AN ABSTRACT OF THE THESIS OF

Tingzhi Li for the degree of Master of Science in Electrical and Computer Engineering presented on December 8, 2016.

Title: Energy-Aware Gossip Techniques for Wireless Broadcasting

Abstract approved: _____

Bechir Hamdaoui

The current state of research on gossip techniques for wireless broadcasting is very limited because past research efforts have mostly focused on using gossip techniques for multicast communication. On the other hand, those research efforts that have focused on using gossip techniques for wireless broadcast communications ignore energy efficiency and network lifetime. With the emergence of Internet of Things (IoT) devices, known with their limited energy and processing resource capabilities, energy consumption is becoming more and more important to account for when designing wireless broadcasting protocols. In this thesis, we propose a new energy-aware broadcasting protocol for wireless ad-hoc networks. Specifically, the proposed protocol dynamically adapts the fanout parameter based on wireless nodes' remaining energy to prolong the lifetime of the network. Our simulation results show that our proposed energy-aware gossip protocol outperforms existing approaches by achieving fast message broadcasting times while extending the nodes' battery lifetime.

Energy-Aware Gossip Techniques for Wireless Broadcasting

by

Tingzhi Li

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented December 8, 2016
Commencement June 2017

Master of Science thesis of Tingzhi Li presented on December 8, 2016.

APPROVED:

Major Professor, representing Electrical and Computer Engineering

---

Director of the School of Electrical Engineering and Computer Science

---

Dean of the Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Tingzhi Li, Author

ACKNOWLEDGEMENTS

I would first like to thank my advisor Dr. Bechir Hamdaoui of the School of Electrical Engineering and Computer Science at Oregon State University. The door to Professor Hamdaoui office was always open whenever I had a question about my research or writing. I would like to express my gratitude to him for the useful remarks and engagement through the researching and writing process of this master thesis.

I would also like to thank Sherif Abelwahab for introducing me to the topic, providing great suggestions and answering my questions in many discussions. This research started as a team project in Advanced Computer Network class. Here, I would also like to acknowledge Marco Falke and Jinming Mu as initial project team members who participated in this project and in many ways shaped my research today.

Finally, I must express my very profound gratitude to my amazing parents Min Li, and Suling Han for their unfailing support and encouragement throughout my years of study abroad and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Moreover, I would like to thank my girlfriend Kendall Bailey for her unwavering support both during graduate school and my life.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1: Introduction

Along with the development of Information Technology, the price of the broadband connectivity continues to decrease. Devices are trending to be smaller and more powerful. People start to explore ways to connect devices to the network for better control and monitor the their status. Devices such as smart phones, smart watches, smart thermostats, and radio-frequency identification (RFID) tags are able to connect to networks and communicate to each other. If these devices are connected to the Internet as well, we call them the *Internet of Things* devices. There is no doubt that IoT is an innovative paradigm [1] because this idea combines the Internet with our everyday gadgets. Either from the perspective of private users or from the perspective of business users, there are infinite possible ways to utilize IoT [1]. As of today, research in the area of IoT is emerging rapidly and many open questions remain to be answered.

Many application services that IoT devices can provide rely on a network broadcast protocol to disseminate information [12]. For example, IoT devices' firmware update package can be broadcasted among devices in a distributed manner. However, the main challenge is that IoT devices are often resource limited meaning they have limited bandwidth and energy, and are restricted by their mobility. Due to the mobile characteristics of IoT devices, topology of physical networks formed from IoT devices is often dynamic. Therefore, a scalable, robust, and fault-tolerant broadcast protocol is needed for these dynamic networks. Flooding is consider to be the simplest broadcast protocol. However, flooding is unsuitable because of excessive overhead, media contention, and packet collision [23] which would severely deplete devices' precious battery power. Gossip techniques instead offer a relatively simple, robust, fast and probabilistic approach. This technique is inspired by the form of gossip seen in social networks.

Besides gossip techniques, several deterministic approaches have been proposed which aim to reduce overhead by shifting message forward responsibility to a subset of nodes in the network [12]. However, there are two main problems for these approaches. First, if any node in the subset fails, nodes that depend on it will not be able to receive new messages[12]. Second, the energy nodes in those subset will be depleted sooner than

other nodes that are not the subset [12].

