other node to gossip with, and both send and receive their values. The update is based on the aggregate function defined in the network. The authors also show that the protocol can be extended to achieve more complex computations (e.g., geometric means, variance etc.), and present (both theoretically and experimentally) how it is robust to node crashes, link failures, and message loss.

In [31], Kempe, Dobra and Gehrke analyze simple gossip-style protocols for computation of aggregates, such as sums and averages of the node values in a network, and show that fast exponential convergence is reached with uniform gossip. They also define and study the data diffusion speed in the network. In addition, they extend their analysis on Push-Sum protocols and the idea of diffusion speed to design protocols for some complex aggregate queries in databases, such as random samples and quantiles of a multiset of elements. The main disadvantage here is that the convergence of the protocols is slow when flooding is used in conjunction with a network having slow mixing random walks.

The use of gossip algorithms to solve the average consensus problem was introduced by Tsitsiklis [32] and is a special kind of aggregation that has been studied extensively in the past [33]. In this problem, each node i in an n-node network has a piece of information, x_i, and the goal is for all the nodes of the network to achieve consensus on a certain quantity, such as the average value.

In [7], Boyd et al. proposed asynchronous, randomized gossip algorithms for modeling the exchange information and computations in an arbitrarily connected network. Those

algorithms divide the computations among different nodes and the gossiping employed in [7] is strictly pairwise gossip; that is a node only communicates with at most one neighbor in any given time slot. Here, each node possesses a clock which ticks at the time of a rate 1 Poisson process. Thus, the inter-tick times at every single node are exponentially distributed. Each node has an initial value, reflecting the measurement it has taken. In the kth iteration, node i is chosen uniformly at random. Node i will need to gossip with one of its neighbors. Node j, one of the neighbors of node i is chosen uniformly at random. Both nodes gossip, exchange their most recent values and then perform an update according to

$$x_i(k) = x_j(k) = \frac{1}{2}[x_i(k-1) + x_j(k-1)] \tag{2.1}$$

It is also shown that the averaging time of the algorithm depends on the second largest eigenvalue of a doubly stochastic matrix that characterizes the algorithm, which means the averaging time is related to the mixing time of a random walk on a suitable graph that describes the algorithm. The averaging time can be defined as the time needed for each node's value to be close to the average [7]. The main disadvantage arising from randomized gossip algorithms is the fact that the convergence rate to a consensus value is slow [14]. Thus, different gossip algorithms were developed to improve this rate.

In [12], a geographic gossip algorithm is introduced. The motivation for geographic gossip algorithms comes from the fact that a significant amount of energy is wasted because certain redundant information is resent in the network. One example of this is when a node tries to communicate with a neighbor whose value is close its own and therefore does not obtain any useful information. There is some inefficiency arising from the slow mixing times of random walks on the communication graph [12]. Geographic gossip algorithms are based on the assumption that sensor nodes in a network usually know their exact locations.

The main idea is that geographic routing can be used to gossip with nodes that are far in the network, rather than exchanging info with a node's immediate, or one-hop, neighbor.

The proposed algorithm by Dimakis et al. [12] assumes that each node, s, knows its own geographic location and the location of its immediate, or one-hop, neighbors within a compact set. For both grids and random geometric graphs, nodes are assumed to know their location in the unit square $[0, 1] \times [0, 1]$. A typical scenario is as follows. Assume that node s, located at $l(s)$, is assigned the k-th clock tick. Node s is activated and uniformly chooses a random point with co-ordinates (x_1, y_1) as the target for its gossip, where (x_1, y_1) is in the unit square. Node s will send a tuple $m_s = (x_s(k), l(s), (x_1, y_1))$ to the neighbor that is closest to the target. This is repeated until it reaches node t, which does not have any one hop neighbors closer to the target location than itself. It is up to node t to make an independent, random decision whether to accept m_s or not. If it accepts, it updates its value according to

$$x_t(k+1) = \frac{1}{2}[x_s(k) + x_t(k)] \tag{2.2}$$

Node t then generates a tuple $m_t = (x_t(j), l(t), l(s))$ to reach node s via the same greedy route to update its value according to

$$x_s(k+1) = \frac{1}{2}[x_s(k) + x_t(k)] \tag{2.3}$$

The algorithm improves the communication complexity for a square grid topology and a random geometric graph (RGG), which are typical connectivity models for nodes in a WSN. For more information on RGGs, the interested reader is advised to read [34]. In fact, Benezit et al. [13] were able to modify the geographic algorithm to get an extra reduction in the communication complexity for a RGG, by averaging all the nodes that lie in the

route (i.e., all the nodes that lie between node s and node t in our above scheme).

There are many disadvantages associated with geographic gossip algorithms [14]. The main disadvantage is the increased overhead because nodes now need to know geographic locations. Storage and computation resources for the routing protocol increase as n increases, in addition to higher probability of losing packets due to the existence of longer routes [12].

Broadcast gossip algorithms [14] use the broadcast nature of wireless communications to achieve a faster rate of convergence to a consensus value, without the need of knowing the geographic locations of nodes. Thus they are trying to achieve better results than those achieved by randomized gossip, without introducing implementation overhead and complexity that characterize geographic gossip algorithms. In the model proposed in [14], each node possesses a clock which ticks at the time of a rate μ Poisson process. Thus, the inter-tick times at every single node are exponentially distributed. Suppose that the k-th clock to tick belongs to node i, then this node activates and broadcasts its current value, $x_i(k)$ to the network. All the nodes within a predefined radius R of node i successfully receive the gossip message. They update their values according to

$$x_j(k+1) = \gamma x_j(k) + (1-\gamma)x_i(k) \tag{2.4}$$

where γ is the mixing parameter, taking values between 0 and 1.

All other nodes, including node i, update their value according to

$$x_j(k+1) = x_j(k) \tag{2.5}$$

This algorithm was proved to achieve consensus with probability 1. However, the sum of the state values is not preserved at every iteration, but the convergence value is in the

neighborhood of the desired value. Simulation results show that broadcast gossip has a faster convergence rate when compared to the convergence rate of both randomized and geographic gossip.

Greedy gossip with eavesdropping (GGE) algorithms, like their broadcast counterparts, use the broadcast nature of wireless communication to improve upon the rate of standard gossip algorithms [15]. The basis for GGE is the fact that other neighbors overhear what messages are being exchanged by the active nodes, but this is not exploited in randomized or geographic gossip.

The GGE algorithm described by Üstebay et al. [15] can be described as follows: The initial value at each node is $x_i(0)$ and each node broadcasts its own value at every iteration. Each node i maintains its local variable, $x_i(k)$ and a copy of the values of its one hop neighbors $x_j(k)$. At the k-th iteration, node s_k is chosen uniformly at random. This node will check the values at each of its neighbors, and then will gossip with node t_k that has the most different value from its own, i.e., $t_k \in \arg\max \{\frac{1}{2}(x_s(k) - x_t(k))^2\}$. Both nodes will then update and broadcast their values. It turns out that GGE has a better performance compared to the randomized gossip and geographic gossip algorithms.

An almost-optimal gossip-based algorithm is presented in [35], which ensures the aggregate value in an n-node network is computed with optimal timing ($O(\log n)$ and optimal number of messages ($O(n \log \log n)$) sent. This is due to the distributed random ranking (DRR) method employed in the algorithm. In DRR, the network is partitioned into a forest of small trees of size $O(\log n)$. Aggregation of each tree occurs at the root of the tree, and then uniform gossip between the different roots guarantees the calculation of the global aggregate.

In [8], the authors present G-GAP, a gossip protocol used for monitoring aggregates, which is scalable and robust to contiguous node failures. They also compare the perfor-

mance of G-GAP with that of the tree-based GAP algorithm [24], and their results show that GAP outperforms G-GAP in terms of accuracy, with and without node failures.

2.4 Neighbor Discovery in a WSN

Neighbor discovery in a WSN is defined in [36] as *"the determination of all nodes with which a given node may communicate directly."* It is one of the earliest steps in initializing a WSN. If during the network's operation the topology changes, then neighbor discovery algorithms can be used again. The discovery algorithm can either be random, where each node transmits at a random time and with high probability discovers all the neighbors by a specific time, or deterministic, where nodes transmit according to a schedule. Important factors have to be taken into account, such as the lack of knowledge of the total number of nodes in the network, coping with collisions, asynchronous operation, different starting time of the discovery process and deciding when to terminate the discovery process.

In [36], an asynchronous, probabilistic discovery algorithm, along with a distributed, companion algorithm, allowing nodes to run the discovery algorithm at different starting times, are introduced. The discovery algorithm can be described as follows. Every node is assumed to have different yet equally-spaced time slots, in which the state of the node in each slot is either transmit or receive. A message m requires a time T_m to be transmitted. A node in the transmit state transmits W copies of message m, which means each slot has a duration $T = WT_m$. If a node is in the receive state in a particular slot, it turns on its receiver and decodes the input. If the received message contains no errors, then the node obtains the transmitting node's identity, and if it wasn't known before to the receiving node, it is added as a neighbor. The advantage of this algorithm is its simplicity and robustness. However, the number of discovered neighbors at a given node may be

incomplete.

In [37], several algorithms for neighbor discovery are designed and analyzed, starting from a synchronous, slotted ALOHA-like discovery algorithm where the number of nodes in the WSN is known, and finishing with an asynchronous algorithm where the number of nodes in the network is unknown and the whole discovery process starts at different times in different nodes. The main advantage here is that this algorithm functions with minimal amounts of information and lack of time synchronization.

The authors [38] take into account the effect of radio interferences on neighbor discovery, which is a realistic issue that needs to be included. A hybrid model is introduced based on two models, a Boolean model and a SINR model. In the Boolean model, simultaneous communications of two or more nodes results in a collision, while the SINR model depends on the ratio of power received by a node y from a node x to the total power received from all nodes, relative to a given threshold to determine successful reception of the signal. The hybrid model introduced uses the Boolean model for internal interferences, and the SINR model for external ones, and allows estimating the expected number of neighbors a given node will discover.

Pagliari et. al [39] look into the implementation of average consensus algorithms in today's sensor networks and a few algorithms based on the Metropolis-Hastings algorithm [40] are included. The basic idea for the algorithm with training depends on sending a number of `hello_messages` and receiving `howAreYou_msg`. This method will be described in detail in Chapter 5.

2.5 Evaluation of Link Quality in a WSN

Radio connectivity is non-uniform, even under ideal conditions, and it is shown that in large scale wireless networks many parameters need to be considered to understand the behavior of WSN protocols, because small effects can actually have a large impact [41]. Many studies have been carried to find a relationship between the wireless link quality and the transmission distance. In [42], a wireless measurement tool called SCALE is introduced and used, and many conclusions were reached. First, there was no clear correlation between the packet delivery and the distance, once the distance is greater than 50% of the communication range of the nodes. Second, the temporal variations of packet delivery were correlated with the mean reception rate of each link. Finally, there is a wide variation in the number of asymmetric links ranging from 5% to 30%. Similar results were achieved in [43]. In [44], a piece-wise empirical model is developed to indicate the relationship between the link quality, transmission distance and the radio power level in a deployment of nodes, which can be used as a guide for practical deployments. Few studies have been carried to analyze the temporal properties of wireless links, with most previous studies focusing on the spatial properties and effects. In [45], the statistical temporal properties of links are studied, and some conclusions were inferred. First, it is better to use the required number of packets as a quality metric instead of the packet reception rate due to high temporal correlations. Second, acknowledgments need to be sent immediately. Third, high quality links are usually persistent, which minimizes the need to update the links. Finally, [46] evaluates link estimation, reliable routing protocols and management of neighborhood tables. The authors provide an empirical model for the reception probability versus the distance. A derivation of this model is used in our simulations, so more details shall be given in Chapter 4.

Chapter 3

Overview of the Collection Tree Protocol (CTP)

Collection trees are used by node deployments to gather data, by providing an unreliable, datagram routing layer. The creation and maintenance of collection trees are provided for by collection protocols. They are widely used in WSN deployments and, in the case of CTP, are implemented in TinyOS [47]. A collection protocol should be able to satisfy three properties: Detecting and repairing routing loops, suppressing duplicate packets and estimating the link quality between any node and its 1-hop neighbour [48]. Each node selects one node to be its parent, through which it forwards the data up the tree, until the data reaches one of the sink, or root, nodes defined in the tree. The selection of parents is done based on a cost metric, which is different for each collection protocol. This chapter will give a description of CTP and its main features. CTP is a complicated protocol and we do not cover all of the details here. Instead, we focus on describing the aspects of CTP most relevant to this thesis. The interested reader is advised to read [16, 48–50] to have a better understanding of CTP and its features that are not mentioned or explained in

detail, for example those relating to congestion of nodes.

3.1 CTP in a Nutshell

CTP is a tree-based collection protocol. It is a packet-based protocol, so packet communications are used to establish the routing information. There are two types of messages transmitted; routing beacons, used to construct the initial tree and maintain it, and data messages, which contain the information being transmitted along the tree. CTP assumes that the data link layer below provides four services: an efficient local broadcast address, synchronous and reliable acknowledgements for unicast packets, a protocol dispatch field to support multiple higher-level protocols, and a single hop source and destination fields [51].

One or more nodes in the network advertise themselves as root (sink) nodes, and other nodes form routes to reach those root nodes. The purpose of CTP is aggregating packets from regular nodes to the root nodes. If there are several root nodes, CTP ensures each regular node has a route to one root node with the lowest routing metric, therefore the protocol is address-free. The routing metric used by CTP is expected transmissions (ETX). This routing metric was designed, implemented and tested by De Couto in his PhD thesis [52], with the main purpose being finding high-throughput routes. The ETX of a link is defined in equation (3.1) below as:

$$ETX = 1/(d_f \times d_r) \tag{3.1}$$

where d_f is the forward ratio of the link, defined as the probability that one packet arrives successfully at the receiver, and d_r is the reverse ratio of the link, defined as the probability that an acknowledgment is received successfully by the sender, given, of course, that the sender's transmission was successful in the first place. The ETX of a node is defined as

the sum of the ETX of its link to its parent and the ETX of its parent, so if a node i has ETX value equal to m, then node i is expected to be able to transmit and deliver a data packet to a root node with a total of m transmissions on average. Root nodes advertise an ETX of zero. Routes are created when nodes are deployed, and are usually only updated if there is an inconsistency detected in the topology. Whenever a node is given the choice of which route to take, it should always choose the one with the minimum ETX.

CTP addresses major causes of packet loss in collection protocols, namely inaccurate link estimations due to highly dynamic links, sampling bias of physical layer information, and formation of loops in dynamic topologies. By incorporating the novel techniques of a hybrid link estimator, an adaptive control traffic rate, and the use of data traffic itself to detect and repair any occurring routing problems, CTP is able to deal with the challenges effectively.

3.2 Link Estimation

Computing the 1-hop ETX between nodes is an essential part of the tasks of CTP, because it is the basis for choosing parent nodes, and determining the path of the data in the network from each node to the root node. CTP uses information from the physical, link and network layers, and combines that with periodic beacons, resulting in a hybrid estimator, which improves the performance compared to a purely beacon-based estimator. The details are available in [49]. Here, we are going to explain how the calculation is done by accounting for both the outgoing and the ingoing links for a node with each of its neighbors.

If a transmitting node is assessing its outgoing link quality to a different, receiving node, then it will send a number of unicast packets, n_u, to the receiving node and will itself receive a number of acknowledgements, n_a. The ratio of n_u over n_a is the quality of the

outgoing link Q_u, i.e.

$$Q_u = \frac{n_u}{n_a} \tag{3.2}$$

The process of assessing outgoing link quality is repeated every window of length w_u packets, which is a set parameter in the system for all nodes. The value of w_u from the TinyOS 2.1 implementation of CTP is 5 [48]. The function that calculates the quality of the overall 1-hop ETX takes into account both the old estimate and the new estimate, as will be shown below. If in a given window zero packets are acknowledged, then the estimate of the outgoing link quality is set to be the total number of unsuccessful packets sent since the last successfully acknowledged packet.