To properly define a variation of gossip technique, three main parameters need to specified. They are *Probability of Gossip*, *Fanout*, and *Message Live Time*. With gossiping, nodes in the network have to forward the message with probability $0 < p_{gossip} \leq 1$ [12]. The idea is that a message can be broadcast successfully without every nodes' participation [12]. This approach can achieve a lower overhead because only a portion of the nodes participated in gossiping. However, the right p_{gossip} can be difficult to choose because global topology information is needed. Furthermore, an optimal p_{gossip} can become sub-optimal over time [12].

In terms of energy consumption, several adaptive energy based probabilistic schemes have been proposed. Most of them focused on dynamically adjusting p_{gossip} based on energy level related parameters. Nitnaware et. al [17] proposed an adaptive gossip protocol based on node's energy level. When a node's energy level is above threshold, it will gossip the message with a fixed p_{gossip}. When a node's energy is below the threshold, it will drop the message. For a special case where a node only has one neighbor, the *Probability of Gossip* is set to be 1 regardless of its energy level [21]. This approach adjusts *Probability of Gossip* in a very coarse manner since a node would only operate in one of two states: gossip with a fixed probability, or drop incoming new messages. In [18], node's remaining energy fraction is used directly as *Probability of Gossip*. Clearly, this is more fine-tuned than [17].

However, none of these efforts focused on another key gossip technique parameter: *Fanout*. From our observation, for any given *Probability of Gossip*, higher *Fanout* setting allows nodes to contact more neighbors each round thus achieve a faster message broadcast time. But higher *Fanout* setting usually is associated with higher energy consumption. On the other hand, lower *Fanout* setting conserves nodes' battery power but takes longer to broadcast a message. Our aim in this thesis is to retain the benefit of high *Fanout* setting (fast message broadcast time), and increase the lifetime of the network. Therefore, we proposed a gossip broadcast protocol that can dynamically adjust *Fanout* parameter based on each gossip node's remaining energy level.

The rest of this thesis is organized as follows. We present related work in Chapter 2. Chapter 3 describes the classic gossip protocol, our basic push-pull gossip protocol, and our proposed energy-aware adaptive fanout extension. Chapter 4 presents the implementation of our energy-aware gossip protocol. We present performance evaluation

in Chapter 5, and conclusion and future work in Chapter 6.

Chapter 2: Related Work

Over the years, gossip techniques have proven to be the corner stone for building scalable and robust distributed computer network systems. This technique is often used to design multicast protocol [7][9] , routing protocol [8][17][18], and broadcast protocol [12][20][22]. Demers et. al [5] demonstrated the advantages of deploying gossip techniques in corporation for database maintenance in the early days. In recent years, gossip techniques have been utilized in wired networks [2] as well as in wireless networks. Many proposed schemes whether is for wired networks or is for wireless networks, all focused on optimizing protocol overhead, or energy consumption. The metrics they used to determine *Probability of Gossip* include network density [3][25], or nodes' energy level [17][18].

In the wired network domain, gossip techniques have being used for peer-to-peer networks [7]. In the wireless network domain, gossip techniques have being used for mobile ad-hoc networks [4][24], and wireless sensor networks [14][16]. There are many proposed schemes that are designed for wired networks that uses network information for adaptive gossiping such as [10][22], but those approaches are not suitable for wireless networks [12].

Some of the basic gossip techniques that are specifically designed for wireless networks includes fixed forward probability scheme [8] where each node has the probability of p_{gossip} to gossip the message to its neighbor while it has the probability of $(1 - p_{gossip})$ to not gossip the message to its neighbor. In this paper, Haas et. al [8] are designing a routing protocol in a wireless ad-hoc network, so each time when a node receives a new message, it will only pick one of its neighbors. In other words, the *Fanout* here is set to be 1. The advantage of this proposed scheme is that it is easy to implement in practice. However, due to the dynamic nature of mobile ad-hoc network or wireless sensor network, a sufficient fixed forward probability can be difficult to choose. Moreover, even an optimal forward probability may become sub-optimal over time.