A similar procedure is carried out for estimating the ingoing link quality of a receiving node from another neighbor node. Each node broadcasts beacons to all the nodes in its wireless range either periodically or if specific events take place. Those beacons are used to assess the quality of the ingoing link, Q_b. The number of beacons received by a node from a broadcasting node, n_b, divided by the total number of beacons the transmitting node broadcasts, N_b, i.e.

$$Q_b = \frac{n_b}{N_b} \tag{3.3}$$

Again, this process is repeated every window of length w_b beacons, with the TinyOS 2.1 implementation value being 3 [48]. However an extra step for evaluating the final value of Q_b takes place, which is using an exponential smoothing filter. This is used to account for the old samples and the new value of Q_b in the calculation of the current value. If $k-1$ samples of Q_b have been calculated, and the task at hand is to calculate the k-th sample, then

$$Q_b[k] = \alpha_b \frac{n_b}{N_b} + (1 - \alpha_b) Q_b[k-1] \tag{3.4}$$

where α_b is a smoothing constant in the range [0,1].

For a given node i, if w_u packets are sent to a neighbor node j, or w_b beacons have been received from node j, a new value for Q_u or Q_b is calculated at node i. The 1-hop ETX value of node i for node j is updated using

$$ETX = \alpha_{ETX} Q + (1 - \alpha_{ETX}) ETX_{old} \tag{3.5}$$

where Q is the new calculated value of Q_u or Q_b, and α_{ETX} is a smoothing constant in the range [0,1].

3.3 Route Selection

The path selection procedure uses a distance vector routing algorithm, with ETX as the cost metric. Initial path selection starts from the root node(s), by broadcasting a beacon. This indicates their presence as root nodes since they advertise an ETX of zero. Nodes receiving this beacon add the root to their routing table. Otherwise, they can only count on the root node receiving their beacons successfully and advertising those links in its beacons. A beacon contains several pieces of information, but the main ones for route selection are the node's current parent, and the multihop ETX to reach the root node.

As observed from this brief description, beacons play a crucial role for initial route selection, because a node sends a beacon to declare its presence and fill the neighboring nodes' routing tables. Beacons are also important in topology maintenance as the algorithm is running, because they are used to advertise important information, such as a significant change in a node's multihop ETX.

It is important to choose a proper interval for sending beacons, because there is a tradeoff between energy consumption at a small interval, vs. stale information at a larger interval.

CTP solves this problem by employing adaptive beaconing. Here, beacons are sent at a rate controlled by a variant of the Trickle algorithm [50]. The beaconing interval ranges from a minimum of 64 ms and a maximum of 1 hour, and the beacon is sent at a random time within the second half of the interval. If there is no change in the network (e.g., no new nodes being added and no change in link quality estimates), then the beaconing interval doubles until it reaches the maximum. If an event that may require topology maintenance arises, then the beaconing interval is reset to the minimum value. Three events can lead to this interval reset, which are a node requesting (pulling) information from its neighbors, a node's cost decreased significantly (ETX decrease of 1.5 or more), and finally a node asked to forward data from a node with a lower ETX value than its own.

Parent selection is either repeated periodically, or called asynchronously when certain events occur, such as the transmission of a beacon, the case of an unreachable neighbour or when a node without a route to the root exists. CTP employs hysteresis to switch routes. That is, a node's route is changed only if the new route has a significantly lower cost, which is typically an ETX of 1.5 or more, less than the current route. This is done to ensure the efficiency of the algorithm is not affected by a rapid change in routes.

3.4 Packet Forwarding

A node can either receive data packets from its neighboring nodes, or can generate its own packets. Regardless of the source, those data packets need to be transmitted along the route until they reach the root node. CTP has many mechanisms in place to ensure reliability and efficiency of packet forwarding.

CTP uses a wait interval of 1-2 packet times, or 7-14 ms, to prevent any self-interference occurring. Self-interference can occur if a child node i forwards a packet to its parent node

j, while node j is transmitting a packet to its own parent at the same time.

Another problem that can arise is duplicate packets, which occurs if the acknowledgments for delivered packets are lost. By including a *Time Has Lived (THL)* field in the data packet, CTP is able to detect and suppress those duplicate packets.

CTP uses the datapath to detect the routing loops quickly. When a packet sent from node i is received by node j, node i's multihop ETX is included in the data packet. Node j compares its multihop ETX with that of node i. Since ETX should strictly decrease along the path to the root, if node j's multihop ETX is higher than that of i, then this may indicate a routing loop, so a topology update is due. The beaconing interval is reset to send beacons quickly to synchronize the routing information with the neighboring nodes. The beacon that is sent has the pull flag set, so that each node receiving the beacon from node j resets its own beaconing interval.

There are many backoff timers used by CTP, mainly used in collision avoiding mechanisms. There are three relevant ones for this thesis. The first two are a TX_{OK} backoff timer, and a TX_{NOACK} backoff timer, both of which are used to balance channel reservations between different nodes. The former is used when an acknowledgement for a sent packet is received, while the latter is used in the case where no acknowledgement is received. The third timer is the loop backoff timer which is used once a loop is detected to stop the forwarding of data packets for a certain time interval, giving the chance for CTP to update the network's topology.

Finally, as mentioned in the link estimation section above, the transmission of data packets (and receiving acknowledgments) is used to calculate the outgoing link quality, which is important for estimating the 1-hop ETX between nodes.

3.5 Summary

This chapter explained how CTP works, by describing the link estimation, route selection and packet forwarding procedures. In the next chapter we will describe the modifications we introduced in CTP to allow for data aggregation functionality instead of packet routing only.

Chapter 4

Implemenation of CTP for Data Aggregation

In the last chapter, the main structure and properties of CTP were described to give the reader a strong background of how CTP works. However, CTP is not a data aggregation algorithm, and for this reason, certain changes were needed to allow CTP to transmit the partial aggregates along the tree routes to the root, and then send the global aggregate all the way from the root to all the nodes in the WSN. In this chapter, we shall describe the main differences and modifications we introduced to CTP so that it can function as a data aggregation algorithm.

4.1 Neighbor Discovery, Link Estimation and Initial Tree Formation

Once the WSN is deployed, each node does not know the number or identity of its neighboring nodes. The first step then is for each node to implement a neighbor discovery procedure.

The mechanism we employ is very simple. Each node i initially broadcasts 3 beacons, and if a node j receives any of the beacons the transmitting node i has broadcasted, it considers node i a neighbor. There are no reply messages or beacons sent by node j to node i to indicate successful reception of a beacon. Therefore, there is always the possibility that a node j considers node i as its neighbor, but not vice versa, because node i did not receive a single beacon from node j. This does not affect the formation of the tree or the data aggregation algorithm in any way. In addition to not knowing its neighbors, each node in a newly deployed network has no knowledge of the link qualities, and therefore has to estimate its 1-hop ETX to every neighbor it discovers. The three initial steps, neighbor discovery, link estimation and formation of the initial tree all occur concurrently. As each step is described, a simple 10-node network will be used as an example to illustrate the events. In Figure 4.1 The root, node 5, broadcasts its initial 3 beacons.

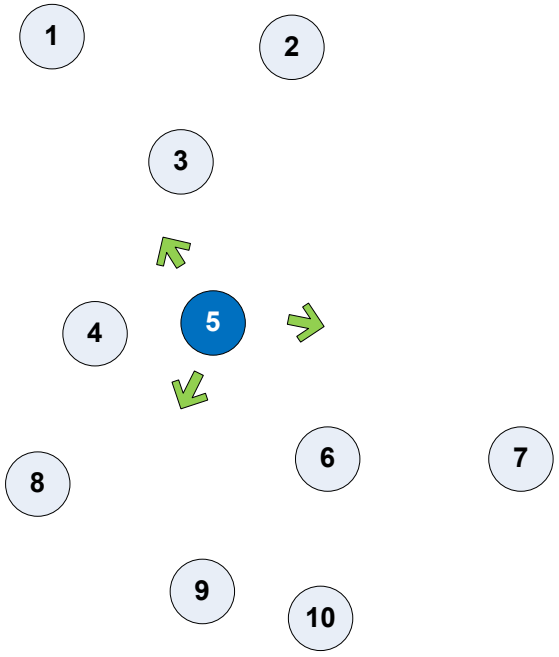

Fig. 4.1 The root node initially broadcasts its 3 beacons.

Any node that successfully receives one of the root's beacons adds the root to its neighborhood table, and attempts to connect to the root, by sending request messages to the root node to become its child. The root node will send an acceptance message to each node whose request message was received successfully and add it to its own neighborhood table. This node is now considered part of the tree structure. If one of the nodes that sent a request message to the root node did not receive an accept message within a pre-specified period of time, it transmits another request message to the root. This is repeated until the root accepts that node as its child. In our example, suppose nodes 3,4 and 6 each successfully receive at least one of node 5's broadcasted beacons, and they all attempt to connect to the root. The root eventually accepts all request messages, and nodes 3,4 and 6 are now all node 5's children and part of the tree. This is shown in Figure 4.2.

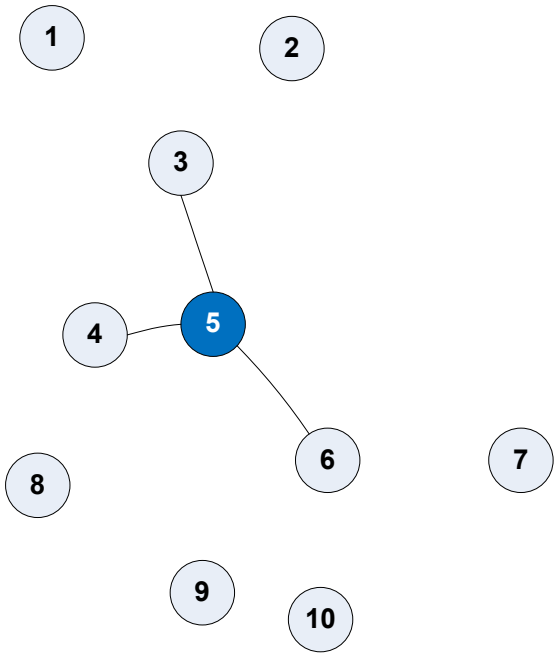

Fig. 4.2 Nodes 3, 4 and 6 join the tree

The step of link estimation for the nodes that just joined the tree follows immediately. Each node has already received a number of beacons from the root node, ranging between 1 and 3. Those that received all 3 beacons successfully can estimate their ingoing link quality, which is essentially 1 at this point, while the others wait to receive more beacons once the data aggregation process starts. Each of those nodes transmits 5 packets to the root node, and depending on the number of successfully received acknowledgements, calculates the outgoing link quality, as mentioned in detail in Section 3.2. Figure 4.3 illustrates nodes 3, 4 and 6 sending unicast packets to the root node. Each red arrow represents a packet being sent to the root.

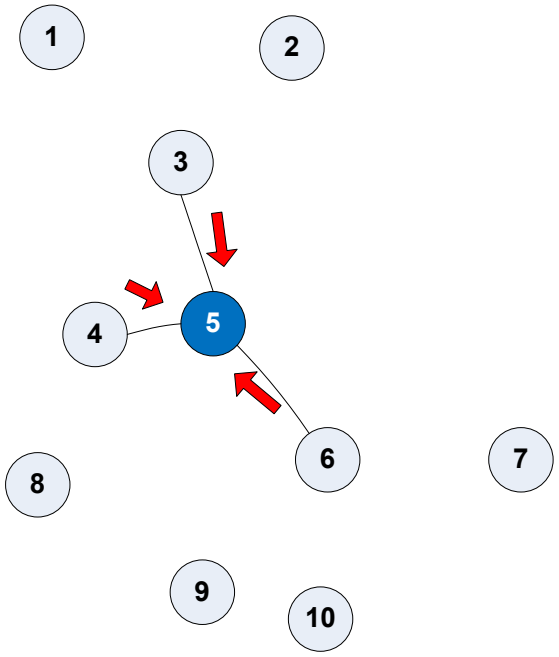

Fig. 4.3 Nodes 3, 4 and 6 each transmits 5 unicast training packets to the root node to estimate the outgoing link quality.

At this point each of the root's children has finished estimating its 1-hop ETX to the root, which is also equal to its multihop ETX. All the root's children broadcast 3

beacons each, advertising their multihop ETX, i.e., the ETX needed to reach the root node. Figure 4.4 shows nodes 3, 4 and 6 broadcasting 3 beacons each.

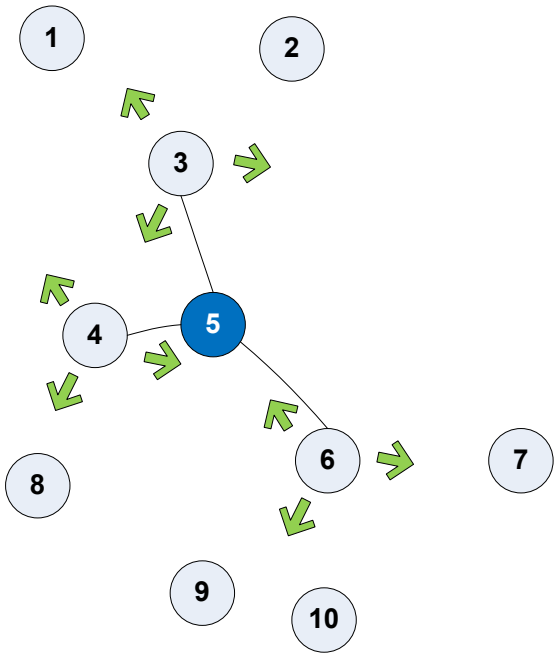

Fig. 4.4 Nodes 3, 4 and 6 broadcast 3 beacons each to alert other nodes to their presence, and notify them of their estimate multihop ETX to the root node.

Nodes that successfully receive at least one beacon from one of the root's children add that node to their neighborhood table. If a node i successfully receives beacons from more than one of the root's child nodes, all of them are included in the neighborhood table, but the one that results in node i having the lowest multihop ETX is selected to be the parent node. In case of a tie, i.e., two or more nodes that result in i having the lowest multihop ETX, one of these nodes will be selected randomly to be the parent. Node i will transmit a request message to that node asking it to be its parent, and waits for an acceptance message before joining the tree. As before, node i keeps transmitting request messages until the other node accepts it as its child. Nodes 1 and 2 have each successfully received one or

more beacons from node 3, and they both become node 3's children. Similarly, nodes 7 and 10 become node 6's children, and node 8 becomes node 4's child. Node 9 has not received a single beacon from any node, and is still not part of the tree. Figure 4.5 illustrates the nodes that joined the tree in this step.

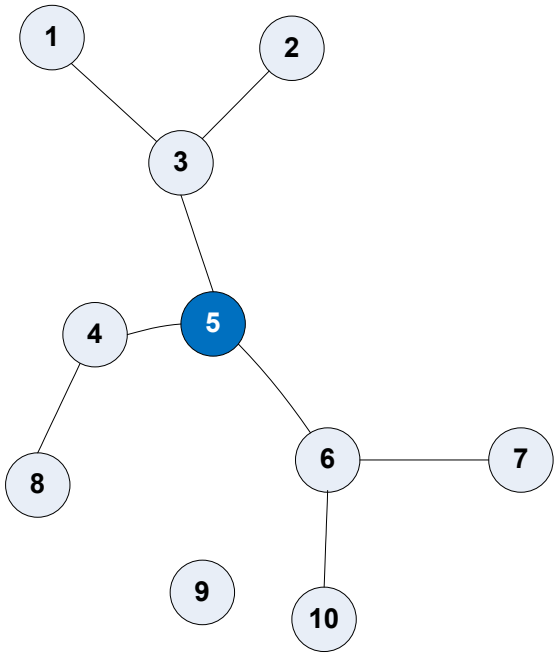

Fig. 4.5 Nodes 1,2,7,8 and 10 successfully received beacons from nodes in the tree, and join the tree as children of those nodes.

Nodes 1,2,7,8 and 10 will now need to estimate their outgoing link quality to each of the nodes in their neighborhood table, by sending the 5 unicast training packets to those nodes. In addition, if the root node received beacons successfully from any of its child nodes in this step, then it will also transmit 5 unicast training packets to each one of them. Node 5 actually successfully receives beacons from all of its children, so it transmits 5 packets to each of nodes 3,4 and 6. This is shown in Figure 4.6.

The nodes that estimated their outgoing link quality are now able to calculate their 1

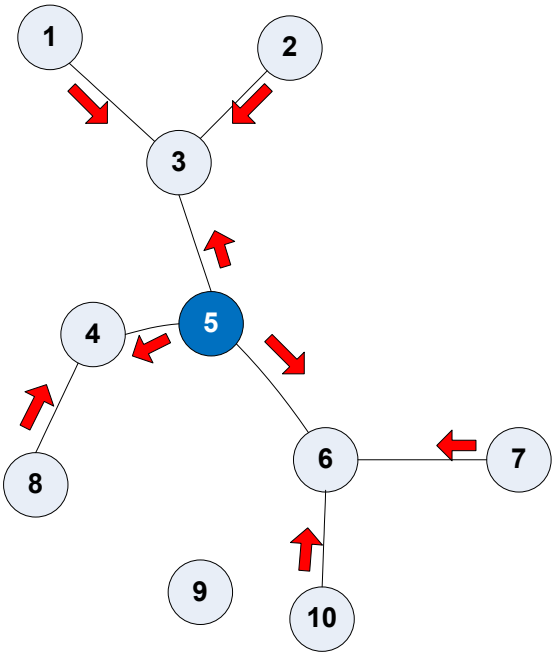

Fig. 4.6 All the nodes that consider nodes 3,4 or 6 as neighbors estimate their outgoing link quality to those 3 nodes.

hop ETX to their parent, and hence calculate their multihop ETX to the root. They now all broadcast 3 beacons each. This is shown in Figure 4.7.

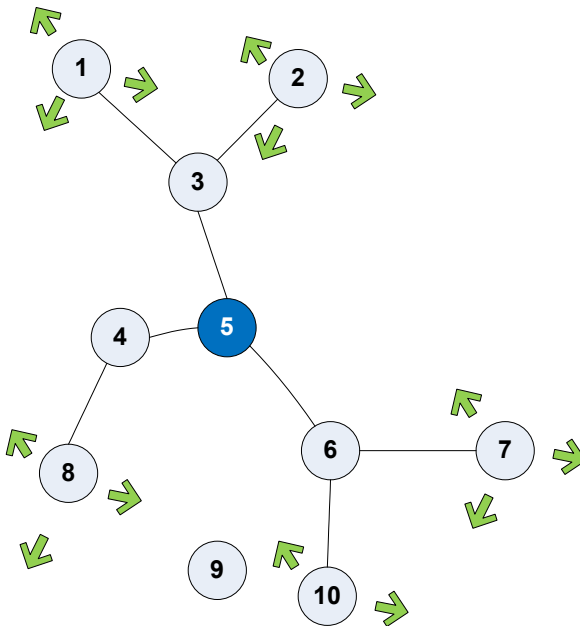

Fig. 4.7 Nodes 1,2,7,8 and 10 broadcast 3 beacons each.

The above steps are repeated until all the nodes have joined the tree, broadcasted their beacons and transmitted their training unicast packets to their neighboring nodes. Figure 4.8 illustrates the final tree. One last step remains at the end, which is for parent nodes to estimate their outgoing link quality to their child nodes (if any). Remember that in our algorithm, a child node definitely received a beacon successfully from its parent, but the parent does not necessarily receive any beacons from its child nodes. This step is important to ensure all link estimates are known both ways for a path in the tree, since the aggregate will be transmitted from the root node to all the nodes in the network.

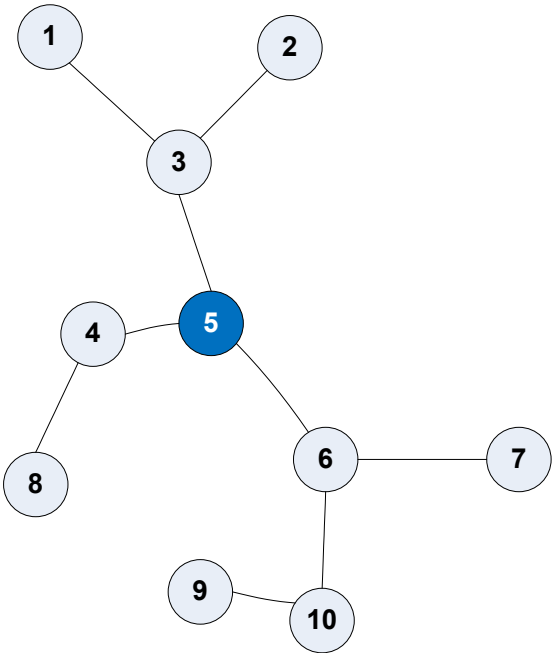

Fig. 4.8 The final tree.

4.2 Loss Model

According to Woo et al. [46], "connectivity is not a simple binary relation, but a statement of the likelihood of successful communication." The loss rate in a link depends on many factors, and the routing protocols need to account for them. The relevant part of the model presented in [46] can be described as follows. A number of nodes sample data and periodically route their data to one sink node. For a number of different pairs of nodes with varying distances, the loss rate is measured, and the mean link quality is calculated as a function of distance. Sensors are placed in a linear arrangement, with a spacing of 2 feet. At any given time, one node transmits 200 packets at a rate of 8 packets/s, with a pre-specified power level. The number of successfully received packets at each receiving node is recorded, and the link quality variation with distance is calculated.

In [3], the author describes the implementation of the Shortest Path Tree (SPT) protocol. An equation derived from the empirical study in [46] is used to calculate the (s-curve) receive probability between different node pairs based on the distances between them. The equation is:

$$p = \frac{d_s}{d_s + e^{4-(Tx_{pow}d_s)}} + 1; \tag{4.1}$$

where p is the reception probability, d_s is the distance between the pair of nodes and Tx_{pow} is the transmission power level. The RF transmit power of the sensor node is less than 1 milli-watt [53]. It is tuned in software to lie in the range 0 to 100, with 0 being the maximum value and 100 being the minimum [46]. In the implementation discussed in [3], the power level is selected randomly at the beginning of the simulation over the range of 0 to 100, and is set equal at all nodes.

In our simulations, we follow a different approach to determine the transmission power level. To guarantee that the network is connected with high probability, a link exists between nodes that are at most at a distance of $\sqrt{\frac{2\log(n)}{n}}$, [54], where n is the number of nodes in the WSN. Therefore, we use this maximum radius value in equation (4.1), to calculate an approximate power level at which all nodes transmit, for a given number of nodes, n. For example, for a network with $n = 100$ nodes distributed uniformly over a 1 km by 1 km region, the critical transmission radius is 303.5 m. The graph of successful receive probability vs. the distance between the transmitting node and the receiving node is shown below in Figure 4.9.

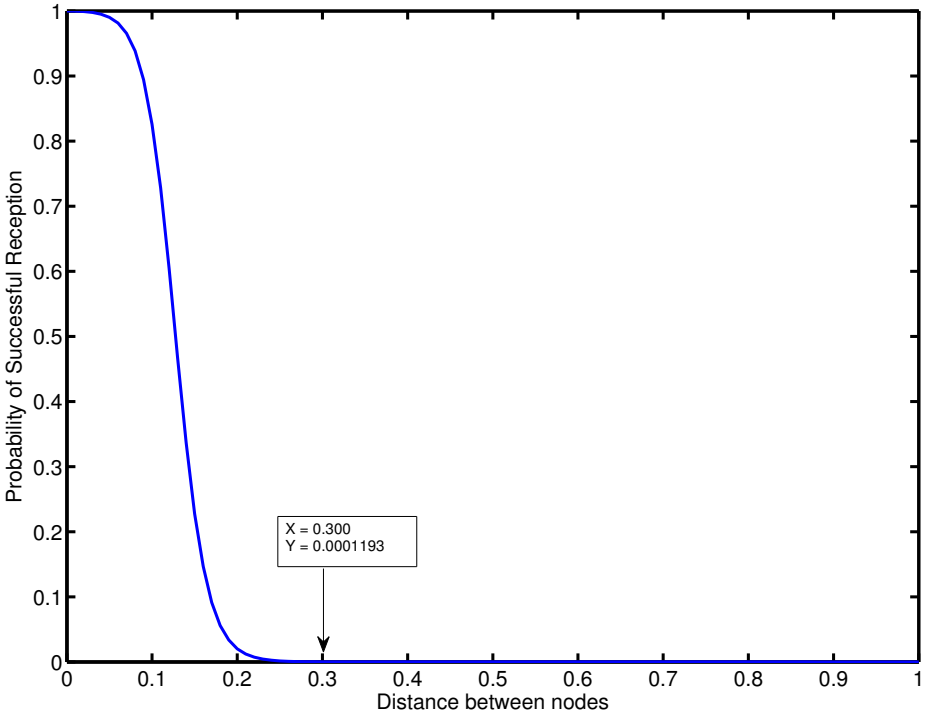

Fig. 4.9 Probability of successful reception vs. distance between the transmitting and receiving node, for $n = 100$.

4.3 Data Aggregation Algorithm

In CTP, data packets are transmitted along the different, established routes to the root node(s). Once a node receives a data packet, it is forwarded using the mechanisms described in Section 3.4. However, in the case of data aggregation, a node has to wait for all it's children to forward their packets, then apply the aggregation function (e.g., average, max, etc.) to the values it receives from its child nodes and its own value. Finally, it forwards the partial aggregate along the route to the root node.

Once the root node received all the data packets from its own child nodes, and applied the aggregation function, then the global aggregate has to be transmitted to all nodes, along the same routes. The broadcast nature of the WSN is used, such that the root broadcasts the aggregate to all its children. A node receiving the global aggregate checks the ID of the sender, and accepts the data packet if it is sent from its parent, and otherwise discards it. Every child node sends an acknowledgement to its parent (in this case, the root node) if it receives the global aggregate successfully. Until the root node receives acknowledgement from all its children, it keeps re-broadcasting the global aggregate. In the meantime, any node that receives the aggregate successfully broadcasts to its children. This broadcast process is repeated until all nodes in the network have received the global aggregate successfully.

4.4 Loop Handling

The CTP handling process described in Section 3.4 is not able to cope with the loops that can occur in the data aggregation algorithm. This is because of one of the modifications we made. In CTP, when a packet sent from node i is received by node j, the multihop ETX of both nodes is compared at node j. ETX should strictly decrease along the path

to the root, so if node j's multihop ETX is higher than that of i, then this may indicate a routing loop.

With data aggregation, a node s knows its parent node, and its child nodes (if any). Node s does not forward the partial aggregate until it receives all the values from its children. So let us give a simple, possible loop scenario involving 3 nodes; s, y, and z. If we assume node y is node s's parent, node z is node y's parent, and node s is node z's parent, then the problem can be seen immediately. Node s knows it has one child, node z, so it waits for it to transmit its value before transmitting to node y. However, node z needs to wait for node y's value, which in turn is waiting for node s's value, reaching a situation of deadlock.

We solved this problem by using beacons, in addition to data packets, to detect routing loops. A beacon contains the sender node's multihop ETX and parent identity [16], so if the receiver of the beacon is the current parent of the transmitting node, it can compare the multihop ETX values. If a loop is suspected, the beaconing interval is reset at the parent node to send beacons quickly to synchronize the routing information with the neighboring nodes. The beacon that is transmitted has the pull flag set, so that any node receiving it also resets its beaconing interval. In addition, the loop backoff timer activates to stop any packets from being forwarded, giving chance for the beacons to update the topology and fix the loop.

The use of beacons required imposing a much lower upper limit on the beaconing interval. If the beaconing rate is really large, it will lead to huge delays before routing loops are resolved in our data aggregation algorithm. Therefore, based on simulation results, we decided to use 10,000 seconds as the maximum beaconing interval instead of the 1 hour interval used in CTP.

4.5 Lack of Communication Between Child and Parent Nodes

One issue that arises when data aggregation is implemented is the following. Suppose that once the initial tree is formed, node i has node j as its only child node. As the algorithm is running and data aggregation is taking place, node j receives a beacon from node m, informing it that it has a much lower multihop ETX than node i, and hence node j requests to become node m's child and is no longer node i's child. There is always a possibility that node i does not receive any of node j's beacons for some time, and therefore it is not aware that it no longer has a child, i.e., it does not know that it can transmit its value to its parent because it does not need to wait for node j to transmit its value to it. This can lead to unnecessary delays, especially once the beaconing interval reaches its maximum, and can have major effects on performance.

To handle this problem, we decided to impose a limit on the maximum allowed time for which a parent node does not hear from a child node, i.e., does not receive any beacons from it. This is only enforced at a node that still needs to transmit its value along the tree to the root node, and is supposedly waiting to receive its child node's value. The maximum time used in our simulations is 20 seconds. This value is twice the maximum beaconing interval, which ensures that the parent node has at least 2 chances to receive beacons from its child node. If it does not receive anything in this amount of time, then it is very likely that it no longer has any child nodes.

4.6 Summary

This chapter described how certain modifications were applied to CTP and the reasoning behind each one of them so that data aggregation can be completed successfully, and the global aggregate broadcasted from the root node along the tree routes to every single node

in the WSN. This involved introducing training data packets and routing beacons, tree formation messages and acknowledgements, loop handling mechanisms, and the main data aggregation algorithm itself.

Chapter 5

Implemenation of the Gossip Algorithms in a Lossy Environment

This chapter will give a brief overview of randomized, pairwise gossip algorithm introduced by Boyd et al. in [7], and broadcast gossip algorithms introduced by Aysal et. al. in [14]. Those algorithms assumes that there is no loss in the network, so we describe their implementation in a lossy environment, using the same model described in Section 4.2. In addition, the neighbor discovery algorithm for randomized gossip is described in detail.

5.1 Implementation of Randomized Gossip

5.1.1 Overview of Randomized Gossip

The most popular example of gossip algorithms for data aggregation is their use in distributed averaging [55]. The problem can be formulated as follows. For a WSN containing n nodes, let $G = (V, E)$ denote the connected network's communication graph, where V is the vertex set containing the n nodes, and E is the edge set. Edge $(i, j) \in E$ if node i and

node j communicate directly, i.e., they are in the wireless range of each other. Each node has an initial value, reflecting the measurement it has taken. For example, if the WSN's application is collecting temperature data, then the initial value at each node would be the temperature it has measured at its location. The goal is to calculate the average of the initial values of all nodes in a distributed manner, also known as the average consensus problem. Thus, let $x(0) = [x_1(0), x_2(0), ..., x_n(0)]^T \in \mathbb{R}$ denote the vector of initial values, where the ith component is the initial value of the ith node. The average is denoted as $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i(0)$, and this is the value that the algorithm needs to converge to after a certain number of iterations. The algorithm stops when a pre-defined level of error is achieved relative to the initial average in the network.