To address the disadvantage of a fixed forward probability scheme, Cartigny et. al [3] proposed a new broadcast scheme for ad-hoc networks that would adjust *Probability of Gossip* based on number of neighbors a node has. The *Probability of Gossip* is calculated

by the following equation:

$$p_{gossip} = \frac{k}{n_b}$$

where k is the propagation factor and n_b is a node's degree (number of neighbors). The minimum and maximum *Probability of Gossip* can be adjusted by changing propagation factor. The authors' idea is that a node with more neighbors will have a lower probability to gossip new messages and vice versa. This scheme reduces overhead by tailoring *Probability of Gossip* for each node but a suitable k value for various ad-hoc network topologies can still be difficult to choose.

Another interesting proposed gossip broadcast scheme is called "Smart Gossip" [12]. It uses the "family classification" method to category a node's neighbors. A node's neighbor can be classified in one of three categories: parent, sibling, or child. Intuitively, the more siblings a node has, the lower the *Probability of Gossip* will be because other siblings may have transmit the new message to the child [21]. Moreover, *Probability of Gossip* is proportional to number of children a node has [21]. When a node has no child, the $p_{gossip} = 0$ because none of its neighbors are depending on the node to receive the new message. When a node has no siblings but has children, the $p_{gossip} = 1$ because its children can only reply on the node to receive the new message. The advantage of this approach is that it takes nodes dependency into account while gossiping. However, this scheme can be complicated to implement and there is no update after establishing the initial hierarchy [21].

In terms of reducing energy consumption while using gossip techniques, Nitnaware et. al [17] proposed a simply scheme by defining a *Energy Level Threshold*. When a node's energy level drops below the threshold, this node will not gossip the new messages it receives. Otherwise, it will gossip the new message with the probability of k. However, when a node only has one neighbor, it will gossip the new message with probability of 1 regardless of its energy level. In [18], the authors proposed to use the remaining energy fraction directly as the *Probability of Gossip*. So the gossip probability is defined as:

$$p_{gossip} = \frac{E_{frac}(\%)}{100}$$

where $E_{frac}(\%)$ is a node's remaining energy fraction in percentage. A node with higher remaining energy fraction will have a higher gossip probability. A more advanced

energy-aware gossip based broadcast scheme is proposed by Reina etl. al in [20]. In this paper, the authors proposed to calculate a node's *Probability of Gossip* according to the following equation:

$$p_{gossip} = \frac{E_i - E_{min}}{E_{max} - E_{min}}$$

where E_i is the node's energy level, E_{max} is the maximum energy level among its neighbors, and E_{min} is the minimum energy level among its neighbors. This approach requires nodes to insert their energy level information when requesting a message update. This scheme is similar to "Smart Gossip" in terms of collecting neighbors information instead of focusing on the information a node itself can obtain.

One thing all these paper have in common is that they all focused on adjusting *Probability of Gossip* to reduce protocol overhead or to reduce protocol energy consumption. They have not explored the possibilities of adjusting *Fanout* to achieve longer network lifetime. Some of the paper mentioned here focused on designing a routing protocol, therefore a *Fanout* setting of 1 is the common practice. A higher *Fanout* is unlikely to improve routing protocol performance and is more complicated for a protocol to maintain the routing table. Among papers that do focus on energy consumption aspect of a gossip technique based broadcast protocol, the authors mainly would propose schemes that adjust the *Probability of Gossip* instead of the *Fanout*. Therefore, we investigated the possibilities of reducing energy consumption for a gossip technique based broadcast protocol by adjusting the *Fanout* in our research.

Chapter 3: Energy-Aware Gossip Protocol

In this chapter, we will first introduce the classic gossip protocol which will serve as a base protocol for other variations of gossip protocols. Then we will explain the detail of our basic push-pull gossip protocol. And finally, we will introduce our proposed energy-aware gossip protocol which is based on the basic push-pull gossip protocol.