Boyd et al. [7] presented two different time models: synchronous and asynchronous. The common factor between both time models is that a node only communicates with one other node at a given time. Each node wakes up in every given time slot to communicate with one of its neighbors uniformly at random. The transmitting node sends its value to the receiving node, and the receiving node sends back its own value to the transmitter. Both nodes will then update their value by averaging their old value with that received from the other node. On the other hand, the asynchronous time model assumes that each node has a clock that ticks independently at a rate 1 Poisson process. Thus the inter-tick times are exponentially distributed and independent across nodes and across time. Asynchronous algorithms do not require all the nodes in the network to have synchronized clocks [56]. An important advantage they provide when compared to their synchronous counterparts for general applications is that they allow for greater implementation flexibility [56]. Therefore, in any given time instant, only one node wakes up and gossips with one of its neighbors, and both update their value as described in equation (2.1). It was shown that the algorithm converges for both time models, that is each node's estimate will converge to the initial

average value of the network, if the network is connected and gossiping occurs frequently enough [55].

In this thesis, we shall be working with the asynchronous time model only. This to ensure a fair comparison with CTP, since CTP does not assume any synchronization between nodes. Beside, in real life deployments, it is more likely that nodes will not be synchronized. From this point onwards, randomized gossip will mean asynchronous, pairwise, randomized gossip.

The common stopping condition for randomized gossip is achieving a pre-defined error value, which can either be absolute or relative. In this thesis, when we use the term error, we explicitly mean the relative error. At transmission k, the relative error is defined as:

$$e(k) = \frac{||x(k) - \bar{x}||}{||x(0) - \bar{x}||} \tag{5.1}$$

5.1.2 Randomized Gossip with Lossy Links

The randomized gossip algorithm described in [7] does not take into account losses of transmitted messages and the need for retransmission arising from varying wireless link qualities. On the contrary, it just assumes that once a node wakes up and randomly communicates with a neighbor, both of them will receive the other's value and update successfully. This leads to an inaccurate estimate of the total number of messages needed to achieve a certain level of error between the true average and the estimated average. Many extra transmissions will be needed, and this is another modification we include to the algorithm in [7].

The loss model used here is the one described in Section 4.2, and is implemented in the same manner it was implemented in CTP. Thus, if at time k, node i's clock ticks, and it uniformly at random chose neighbor j to communicate with, it will transmit its value to

node j. If this transmission is unsuccessful, then nothing happens, and neither node i nor node j update their value. The next time node i wakes up, it will again choose a neighbor uniformly at random, so node j is as likely to be chosen as any other of node i's neighbors. On the other hand if node i's transmission reached node j successfully, then node j will transmit its own value to node i. Node j updates its value, while for node i two scenarios are possible. It can receive node j's value successfully, compute the average and update its value, or it does not receive node j's value, and therefore nothing happens.

5.1.3 Neighbor Discovery Algorithm for Randomized Gossip

Boyd et al. [7] do not mention any neighbor discovery mechanism. They just describes how the algorithm functions assuming that each node knows its neighbors. However, in a typical practical deployment, nodes have no information regarding who their neighbors are, or even the total number of nodes in the network [37]. Therefore we have to implement neighbor discovery ourselves. Since CTP's neighbor discovery method is based upon transmitting beacons, as described in Chapters 3, we opted for using a similar mechanism. Pagliari and Scaglione [39] look into the implementation of a few algorithms based on the Metropolis-Hastings algorithm [40]. One of the algorithms involves the use of training messages to discover neighbors, and calculate each node's degree and its neighbors' degrees. We are not interested in the degrees of nodes in our simulations, so we shall not worry about it. This is how the rest of the discovery mechanism we implemented works. A pre-determined number of `hello_messages` are sent by each node. Every time the transmitting node receives an answer from a node, in the form of a `howAreYou_msg`, it updates its list of neighbors. Therefore, we see that if a `hello_message` is received successfully, a node will transmit a `howAreYou_msg`. However, if a node receives a `howAreYou_msg` from another node, it only update its own list of neighbors, but does not send a reply. It is important to note that

the messages are broadcasts, so several nodes can receive each message. In our algorithm, if a `howAreYou_msg` is received by any node other than the node that transmitted the `hello_message`, it is ignored.

In [39], a `hello_message` can be transmitted while the averaging algorithm is running, and any node that receives it replies with a `howAreYou_msg`, and also updates its knowledge of the local degree of that neighbor. Since we mentioned we are not interested in degrees of nodes, the `hello_messages` broadcasted while averaging is taking place do not have a real benefit for us. Therefore, all nodes perform neighbor discovery and transmit all the pre-specified number of `hello_messages` before the averaging phase starts. Here we assume that all nodes are deployed before the data collection process begins, and there is no need for re-doing the neighbor discovery process. We specify 3 `hello_messages` to be sent per node. This is all in line with the CTP neighbor discovery and link estimation procedure, where 3 beacons are sent for neighbor discovery, and where the whole procedure is completed before data collection commences.

It is important to realize that in many cases more nodes may be deployed later on in the network's life to cover a larger area or to collect more readings, There is also the possibility that some links may be down initially, and therefore nodes will not discover all their neighbors. In those cases, the neighbor discovery process should be repeated periodically to ensure that nodes have up-to-date information on the identity of their neighbors.

The above neighbor discovery mechanism works well when the wireless links are assumed to be ideal. However, once the links are lossy, this results in the following problem. Let us assume that the wireless link between i and j is really lossy, i.e., the probability of successful transmission each way is about 30%. If node i broadcasted all its 3 `hello_messages`, and node j only receives one of them, then nodes i and j become neighbors. If at time k, node j's clock ticks, and it uniformly at random chose i to communicate with, j will

transmit its value to node i. The probability of j's value reaching i successfully is still 30%, but the probability of i's value also reaching j successfully is only 9%. Therefore it is a likely scenario that only i receives the packet successfully, and updates its value, while j receives nothing. Over many iterations and with many different nodes suffering the same problem, the average value in the network will be modified, and thus we do not converge to the true consensus value. Therefore, we decided to impose a more stringent condition on the neighbor discovery algorithm. A node has to receive at least 2 `hello_messages` or 2 `howAreYou_msgs` from another node to consider it a neighbor. We tried the requirement of 3 messages instead of 2, but this meant that the network will be disconnected in most simulation cases, so 2 messages was the best compromise.

5.2 Implementation of Broadcast Gossip

5.2.1 Overview of Broadcast Gossip

Broadcast gossip algorithms [14] use the broadcast nature of wireless communications to achieve a faster rate of convergence to a consensus value. In the model proposed in [14], each node possesses a clock which ticks at the time of a rate μ Poisson process. Thus, the inter-tick times at every single node are exponentially distributed. We use μ equal to 1 here to be consistent with the value used in randomized gossip. In every iteration, one node is chosen uniformly at random, and this node broadcasts its current value to the network. Suppose that at time k node i wakes up, then this node activates and broadcasts its current value, $x_i(k)$ to the network. All the nodes within a predefined radius R of node i are assumed to be within the broadcast range of node i, and they all successfully receive the gossip message. The R value used here is $\sqrt{\frac{2\log(n)}{n}}$, [54], for the same reason explained in the description of the lossy model in Section 4.2. The set of nodes that received the value

of node i will then calculate a weighted average of that value and their own value. If node j is one of those nodes, it will update its value according to equation (2.4), reproduced below:

$$x_j(k+1) = \gamma x_j(k) + (1-\gamma)x_i(k) \tag{5.2}$$

where γ is the mixing parameter, taking values between 0 and 1. It has been shown in [14] that γ equal to 0.5 is the optimal value when it comes to convergence speed, and hence this is the value we shall be using in our simulations. All other nodes, including node i itself, do not update their own value.

This algorithm was proved to achieve consensus with probability 1. However, the average value is not preserved at every iteration, because each node updates its own value independently using the broadcast value and its current value. Hence the convergence value can be different from the true average, but is in the neighborhood of the desired value [14]. Simulation results show that broadcast gossip has a faster convergence rate than randomized gossip, i.e., it is able to achieve a low per-node variance.

From the discussion in the previous paragraph, we see that we cannot use the relative error in the average value defined in equation (5.3) as a stopping condition. Instead, we check that the per-node variance has reached a pre-specified value. At transmission k, this is defined as:

$$\frac{||x(k) - \bar{x}(k)||^2}{||x(0) - \bar{x}(0)||^2} \tag{5.3}$$

where $\bar{x}(k) = \frac{1}{n}\sum_{i=1}^{n} x_i(k)$, is the average of the nodes' values at time k.

5.2.2 Broadcast Gossip with Lossy Links

The broadcast gossip algorithm described in [14] assumes that once a node wakes up and broadcasts its value, all the nodes within its wireless range successfully receive its value and update. It does not take into account varying wireless link qualities. Certain modifications are needed to the algorithm for the same reasons we modified randomized gossip to account for lossy links.

The loss model used is again the one described in section 4.2, and it is very simple to incorporate here. Let us assume that at time k, node i's clock ticks, and it broadcasts its value to the network. For every node in the range of i, the value can either reach that node successfully or not depending on the link quality. If a node receives i's value successfully, an update in the receiving node's value occurs, otherwise the node receives nothing and does not update its value.

5.3 Summary

This chapter described how we implemented both randomized gossip and broadcast gossip algorithms in a lossy environment. The working mechanism of each algorithm was described so that the reader can understand the differences between them. The neighbor discovery technique employed by randomized gossip was also explained in detail, and the reasons behind our choices were discussed. Broadcast gossip does not depend on a node discovering its neighbors, so no additional techniques needed implementation.

Chapter 6

Simulations & Results

In this chapter, we describe the simulations we carried out to compare CTP to both randomized gossip and broadcast gossip. We report the results for two main sets of simulation experiments. The first set of simulations tests the effect of increasing the network size on the performance of each algorithm, i.e., the number of transmissions required to stop. The second set of simulations tests the effect of link failure on the performance of each algorithm. An initialization field contains the set of data which the nodes sample to obtain their initial value, and different fields for the gossip algorithms have an important impact on the performance. Thus, we incorporate several initializations in our simulations. We conclude that CTP performs better than randomized gossip in all experiments. When compared to broadcast gossip, CTP performs similarly for small network sizes, except for a spike initialization where broadcast gossip is much better. However, CTP is worse than broadcast gossip once the network becomes large regardless of the initialization field.

6.1 CTP Simulation Parameters

In Chapters 3 and 4, CTP's different mechanisms were described, including the many different parameters that need to be defined and fixed when simulations are run. Table 6.1 summarizes all of the parameters and values used in our simulations of CTP for data aggregation. Most of those values are identical to the ones used in [16, 48, 49]. The ones that we modified or added have the ▷ symbol preceding the parameter's name. The first parameter we added was the Parent Unreachable Threshold and is set to an ETX of 7. This value ensures not changing routes too quickly nor waste many transmissions on a poor link. The second parameter that was added was the time limit for not receiving beacons from a child or a parent node and is set to 20,000 ms. This was an important parameter so that parents who no longer have children and yet do not know of this update need not wait to receive any values before transmitting up the tree. The ones that were only modified are the following. The beaconing interval is limited to vary between 70 ms and 10,000 ms, instead of varying between 64 ms and one hour, because beacons are used to help nodes detect routing loops, so a long interval harms performance, and a short one can lead to unnecessary beacon transmissions. The Parent Switch Threshold and Significant Decrease Threshold are set to an ETX of 3 instead of 1.5, because a smaller value leads to unnecessary message transmissions, and a larger value can leave more optimal routes undetected.

Table 6.1 Parameters used to implement and simulate CTP for data aggregation

Parameter & Description	Value
α_b (Smoothing constant used in equation (3.4))	0.9
α_{ETX} (Smoothing constant used in equation (3.5))	0.9
w_b (Number of beacons needed to update ETX)	3
w_a (Number of packets needed to update ETX)	5
TX_{OK} Backoff (in ms)	15.6-30.3
TX_{NOACK} Backoff (in ms)	15.6-30.3
LOOP Backoff (in ms)	62.5 - 124
Packet Time (in ms)	7
▷ Minimal beaconing interval (in ms)	70
▷ Maximal beaconing interval (in ms)	10,000
▷ Parent Switch Threshold (ETX value)	3
▷ Significant Decrease Threshold (ETX value)	3
▷ Parent Unreachable Threshold (ETX value)	7
▷ Time allowed for not receiving beacons from child/parent (in ms)	20,000

6.2 Effect Of Network Size on Performance

6.2.1 Simulations' Description

We simulate CTP for data aggregation, randomized gossip and broadcast gossip, on networks with different sizes. All links are assumed to be lossy and the network sizes tested are 50, 75, 100, 150, 200 and 500 nodes. All messages transmitted by each algorithm are accounted for, including but not limited to messages exchanged for neighbor discovery, messages transmitted to calculate the aggregate and messages re-transmitted due to dropped packets in the WSN. For CTP, the algorithm stops when all the nodes in the network successfully receive the aggregate. For randomized gossip, the algorithm stops when a pre-specified error value is reached. Finally, broadcast gossip stops when the pre-specified per-node variance is reached. Initially, the error value is chosen to be 0.001, and for fair comparison, the per-node variance is also chosen to be 0.001. For all plots, RG stands for

randomized gossip, while BG stands for broadcast gossip.

Before showing any results, it is worth mentioning a few issues that sometimes arise due to the lossy nature of the wireless links. In CTP, a leaf node (a node with no children) might have only one neighbor, which is its parent. After some time it might actually exceed the parent unreachable threshold value, and therefore it cannot transmit its value up the tree, or receive the aggregate that the root calculated and transmitted back down the tree. In this case, not all nodes receive the aggregate, and CTP does not terminate, so we exclude it from our results, because a large number of messages will be sent to update the tree topology to no avail. In randomized gossip, the problem that occurs is that the consensus value deviates from the true average because one node can receive a gossip message and update its value while the other one does not. Therefore, we decide to change the pre-specified error value here to 0.01 only, but the per-node variance stopping condition for broadcast gossip is kept as 0.001. There are certain simulations where we use randomized gossip with ideal links, i.e., the success rate is 100% along those links. For those cases, the pre-specified error value is 0.001. Therefore, for each simulation the nature of the links used for randomized gossip will be explicitly stated. For CTP and broadcast gossip, lossy links are always used.

All results reported correspond to the average over 1000 Monte Carlo trials. We report the average number of messages needed to compute and disseminate the average measurement value to all nodes across the network (i.e., to achieve the average consensus). The network topology is a RGG augmented with lossy links. What this means is the following. In a RGG, all nodes within a defined radius R from a specific node are in the wireless range of that node, and are assumed to be its neighbors. However, due to the neighbor discovery mechanisms we use and the lossy wireless link model not all nodes are discovered, resulting in this form of a RGG. It is important to note that we incorporate this R value in our loss

model, as explained previously in Chapters 4 and 5.

Different initializations for the nodes' values are used in different simulations, because some initializations can lead to a faster convergence of the gossip algorithms compared to others. Note that the performance of CTP does not depend on the initial value at each node. We used the following three initialization fields:

1. Gaussian Bumps: Four two-dimensional Gaussian functions are mixed. Their amplitudes are 7, 8, 18 and 25, respectively. The Gaussian peaks are centered at (0.65,0.3), (0.19,0.19),(0.3,0.4) and (0.15,0.75), respectively. In addition, the functions' variances are equal to 0.0078, 0.0137, 0.0048 and 0.0138, respectively. The nodes sample this field containing the Gaussian bumps.

2. Spike: All nodes, except one, have a value of zero. In this field, the value of the node that contains the spike will have to spread across the network [14].

3. Slope: The field which the nodes sample varies in a linear manner.

6.2.2 Results and Discussions

The goal of our first simulation is to compare CTP, broadcast gossip and randomized gossip with lossy links. The nodes initialization is a field containing Gaussian bumps. Randomized gossip is an iterative algorithm, and in our simulations the number of iterations is chosen so that the final relative mean squared error is below 0.01. Similarly, broadcast is run until the per-node variance is below 0.001. Figure 6.1 shows the number of messages transmitted vs. the number of nodes for the three algorithms.

The graph shows that CTP and broadcast gossip have comparable performance for networks with 50, 75 and 100 nodes, respectively, with broadcast gossip performing slightly better. As the network size increases, the performance of broadcast gossip is much better

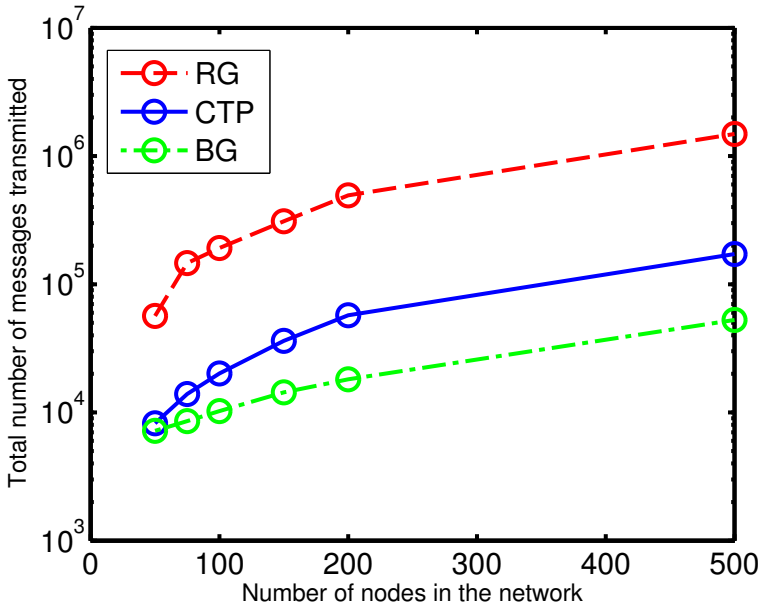

Fig. 6.1 Total number of messages transmitted for the algorithm to stop vs. different network sizes. The algorithms compared are CTP, randomized gossip, and broadcast gossip. All algorithms were simulated with lossy links. The Gaussian bumps initialization is used for the two gossip algorithms.

than CTP, requiring only 52,727 messages to stop compared to 172,140 messages needed by CTP. The reason behind this is twofold. First, as the network size increases, CTP's neighborhood discovery mechanism, link estimation and initial tree formation requires a much larger number of messages to be transmitted. Broadcast gossip requires none of the above steps, and therefore is not affected by the increasing size of the network. Second, a larger number of beacons are sent to maintain and update the tree topology. Broadcast gossip does not send any beacons.

Randomized gossip with lossy links has a much worse performance than both CTP and broadcast gossip, even for a small network size of 75 nodes. This can be explained due to randomized gossip being slow in general, even when links are ideal (i.e., not lossy). To verify this, we simulate randomized gossip with ideal links and compare its performance with CTP and broadcast gossip when they are simulated with lossy links. The Gaussian bumps initialization field was used again. However, the number of iterations is chosen so that the final relative mean squared error is below 0.001. The results are shown in Figure 6.2.

We can clearly see that even when randomized gossip has ideal links, which is an unrealistic assumption, CTP and broadcast gossip perform much better than randomized gossip. There are many wasted transmissions where a node tries to communicate with a neighbor whose value is close to its own, and hence no significant information is obtained from that gossip round. In CTP, each node waits for its children to transmit their values to it before applying data aggregation, and hence all successful transmissions are useful in that sense. In broadcast gossip, several nodes receive a single node's value in one transmission, so even if one node does not really benefit, others can modify their value significantly.

The next simulation we carried out was to see if broadcast gossip performs better than CTP for different node initializations. Randomized gossip performs much worse than CTP

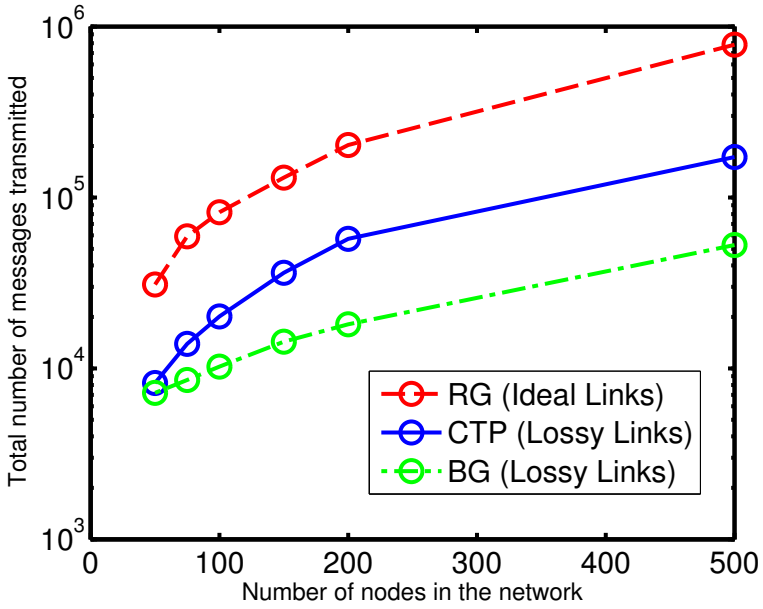

Fig. 6.2 Total number of messages transmitted for the algorithm to stop vs. different network sizes. The algorithms compared are CTP with lossy links, randomized gossip with ideal links, and broadcast gossip with lossy links. The Gaussian bumps initialization is used for two gossip algorithms. The two differences between this Figure and Figure 6.1 are the following. First, in this Figure randomized gossip is simulated with ideal links while in Figure 6.1 randomized gossip is simulated with lossy links. Second, in this Figure the error level required for randomized gossip is 0.001 instead of 0.01 in the previous one.

for all initializations, with both ideal links and lossy links, so the results are not shown in the following plot. Figure 6.3 shows the results when CTP is compared to broadcast gossip with a spike and then with a slope.

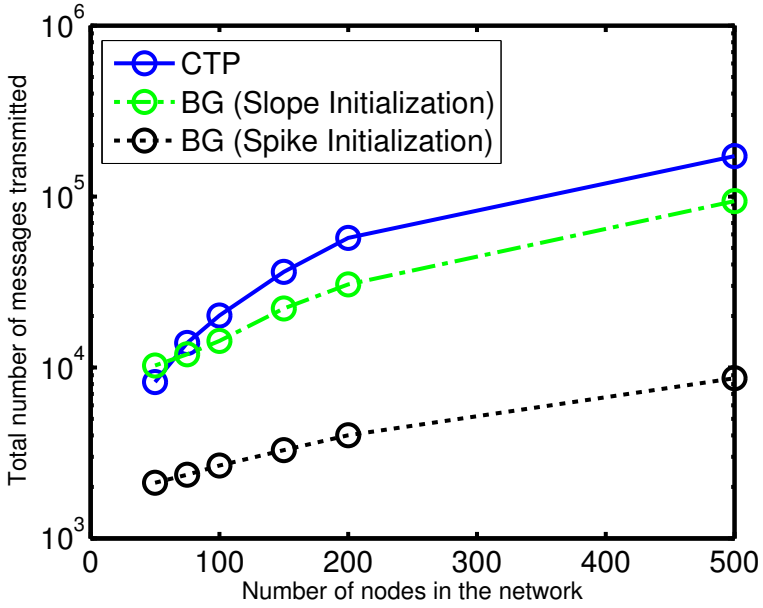

Fig. 6.3 Total number of messages transmitted for the algorithm to stop vs. different network sizes. The algorithms compared are CTP and broadcast gossip. The spike initialization field is used first for the broadcast gossip algorithm, then the slope initialization field is used.

With a spike initialization, broadcast gossip performs much better than CTP for different network sizes, with the performance gap more apparent as the network size increases. In fact, this initialization results in the least number of messages transmitted by broadcast gossip among the three initializations (Gaussian bumps, spike and slope). However, things change when a slope initialization is used. CTP performs better than broadcast gossip for networks with 50 nodes. As the network size increases, broadcast gossip goes back to being the superior algorithm. Therefore, we deduce that as the network size increases, the extra messages that CTP needs to send as mentioned above actually limit its performance.

Table 6.2 summarizes the average number of total messages sent by each algorithm we simulate and indicates the standard deviation of the number of messages required to achieve a consensus for different network sizes. We can make two important observations. First, there is a pretty large variation in the number of messages required by the different algorithms. Second, in some instances, although the mean of one algorithm is lower, the standard deviation may be higher.

Table 6.2 The average total number of messages transmitted by each algorithm and the standard deviation in each value for different network sizes. Note that L preceding an algorithms name stands for Lossy, and I stands for Ideal. Sk is a spike initialization, GB is Gaussian bumps initialization, and Sl is a slope initialization.

	50	**75**	**100**	**150**	**200**	**500**
CTP	8,215±5,289	13,909±7,630	20,146±10,304	32,216±18,753	57,292±29,694	172,140±54,503
L-RG (GB)	56,642±34,687	146,140±84,210	191,360±77,378	283,320±175,455	494,199±219,330	1,485,200±368,690
L-RG (Sk)	53,330±41,817	89,210±59,674	123,750±65,280	189,410±105,649	272,560±172,290	510,920±269,210
L-RG (Sl)	107,790±39,839	142,180±65,568	221,880±63,079	340,780±131,745	587,130±169,880	2,015,600±353,160
I-RG (GB)	30,920±20,829	59,290±40,165	81,930±48,037	130,470±56,312	202,600±73,287	783,840±104,444
I-RG (Sk)	26,630±18,900	44,250±32,166	60,240±34,926	93,133±39,217	125,880±47,633	399,390±105,680
I-RG (Sl)	39,606±23,492	71,365±42,987	101,480±51,333	173,860±67,904	255,800±79,072	1,015,200±122,000
L-BG (GB)	7,182±11,429	8,552±8,747	10,228±9,870	14,320±11,817,694	18,104±12,213	52,727±13,171
L-BG (Sk)	2,117±3,074	2,359±3,382	2,669±3,408	3,281±3,019	4,009±3,704	8,656±6,720
L-BG (Sl)	10,269±14,287	11,940±10,397	14,823±10,328	22,145±13,288	30,573±16,960	94,088±15,624

Table 6.3 shows the breakdown of different types of messages sent in CTP, on average, to perform different tasks for different network sizes. The initial steps are neighbor discovery and initial beaconing for link quality estimation. The value for beacons accounts for the ones transmitted after the initial steps. Table 6.4 shows the number of messages transmitted for neighbor discovery and for aggregation in RG. We conclude that even though CTP is overall better than RG, there is a large amount of overhead involved in setting up the tree, and only a modest number of messages are actual data aggregation messages. This may be tolerable in settings where the link qualities are fairly stable, but as we will see next, when the tree needs to be maintained because of time-varying link qualities, CTP incurs a major performance hit.

Table 6.3 The average number of messages transmitted by CTP in different stages of the data aggregation process

	50	**75**	**100**	**150**	**200**	**500**
Initial Steps	2,328	3,986	5,797	9,872	14,221	44,887
Tree Formation	716	1,207	1,673	2,597	3,747	9,956
Beacons	422	877	1,832	3,595	5,840	15,793
Topology Update	1,268	2,232	3,689	7,925	22,047	86,391
Data Aggregation	3,481	4,060	7,155	12,227	11,437	15,113
Total	8,215	13,909	20,146	36,216	57,292	172,140

Table 6.4 The average number of messages transmitted by RG when GB is used

	50	**75**	**100**	**150**	**200**	**500**
Neighbor Discovery	706	1,190	1,720	2,830	4,200	12,840
Data Aggregation	30,220	58,100	80,210	127,640	198,400	771,000
Total	30,920	59,290	81,930	130,470	202,600	783,840

Table 6.5 shows the percentage of simulations for CTP which converged, i.e., the ones where the aggregate value was computed and disseminated to all nodes. CTP does not terminate when one or more nodes become disconnected from their parent nodes and cannot reconnect to the tree, so a certain percentage did not converge. For most of the cases for which this occurs, it was because one node in the network topology had only one neighbor node, and once it loses connection with that node because of the lossy nature of the links, the network is disconnected and CTP does not converge.

6.3 Effect of Time-Varying Link Quality on Performance

6.3.1 Simulations' Description

In this set of experiments, we simulate CTP, randomized gossip and broadcast gossip, on networks where links qualities vary over time. The state of each link evolves according to an independent (from link to link) two-state discrete time Markov chain (DTMC), where

Table 6.5 Percentage of trials that terminate for CTP. Trials that terminate are the ones where all nodes in the network successfully receive the value of the aggregate

Network Size	Percentage of Trials Terminating
50	14.8
75	15.5
100	23.1
150	22.2
200	29.0
500	31.1

the states are on and off (for each link). This is a random process where the future state depends on the current state only and not on the past [57]. Figure 6.4 shows the state transition diagram of a our two-state DTMC, where state 0 represents a link being off, and state 1 represents a link being on.

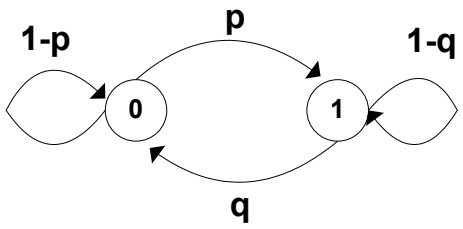

Fig. 6.4 Two-State DTMC model for link failure. State 0 represents the link being on and State 1 represents the link being off. p represents the probability of a link changing status from off to on, i.e., $p = \Pr(\text{on}|\text{off})$, while q represents the probability that a link that is on goes off, i.e., $q = \Pr(\text{off}|\text{on})$. If a link is off, it stays off with probability 1-p, and if it is on it stays on with probability 1-q.

Table 6.6 shows the three probability sets we used, and the probabilites of states changing from on to off and vice versa. Those probability sets were chosen to given a good indication of the performance of each algorithm under different link characteristics. Low

Link Variability (LLV) indicates favorable conditions where functioning links keep working fine the next round with a probability of 90%, and even if a link fails, it functions again the next round with a probability of 70%. With the Moderate Link Variability (MLV) set, we kept the probability of a failed link to function again the next round at 70%, but increased the probability of failure for a functioning link from 10% to 30%. When the High Link Variability (HLV) set is used, any link can function or fail the next round with a probability of 50%. We tried testing for values of Pr(off|on) = 70% and 90%, but CTP did not terminate when such probabilities were used, regardless of the value of p chosen. It is worth mentioning that both randomized gossip and broadcast gossip algorithms were able to converge under different initializations and network sizes when such values of q were used.

Table 6.6 The link failure probability sets used in our simulations

	Pr(on|off)	**Pr(off|on)**
Low Link Variability (LLV)	70%	10%
Moderate Link Variability (MLV)	70%	30%
High Link Variability (HLV)	50%	50%

All links are still assumed to be lossy and the network sizes tested are 50, 75, 100, 150 and 200 nodes. In our simulations, a network of 500 nodes running CTP with different link failure probabilities rarely converges, hence we are not able to get sufficient information to draw any conclusions and so we do not simulate for networks larger than 200 in this set of experiments. All other simulation parameters and initialization fields described in section 6.2 remain the same. Again all results reported correspond to the average over 1000 Monte Carlo trials.

6.3.2 Results & Analysis

We first compare CTP, broadcast gossip and randomized gossip with lossy links. The nodes initialization is a field containing Gaussian bumps. In our simulations of randomized gossip the number of iterations is chosen so that the final relative mean squared error is below 0.01. Similarly, broadcast gossip is run until the per-node variance is below 0.001. Figure 6.5 shows the number of messages transmitted vs. the number of nodes for the three algorithms when the LLV set (cf. Table 6.6) is used in the simulations.

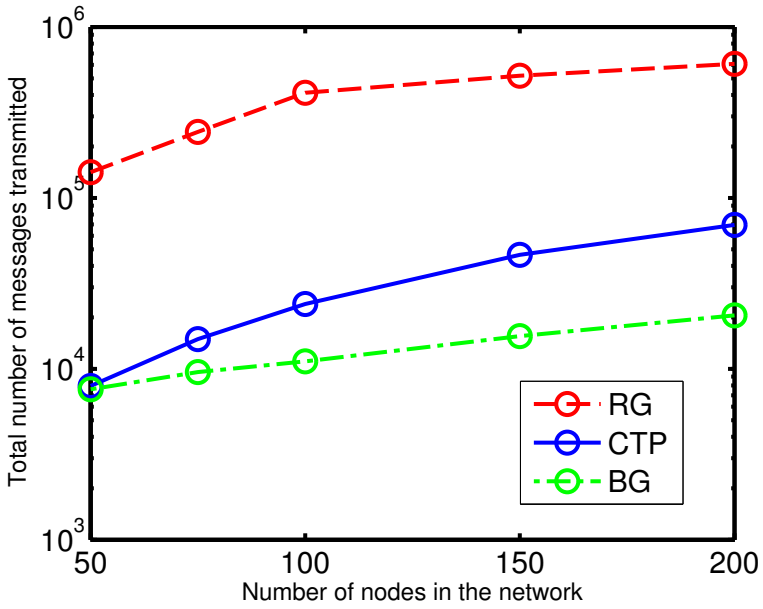

Fig. 6.5 Total number of messages transmitted for the algorithm to stop vs. different network sizes. The algorithms compared are CTP, randomized gossip, and broadcast gossip. All algorithms were simulated with lossy links. The quality of links varied according to the LLV set. The Gaussian bumps initialization is used for the two gossip algorithms.

Figure 6.5 shows that CTP and broadcast gossip have comparable performance for networks with 50 nodes. As the network gets larger to 75, 100, 150 and 200 nodes, respectively, broadcast gossip performs much better than CTP, with the biggest difference occurring at

200. In addition to the two reasons given for why broadcast gossip performs better than CTP as network size increases, another factor comes into play here. For CTP, when a node tries to communicate over a link that is down, it obviously fails, which has two effects. First, it increases the number of wasted transmissions. Second and more important, the 1 hop ETX estimate to that node increases. If the node is down for a long time, the ETX value might exceed the Parent Unreachable Threshold, which leads to more beacons transmitted to update the tree topology, in addition to more `Parent Request` and `Child Accept` messages being exchanged.

Before going into more simulation to confirm the above analysis, let us take a minute to look at the performance of randomized gossip with lossy links. We see that it has a much worse performance than both CTP and broadcast gossip. Even for a network with 50 nodes, randomized gossip requires almost twenty times the number of messages used by CTP. This can be explained by two reasons. The first is that only messages are wasted when a node tries to communicate with a neighbor whose value is close to its own and hence no significant information is obtained from that gossip round, which is the same for the case when link failures do not occur. What is unique to this case is the second reason, which is there are so many wasted transmissions when a node tries to communicate with a neighbor but the message never goes through, so for some time the node value does not change as it keeps trying to gossip with its neighbors.

We again try to see if simulating randomized gossip with ideal links while keeping both CTP and broadcast gossip with lossy links makes a huge difference. LLV set (cf. Table 6.6) is used here again, and all conditions are kept the same as the previous experiment, except that the error value for randomized gossip to terminate is set to 0.001. The results are shown in Figure 6.6. The simulations reveal that even when randomized gossip has ideal links, which is an unrealistic assumption, CTP and broadcast gossip perform much better

in the presence of link failure.

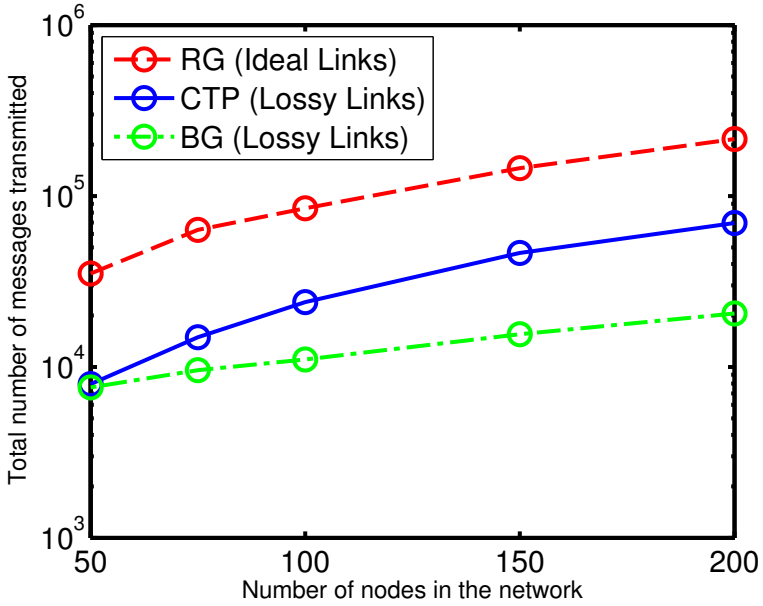

Fig. 6.6 Total number of messages transmitted for the algorithm to stop vs. different network sizes. The algorithms compared are CTP with lossy links, randomized gossip with ideal links, and broadcast gossip with lossy links. The quality of links varied according to the LLV set. The Gaussian bumps initialization is used for the two gossip algorithms. There are two differences between this Figure and Figure 6.5. First, in this Figure randomized gossip is simulated with ideal links while in Figure 6.5 randomized gossip is simulated with lossy links. Second, in this Figure the error level required for randomized gossip is 0.001 instead of 0.01 in the previous one.

From our simulations in section 6.2, we saw that a spike initialization field for gossip algorithms requires the least number of messages for the algorithm to converge amongst the three initialization fields we use in these simulations. Therefore, we want to see how randomized gossip with ideal links and a spike initialization compares to CTP. All other conditions are kept the same as the ones in the previous two experiments. The error level specified for randomized gossip to terminate is still 0.001. Figure 6.7 shows the results.

CTP still performs better than randomized gossip even though randomized gossip is

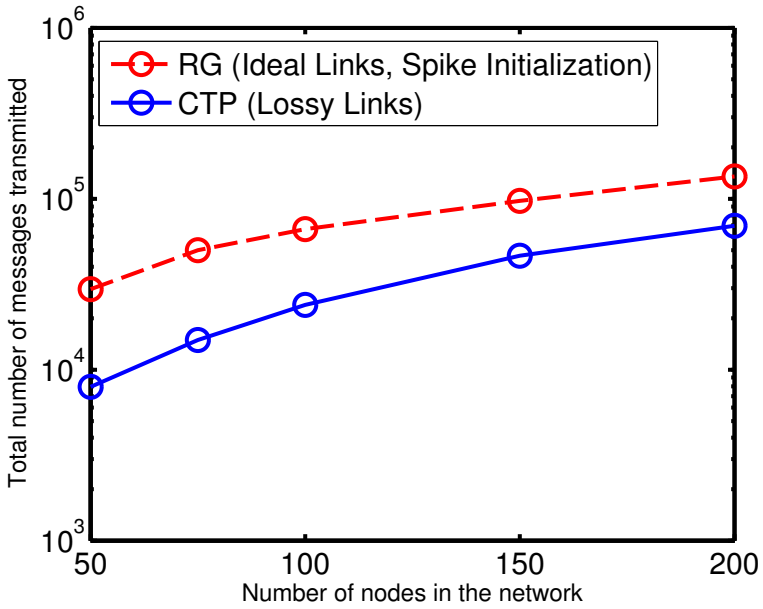

Fig. 6.7 Total number of messages transmitted for the algorithm to stop vs. different network sizes. The algorithms compared are CTP with lossy links and randomized gossip with ideal links and a spike initialization field. The quality of links varied according to the LLV set.

given the most favorable conditions. As a last step, we simulate CTP with lossy links vs. randomized gossip with ideal links using the HLV Set (cf. Table 6.6) to check if this is true for different probabilities of links failing and turning back on. The results are shown in Figure 6.8.

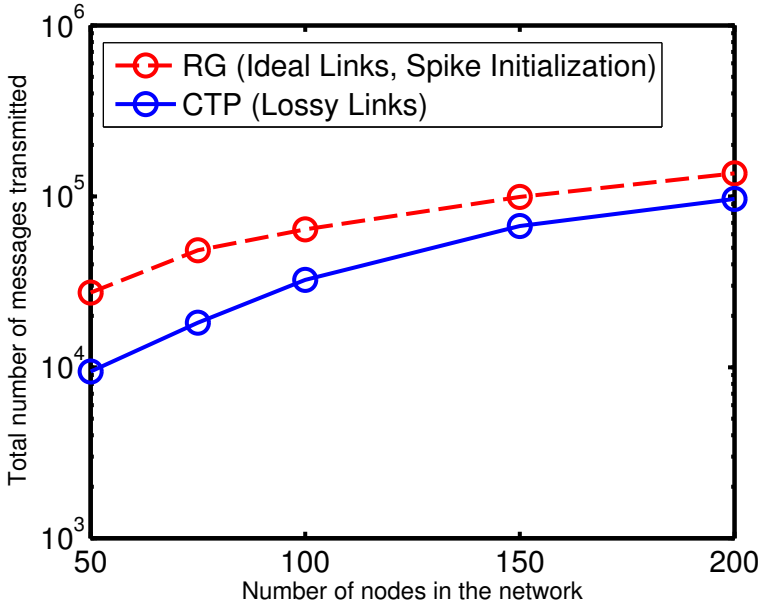

Fig. 6.8 Total number of messages transmitted for the algorithm to stop vs. different network sizes. The algorithms compared are CTP with lossy links and randomized gossip with ideal links and a spike initialization field. The quality of links varied according to the LLV set. The difference between this figure and Figure 6.7 are the probabilities of links failing and waking up.

Again, CTP has the upper hand and is better than randomized gossip with ideal links and a spike initialization field, even at higher probabilities of links failing and higher probabilities of failed links not functioning properly again. Therefore we can safely say that CTP performs better than randomized gossip for different network sizes and different probabilities of link failure.

Now we return to comparing CTP with broadcast gossip, both having lossy links. In Figure 6.5 we saw how CTP performs compared to broadcast gossip when the Gaussian

bumps initialization and the LLV set is used. The next simulation we conducted was to see if broadcast gossip performs better than CTP for different node initializations with the same error probability set. Figure 6.9 illustrates how CTP compares to broadcast gossip when the LLV set is used and the nodes are initialized according to a spike first, and then to a slope.

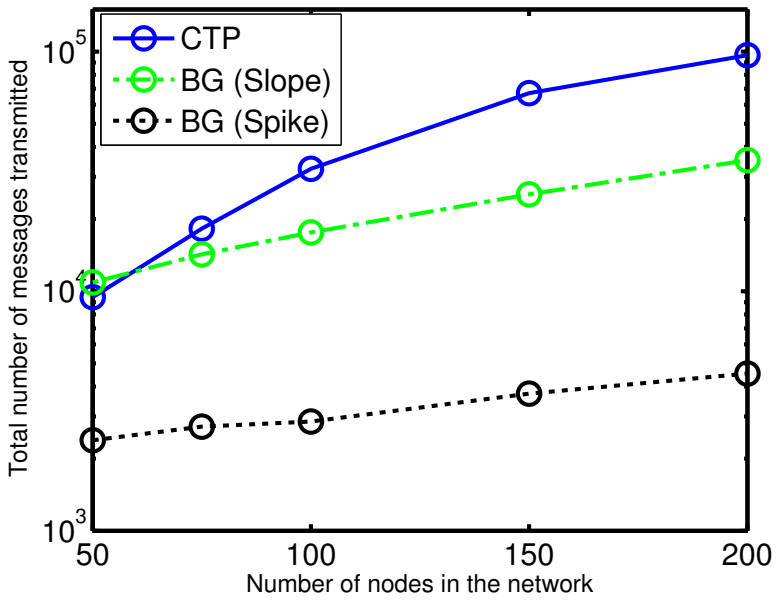

Fig. 6.9 Total number of messages transmitted for the algorithm to stop vs. different network sizes. The algorithms compared are CTP and broadcast gossip. The LLV set is implemented. The spike initialization field is used first for the broadcast gossip algorithm, then the slope field is used.

Broadcast gossip with a spike initialization always outperforms CTP. However, when broadcast gossip employs a slope initialization, CTP performs better for a network of size 50 nodes, and performs very similarly for networks containing 75 and 100 nodes. For a network containing 200 nodes, broadcast gossip performs much better than CTP. This can be explained as follows. The slope initialization leads to the worst performance in terms of number of messages to converge. For small network sizes, CTP has the advantage that it is

not affected by the initialization field. However as the network size grows, broadcast gossip starts to lead because it does not have to worry about the overhead messages employed by CTP, such as routing beacons.

To check if this is always true, we carry further simulations. Figure 6.10 shows the results when broadcast gossip is simulated with the three different initialization fields vs. CTP, using lossy links and the MLV set, while Figure 6.11 shows the results when broadcast gossip is simulated with the three different initialization fields vs. CTP, using lossy links and the HLV set.

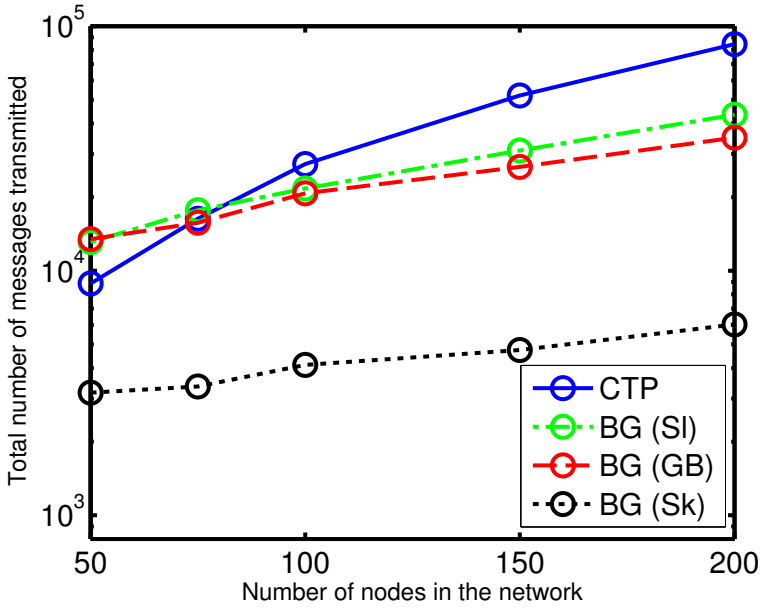

Fig. 6.10 Total number of messages transmitted for the algorithm to stop vs. different network sizes. The algorithms compared are CTP and broadcast gossip. The MLV set is implemented. All three initialization fields for broadcast gossip are tested.

It turns out that our conclusions were quite right, and CTP always performs better than broadcast gossip with a slope initialization for a small network containing 50 nodes. In fact, we see that as the link failure probability increases, and the probability of failed

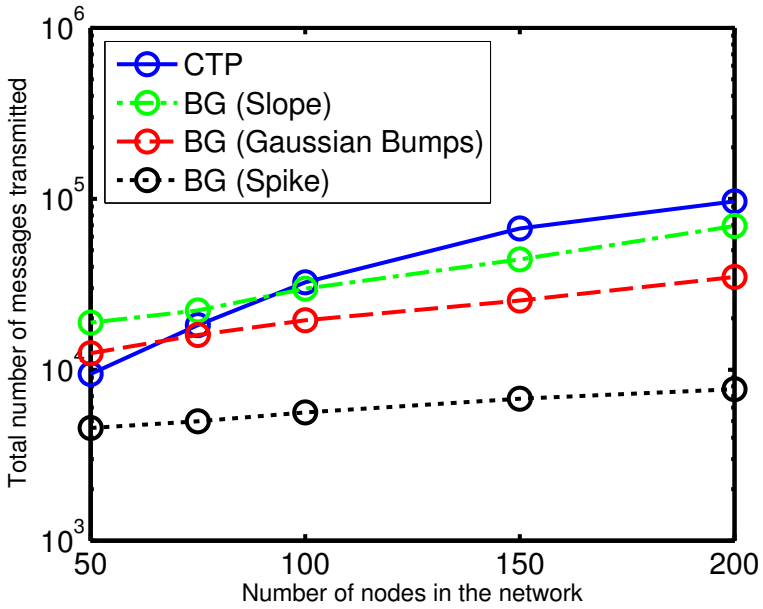

Fig. 6.11 Total number of messages transmitted for the algorithm to stop vs. different network sizes. The algorithms compared are CTP and broadcast gossip. The HLV set is implemented. All three initialization fields for broadcast gossip are tested.

links staying off for a longer time also increases, i.e., Pr(off|on) increases and Pr(on|off) decreases in our DTMC model, CTP is better even for graphs containing 75 nodes. The spike initialization provides the most favorable conditions for gossip algorithms to converge amongst our initializations, and therefore it is always the best to use and is much better than CTP. Gaussian bumps initialization is somewhere in the middle, but it still leads to better performance when employed with broadcast gossip compared to CTP.

Table 6.7 summarizes the average number of total messages sent by each algorithm we simulate and indicates the standard deviation of the number of messages required to achieve a consensus for different network sizes when the LLV set is used, while Table 6.8 shows the same information but when the HLV set is used. We see that again there is a pretty large variation in the number of messages required by the different algorithms at different link failure probabilities.

Table 6.7 The average total number of messages transmitted by each algorithm and the standard deviation in each value for different network sizes. Note that L preceeding an algorithms name stands for Lossy. The LLV set and Gaussian bumps initialization is used.

	50	**75**	**100**	**150**	**200**
CTP	7,932±5,052	14,910±10,002	23,963±13,651	46,391±22,178	69,423±34,271
L-BG (GB)	7,584±8,698	9,577±11,684	11,062±8,425	15,550±10,391	20,560±15,655
L-BG (Sk)	2,389±3,397	2,722±3,520	2,858±2,987	3,783±3,218	4,532±4,409
L-BG (Sl)	10,874±14,511	14,247±14,890	17,578±12,816	25,385±14,721	35,217±19,972

Table 6.8 The average total number of messages transmitted by each algorithm and the standard deviation in each value for different network sizes. Note that L preceding an algorithms name stands for Lossy. The HLV set is used.

	50	**75**	**100**	**150**	**200**
CTP	9,435±4,002	18,246±7,146	32,425±23,452	67,113±28,716	96,521±37,318
L-BG (GB)	12,484±14,328	15,954±16,292	19,450±16,995	25,382±17,012	34,916±19,384
L-BG (Sk)	4,563±6,027	4,985±7,134	5,624±6,041	6,761±6,532	7,691±6,625
L-BG (Sl)	18,826±18,733	22,212±16,6114	29,802±18,593	44,237±20,014	60,378±24,196

Table 6.9 summarizes the different numbers of messages sent in CTP, on average, to perform different tasks for different network sizes, when the LLV set is used.

Table 6.9 The average number of messages transmitted by CTP in different stages of the data aggregation process when the LLV set is used.

	50	**75**	**100**	**150**	**200**
Neighbor Discovery & Link Estimation	2,333	3,973	5,804	9,850	14,266
Initial Tree Formation Messages	875	1,075	1,742	2,747	3,802
Total Beacons (excluding neighbor discovery)	1,258	2,585	4,142	8,608	13,065
Topology Update Messages	1,049	3,039	5,777	12,175	18,826
Data Aggregation Messages	2,417	4,238	6,498	13,011	19,464
Total	7932	14,910	23,963	46,391	69,423

Table 6.10 summarizes the different numbers of messages sent in CTP, on average, to perform different tasks for different network sizes when the HLV set is used.

Table 6.10 The number of messages transmitted by CTP in different stages of the data aggregation process when the HLV set is used

	50	**75**	**100**	**150**	**200**
Neighbor Discovery & Link Estimation	2,423	4,053	5,816	10,054	12,866
Initial Tree Formation Messages	779	1,322	1,401	3,046	5,327
Total Beacons (excluding neighbor discovery)	1,552	3,350	4,930	13,037	18,752
Topology Update Messages	1,296	3,591	9,983	17,515	26,182
Data Aggregation Messages	3,385	5,924	10,295	23,461	33,394
Total	9,435	18,240	32,425	67,113	96,521

By comparing Table 6.9 to Table 6.10, we see that there are a few major differences. Many more beacons, topology update messages and data aggregation messages are required when the probability of link failure is higher. More beacons are sent to update the nodes of the new estimates. Topology update messages increase greatly because many more nodes keep changing parents due to the failed links. Similarly, more messages are needed to transmit the nodes' values along the tree to the root and then to distribute it among the nodes because each node requires more messages on average to successfully transmit its value.

Finally, Table 6.11 shows the percentage of simulations for CTP which did converge when different probability sets are used.

Table 6.11 Percentage of trials that terminate for CTP when different Probability Sets are used. These trials are the ones where all nodes in the network successfully receive the value of the aggregate.

Network Size	LLV	MLV	HLV
50	11.7%	8.3%	2.4%
75	15.9%	8.9%	1.5%
100	15.5%	8.1%	1.0%
150	17.7%	7.8%	1.0%
200	15.9%	7.9%	1.0%

By comparing the different values in Table 6.11, we see that the general trend is a decrease in the percentage of trials that terminate successfully as both the link failure probability increases and the probability of a failed link not turning on increases, for different network sizes. This is an expected result, since for CTP more unsuccessful messages would be sent, which increase the 1 hop ETX between neighbor nodes, and therefore nodes will try to change their parents more often. If this process is unsuccessful, because of links being down and other links being very lossy, then the network becomes disconnected.

6.4 Inaccuracies in Broadcast Gossip

From the previous results, we were able to conclude that BG is superior to both CTP and RG, especially as the network becomes larger. However, as mentioned previously in Section 5.2, the convergence value when BG is used can be different from the true average in the network. Therefore, it is important to analyze how different the final mean in the network is compared to the true average for networks with different sizes, different initializations and varying link qualities.

Let $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i(0)$ denote the true average of the initial values at each node, and let $\hat{x}$ denote the value to which Broadcast Gossip converges. Then the *percentage difference* is given by $100 \times \frac{\bar{x}-\hat{x}}{\bar{x}}$.

The histogram in Figure 6.12 illustrates the percentage difference between the true average and the convergence value over 1000 Monte Carlo trials in a network with 50 nodes using BG with lossy links and a Gaussian bumps initialization.

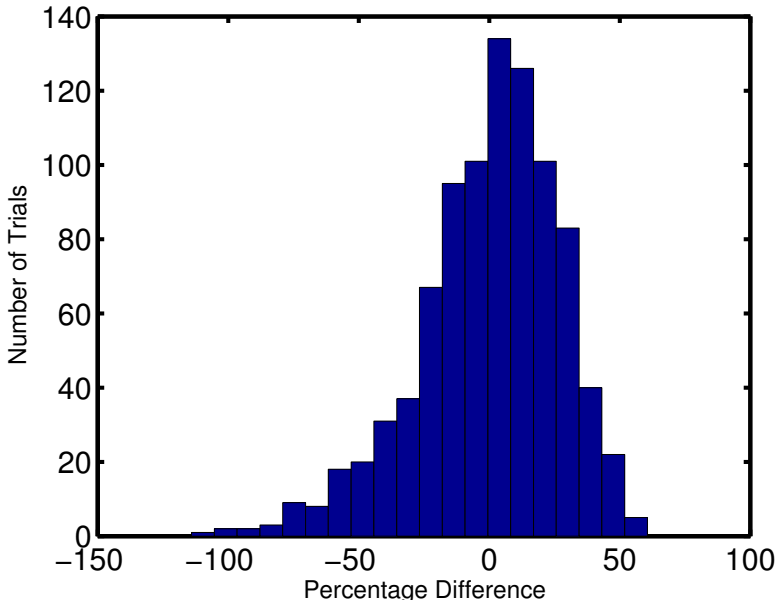

Fig. 6.12 Percentage difference between the true average and the convergence value over 1000 Monte Carlo trials in a network with 50 nodes. The algorithm used is BG with lossy links and a Gaussian bumps initialization.

We can see that more than 40% of the convergence values lie within 20% of the true averages in the network. We repeated the same experiment twice using the same exact conditions, but with changing the network size to 150 and 500 nodes, respectively. Figure 6.13 illustrates the result for a network with 150 nodes, and Figure 6.14 illustrates the result for a network with 500 nodes.

By analyzing those 3 figures, we can conclude that as the network size increases, the

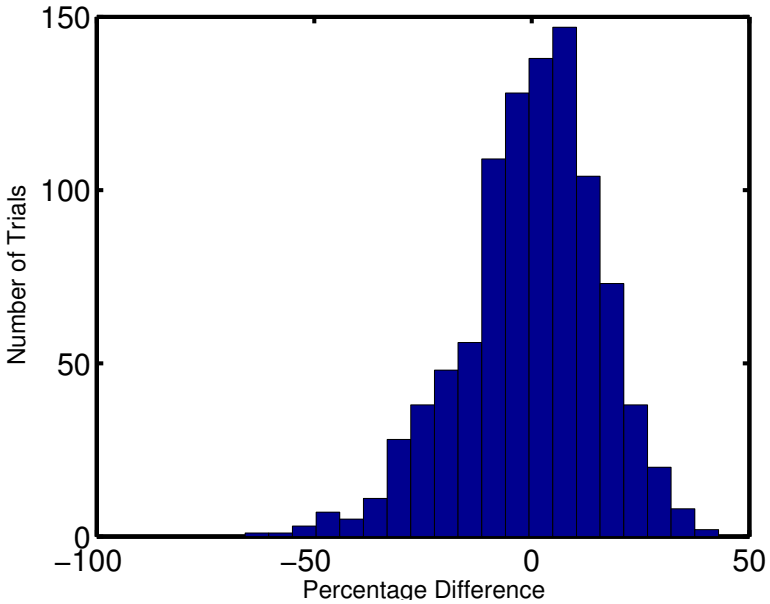

Fig. 6.13 Percentage difference between the true average and the convergence value over 1000 Monte Carlo trials in a network with 150 nodes. The algorithm used is BG with lossy links and a Gaussian bumps initialization.

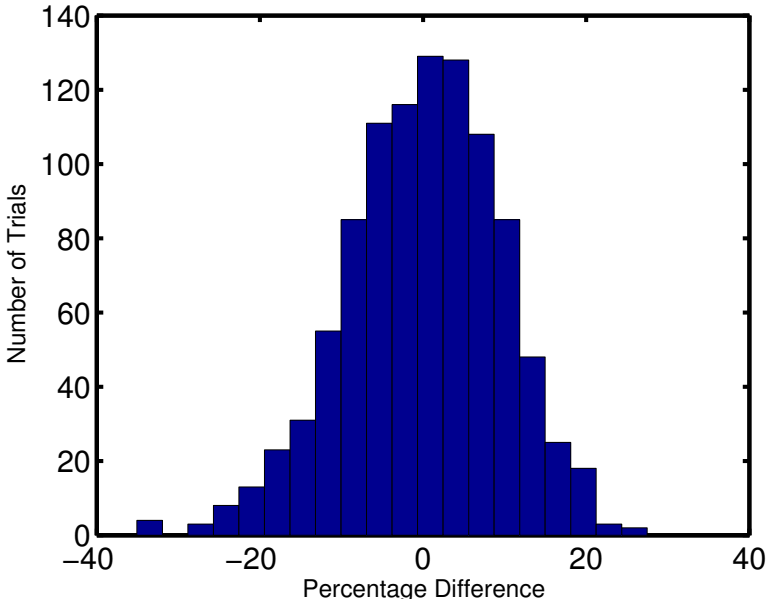

Fig. 6.14 Percentage difference between the true average and the convergence value over 1000 Monte Carlo trials in a network with 500 nodes. The algorithm used is BG with lossy links and a Gaussian bumps initialization.

percentage difference between the true average and the convergence value decreases. This is because the larger the network, more messages are transmitted before the algorithm converges, so the convergence value is closer to the true average.

Next, we check the effect of varying link quality on the percentage difference between the true average and the convergence value over 1000 Monte Carlo trials in a network with 150 nodes using BG with lossy links and a Gaussian bumps initialization. Figure 6.15 illustrates the result when the LLV set is used, while Figure 6.16 illustrates the result when the HLV set is used.

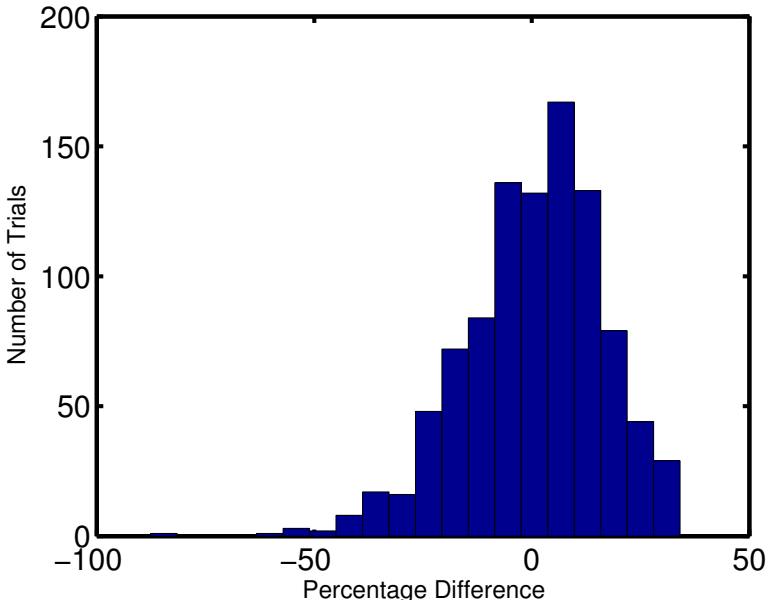

Fig. 6.15 Percentage difference between the true average and the convergence value over 1000 Monte Carlo trials in a network with 150 nodes. The algorithm used is BG with lossy links. A Gaussian bumps initialization and the LLV set are used.

As the link quality in the network gets worse, i.e., the probability of links failing and staying down increases, the percentage difference between the convergence value and the true average in the network decreases. Again, this is because more transmissions are

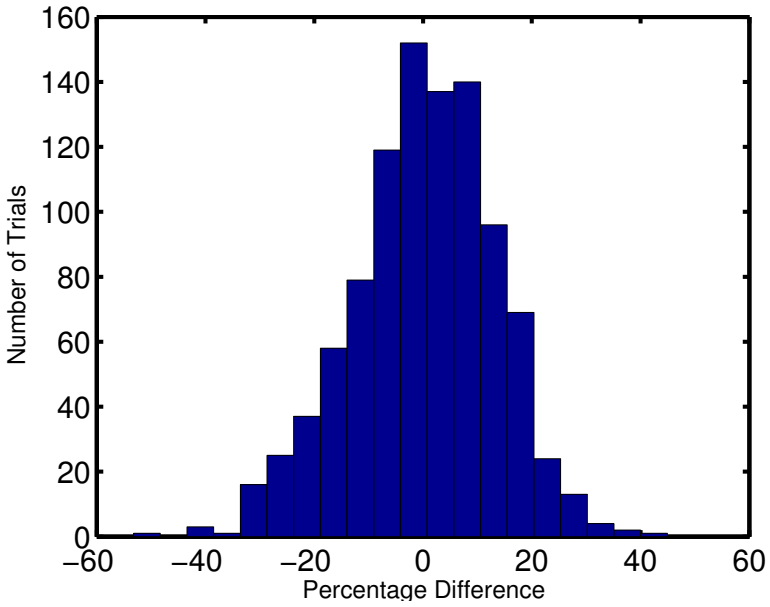

Fig. 6.16 Percentage difference between the true average and the convergence value over 1000 Monte Carlo trials in a network with 150 nodes. The algorithm used is BG with lossy links. A Gaussian bumps initialization and the HLV set are used.

needed for the algorithm to converge, thus giving it time to have more exchanges amongst the nodes, which leads to the convergence value being closer to the initial true average.

6.5 Summary

This chapter summarized the experiments we conducted to compare CTP to randomized gossip and broadcast gossip, including the assumptions and models used. The simulations indicated that CTP performs well in small networks, such that its performance is much better than that of randomized gossip and comparable to that of broadcast gossip for most node initializations. However, once the network becomes larger (i.e., has more nodes), CTP has a much worse performance in terms of the number of messages transmitted compared to broadcast gossip, yet is still better than randomized gossip. This is true for cases where links fail with different probabilities as well.

Chapter 7

Conclusion & Future Work

In this thesis, we compared a tree-based protocol with two different gossip algorithms for data aggregation in a WSN. The Collection Tree Protocol (CTP) [16] was chosen as the representative tree-based protocol, because it is used in current WSN deployments and is implemented in the TinyOS operating system. CTP is used for routing packets and not for data aggregation so we implemented several modifications and additions to the original protocol to be able to perform data aggregation. The first gossip algorithm implemented was asynchronous, pairwise, randomized gossip presented in [7] by Boyd et al. and the second was broadcast gossip presented by Aysal et al. in [14]. All algorithms were implemented with lossy links following a modified version of the empirical model presented in [3], which itself is based on the study and model by Woo et al. in [46].

The goal of the comparison was to investigate which of the three aforementioned algorithms uses the least number of messages before termination. The number of messages accounted for were all the messages transmitted from the time the nodes are deployed until the algorithm attains its stopping criteria. Each algorithm had a different termination condition. For CTP, all nodes need to receive the aggregate value before the algorithm is

considered done. For randomized gossip, the error in the average of all nodes has to be below a pre-specified error value. Finally, for broadcast gossip, the per-node variance has to be below a pre-specified value.

Simulations revealed that CTP performs better than randomized gossip for different network sizes and different initializations. We also compared CTP with lossy links and randomized gossip with ideal links, but CTP had the upper hand here too. Even when links fail in the netwok with different probabilities during the aggregation process, CTP still requires a smaller number of messages to be transmitted to terminate. When CTP was compared to broadcast gossip, the results were more interesting. For small network sizes such as 50 or 75 nodes, the performance of both algorithms was similar, with broadcast gossip performing slightly better in all initialization fields except when a slope field was used. However, once the network size increased to 100, 200 and even 500 nodes, broadcast gossip was clearly superior and required a much smaller number of messages to terminate. When link failures occurred with different probabilities, again the results needed further scrutiny. Depending on the initialization field used for broadcast gossip, CTP can perform better or worse at different probabilities of link failure. For a spike initialization, broadcast gossip is always better. For Gaussian bumps, CTP and broadcast gossip perform similarly for small network sizes, but CTP performs worse as the network size increases. For a slope initialization, CTP is better when the network contains 50 nodes, but broadcast gossip is better as the network size increases to reach 100 or 200 nodes.

The results are important for practical deployments of WSN's. The network designers can decide which algorithm is better to implement given the resources and the application at hand. For example, broadcast gossip is better than CTP for large networks, but the consensus average can be different from the true average [14, 15], which may or may not be acceptable.

The work in this thesis can be expanded in several directions. The first issue is the implementation of a more efficient CTP data aggregation protocol. One of our assumptions was that links are symmetric, and even though it is one that is used in many places in the literature, in practice links are asymmetric, which can impact the number of messages transmitted. Another point here is introducing different mechanisms to deal with routing loops or dropped packets, that have aggregation in mind rather than packet routing.

An interesting question is what happens if we compare CTP to different gossip algorithms, such as greedy gossip with eavesdropping [15]. This algorithm requires a neighbor discovery mechanism to be implemented and can give more insight on how CTP performs compared to a pairwise gossip algorithm that is more efficient than randomized gossip.

It is important to realize that several applications are now depending on the use of WSN's, and the algorithms implemented in those applications play a significant role on pro-longing the life of the network. Therefore, it is crucial to come up with new algorithms or modify older ones to limit wasteful transmissions.

References

[1] J. Yick, B. Mukherjee, and D. Ghosal, "Wireless sensor network survey," *Comp. Networks*, vol. 52, pp. 2292–2330, Aug. 2008.

[2] J. Wu, *Reliable Routing Protocols for Dynamic Wireless Ad Hoc and Sensor Networks*. PhD thesis, University of Twente, Dec. 2007.

[3] W. M. Everse, "Modelling and verification of a shortest path tree protocol for wireless sensor networks : Towards a platform for formal verication experiments," Master's thesis, University of Twente, Jul. 2009.

[4] D. K. Tesfaye, "Gossip vs. tree-based monitoring under different networking conditions," Master's thesis, KTH Royal Institute of Technology, Feb. 2010.

[5] S. Keshav, "Efficient and decentralized computation of approximate global state," *SIGCOMM Comput. Commun. Rev.*, vol. 36, pp. 69–74, Jan. 2006.

[6] R. Bakhshi, "Formal analysis of tree-based aggregation protocols," Master's thesis, KTH Royal Institute of Technology, Aug. 2006.

[7] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Randomized gossip algorithms.," *IEEE Trans. Info. Theory*, vol. 52, pp. 2508–2530, Jun. 2006.

[8] F. Wuhib, M. Dam, R. Stadler, and A. Clem, "Robust monitoring of network-wide aggregates through gossiping," *IEEE Trans. On Network and Service Management*, vol. 6, pp. 95–109, Jun. 2009.

[9] R. Snader, A. Harris, and R. Kravets, "Tethys: A distributed algorithm for intelligent aggregation in sensor networks," in *Proc. IEEE Wireless Comm and Netw. Conf WCNC07*, (Hong Kong), pp. 4117–4122, 2007.

[10] A. Harris, R. Kravets, and I. Gupta, "Building trees based on aggregation efficiency in sensor networks," *Ad Hoc Networks*, vol. 5, pp. 1317–1328, Nov. 2007.

[11] L. Xiao and S. Boyd, "Fast linear iterations for distributed averaging," *Systems Control Letters*, vol. 53, no. 1, pp. 65–78, 2004.

[12] A. Dimakis, A. Sarwate, and M. Wainwright, "Geographic gossip: Efficient averaging for sensor networks," *IEEE Trans. Signal Processing*, vol. 56, pp. 1205–1216, Mar. 2008.

[13] F. Benezit, A. Dimakis, P. Thiran, and M. Vetterli, "Gossip along the way: Order-optimal consensus through randomized path averaging," in *Proc. Allerton Conf. on Comm., Control, and Comp.*, (Urbana-Champaign, IL), Sep. 2007.

[14] T. Aysal, E. Yildiz, A. Sarwate, and A. Scaglione, "Broadcast gossip algorithms for consensus," *IEEE Trans. Signal Processing*, vol. 57, pp. 2748–2761, Jul. 2009.

[15] D. Üstebay, B. Oreshkin, M. Coates, and M. Rabbat, "Greedy gossip with eavesdropping," *IEEE Trans. Signal Processing*, vol. 58, pp. 3765–3776, Jul. 2010.

[16] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis, "Collection tree protocol," in *Proc. of the 7th ACM Conf. on Embedded Networked Sensor Systems (SenSys)*, (Berkeley, CA), Nov. 2009.

[17] A. Mainwaring, J. Polastre, R. Szewczy, D. Culler, and J. Anderson, "Wireless sensor networks for habitat monitoring," in *Wireless Sensor Network and Applications WSNA02*, (Atlanta, GA), Sep. 2002.

[18] K. Lorincz and M. Welsh, "Motetrack: A robust, decentralized approach to rf-based location tracking," in *Proc. of the Int'l Workshop on Location- and Context-Awareness (LoCA 2005)*, (Oberpfaffenhofen, Germany), May 2005.

[19] I. Solis and K. Obraczka, "The impact of timing in data aggregation for sensor networks," in *Proc. IEEE Int'l Conf. on Communications*, (Paris, France), Jun. 2004.

[20] E. Fasolo, M. Rossi, J. Widmer, and M. Zorzi, "In-network aggregation techniques for wireless sensor networks: a survey," *IEEE Wireless Comm.*, vol. 14, no. 2, pp. 70–87, 2007.

[21] K. Akkaya, M. Demirbas, and R. Aygun, "The impact of data aggregation on the performance of wireless sensor networks," *Wireless Comm. and Mobile Comp. Journal*, vol. 8, pp. 171–193, 2004.

[22] M. Ding, X. Cheng, and G. Xue, "Aggregation tree construction in sensor networks," in *Proc. of the IEEE 58th Vehicular Technology Conf. VTC*, (Orlando, FL), pp. 2168–2172, Oct. 2003.

[23] S. Madden, Michael, M. J. Franklin, J. Hellerstein, and W. Hong, "Tag: a tiny aggregation service for ad-hoc sensor networks," in *Proc. of the 5th Symp. on Operating Systems Design and implementation OSDI02*, (Boston, MA), Dec. 2002.

[24] M. Dam and R. Stadler, "A generic protocol for network state aggregation," in *In Proc. Radiovetenskap och Kommunikation*, (Linkping, Sweden), pp. 14–16, Jun. 2005.

[25] F. Wuhib and R. Stadler, "M-gap: a new pattern for cfengine and other distributed software," tech. rep., Royal Institute of Technology, KTH, Jan. 2008.

[26] S. K. Verma, *Techniques for Improving Predictability and Message Efficiency of Gossip Protocols*. PhD thesis, National University of Singapore, 2009.

[27] F. Wuhib, M. Dam, and R. Stadler, "Gossiping for threshold detection," in *IFIP/IEEE Int'l Symp. on Integrated Network Management*, (Long Island, NY), Jun. 2009.

[28] T. C. Aysal, M. Coates, and M. Rabbat, "Distributed average consensus using probabilistic quantization," *Statistical Signal Processing SSP '07. IEEE*, pp. 640–644, Aug. 2007.

[29] M. Rabbat, "On spatial gossip algorithms for average consensus," in *Proc. of the IEEE/SP 14th Workshop on Statistical Signal Processing SSP*, (Madison, WI), 2007.

[30] M. Jelasity, A. Montresor, and O. Babaoglu, "Gossip-based aggregation in large dynamic networks," *ACM Trans. on Comp. Systems*, vol. 23, pp. 1317–1328, Aug. 2005.

[31] D. Kempe, A. Dobra, and J. Gehrke, "Gossip-based computation of aggregates," in *Proc. of the 44th Annual IEEE Symp. on Foundations of Comp. Science*, (Ithaca, NY), 2003.

[32] J. N. Tsitsiklis, *Problems in decentralized decision making and computation*. PhD thesis, Massachusetts Institute of Technology, 1984.

[33] M. Yildiz and A. Scaglione, "Directed gossiping for distributed data aggregation," in *Proc. IEEE CAMSAP*, (Aruba, Dutch Antilles), Dec. 2009.

[34] M. Penrose, *Random Geometric Graphs*. Oxford University Press, 2003.

[35] J. Chen and G. Pandurangan, "Optimal gossip-based aggregate computation," in *Proc. of 22nd ACM Symp. on Parallel Algorithms and Architectures, SPAA*, (Santorini, Greece), 2010.

[36] S. Borbash, A. Ephremides, and M. McGlynn, "An asynchronous neighbor discovery algorithm for wireless sensor networks," *Ad Hoc Networks*, vol. 5, pp. 998–1016, Sep. 2007.

[37] S. Vasudevan, D. Towsley, D. Goeckel, and R. Khalili, "Neighbor discovery in wireless networks and the coupon collectors problem," in *Proc. 15th Annual Intl Conf. Mobile Comp. and Networking*, (Beijing, China), Sep. 2009.

[38] E. B. Hamida, G. Chelius, and E. Fleury, "Neighbor discovery analysis in wireless sensor networks," in *Proc. of ACM CoNEXT'06*, (Lisboa, Portugal), Dec. 2006.

[39] R. Pagliari and A. Scaglione, *Grid Enabled Remote Instrumentation*, ch. Implementation of Avergage Consensus Protocols For Commercial Sensor Network Platforms, pp. 81–95. Springer-Verlag, 2009.

[40] S. Boyd, P. Diaconis, and L. Xiao, "Fastest mixing markov chain on a graph," *SIAM Rev.*, vol. 46, no. 4, pp. 667–689, 2003.

[41] D. Ganesan, B. Krishnamachari, A. Woo, D. Culler, D. Estrin, and S. Wicker, "Complex behavior at scale: An experimental study of low-power wireless sensor networks," tech. rep., UCLA/CSD-TR02-0013, Feb. 2002.

[42] A. Cerpa, N. Busek, and D. Estrin, "Scale: A tool for simple connectivity assessment in lossy environments," tech. rep., CENS, Univ. of California, Sep. 2003.

[43] A. Cerpa, J. Wong, L. Kuang, M. Potkonjak, and D. Estrin, "Statistical model of lossy links in wireless sensor networks," tech. rep., CENS, Univ. of California, Apr. 2004.

[44] S. Wahba, K. LaForce, J. Fisher, and J. Hallstrom, "An empirical evaluation of embedded link quality," in *International Conference on Sensor Technologies and Applications*, (Valencia, Spain), pp. 430–435, Oct 2007.

[45] A. Cerpa, J. Wong, M. Potkonjak, and D. Estrin, "Temporal properties of low power wireless links: Modeling and implications on multi-hop routing," in *Proc. of the 6th ACM international symp. on Mobile ad hoc netw. and comp.*, (Urbana-Champaign, IL), May 2005.

[46] A. Woo, T. Tong, and D. Culler, "Taming the underlying challenges of multihop routing in sensor networks," in *Proc. of the 1st ACM Conf. on Embedded Networked Sensor Systems*, (Los Angeles, CA), Nov. 2003.

[47] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler, *In Ambient Intelligence*, ch. TinyOS: An operating system for sensor networks, pp. 116–147. Springer Verlag, 2004.

[48] U. Colesanti and S. Santini, "A performance evaluation of the collection tree protocol based on its implementation for the castalia wireless sensor networks simulator," tech. rep., ETH Zurich, Aug. 2010.

[49] O. Gnawali, R. Fonseca, K. Jamieson, and P. Levis, "Ctp: Robust and efficient collection through control and data plane integration," tech. rep. sing-08-02, Stanford Univ., 2008.

[50] P. Levis, N. Patel, D. Culler, and S. Shenker, "Trickle: A self-regulating algorithm for code maintenance and propagation in wireless sensor networks," in *Proc. of the USENIX NSDI Conf.*, (San Francisco, CA), Mar. 2004.

[51] R. Fonseca, O. Gnawali, K. Jamieson, S. Kim, P. Levis, and A. Woo, "Tep123: The collection tree protocol (CTP)," Aug. 2006.

[52] D. S. J. D. Couto, *High-Throughput Routing for Multi-Hop Wireless Networks*. PhD thesis, Massachusetts Institute of Technology, Jun. 2004.

[53] J. Hill and D. Culler, "A wireless-embedded architecture for system level optimization," tech. rep., UC Berkeley, 2002.

[54] P. Gupta and P. Kumar, "The capacity of wireless networks," *IEEE Trans. Info. Theory*, vol. 46, pp. 388–404, Mar. 2000.

[55] A. Dimakis, S. Kar, J. Moura, M. Rabbat, and A. Scaglione, "Gossip algorithms for distributed signal processing," *Proc. of the IEEE*, vol. 98, pp. 1847–1864, Nov. 2010.

[56] D. Bertsekas and J. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*. Athena Scientific, 1997.

[57] H. Stark and J. Woods, *Probability and Random Processes with applications to Signal Processing*. Prentice-Hall, third ed., 2002.