3.1 Classic Gossip Protocol

3.1.1 How It Works

The objective of the gossip protocol is to broadcast messages in an efficient manner by mimicking social activities such as when people spread rumors in office by gossiping among each other. The classic gossip protocol works as follows: when a node has a new message, it will send it to multiple randomly selected nodes in the network. Every node that receives the new messages then will each randomly select multiple nodes and share the message with them. After a couple rounds of gossiping, a majority of the nodes in the network will have received this new message. The number of nodes a node tries to contact in each round is defined as the *Fanout* of the gossip protocol. It is denoted as f. Each time when a node face the decision of whether sending a new message to another node or not, the probability of doing so is defined as p_{gossip}. The probability of not gossiping the new message is $(1 - p_{gossip})$. In the rest of this thesis, I will refer to the *Probability of Gossip* as p_g. Once a node receives a new message, the number of times it will contact other nodes is defined as the *Message Live Time* of the gossip protocol. *Message Live Time* is denoted as T_l.

In a wired network setting , the p_g of classic gossip protocol is set to 1 and the *Fanout* is usually set to 1 or 2. *Message Live Time* can vary depending on the applications' requirement. In a wireless ad-hoc network setting, a simple broadcasting by flooding would cause the *broadcast storm* problem [23]. Due to overlapping radio signals in a geographical area, flooding often causes excessive redundancy, serious contention, and collision.

Instead of overwhelming the network, the *Fanout* is limited to 1 or 2. However, people often tweak p_g based on local or global network information such as the total number of nodes, or the node's degree (number of neighbors). Their goal is to reduce protocol overhead by lowering p_g while still achieving decent message broadcasting coverage.

3.1.2 Key Gossip Protocol Control Parameters

Four key parameters that define the behavior of the gossip protocol in a wireless ad-hoc network are:

- *Probability of Gossip*: $p_g \quad (0 < p_g \leq 1)$
- *Fanout*: $f = 1, 2, 3, \ldots$
- *Message Live Time*: $T_l = 1, 2, 3, \ldots$
- *Gossip Interval* ΔT_g (applicable when $T_l > 1$)

When $p_g = 1$ and $f =$ node's degree, this protocol is closely resemble to the flooding broadcast scheme which is not suitable for a wireless ad-hoc network. When $p_g = 1$ and $f = 1$ or 2, this protocol is configured to be the classic gossip protocol. T_l is a parameter that is closely related to a node's memory constraint. A large T_l setting will increase the message broadcasting successful rate at the expense of a higher memory requirement and a greater protocol overhead.

3.1.3 Variations of Gossip Protocol

It is more clear when we categorize different variations of the gossip protocol into a matrix as shown in Table 3.1.

Table 3.1: Gossip Protocol Category Matrix

	Global Network Information	Local Network Information
Fixed p_g	**Quadrant I**	**Quadrant II**
Adaptive p_g	**Quadrant III**	**Quadrant IV**

The p_g can be either a fixed or an adaptive value. The basis of calculating p_g can either be local network information such as each node's degree (number of neighbors) or

global network information such as the number of nodes in the network. Therefore, we have four quadrants in this matrix.

- **Quadrant I**: fixed p_g based on global network information.
- **Quadrant II**: fixed p_g based on local network information.
- **Quadrant III**: adaptive p_g based on global network information.
- **Quadrant IV**: adaptive p_g based on local network information.

One observation that researchers mainly focused on is adjusting p_g. Very little attention has been paid to another gossip protocol parameter the *Fanout*. Approaches using fixed p_g can calculate its probability based on the network density metrics, the distance among nodes, or the network speed [21]. In this scheme, nodes forward an incoming message with a fixed p_g, and the probability of not forwarding the incoming packet is $(1 - p_g)$ [21]. The major challenge of a fixed scheme is determining the optimal p_g. Due to the dynamic nature of wireless ad-hoc networks, even an optimal initial global p_g may become sub-optimal overtime.

Approaches using adaptive p_g utilize local or global network information such as the network density or the network speed to adjust individual or global p_g. Adaptive schemes can be divided into two categories, adaptive non-counter-based schemes or adaptive counter-based schemes [21]. Adaptive density-based schemes usually utilize each node's degree metrics. In the nb-scheme, the p_g has an inverse relationship with the number of neighbors a node has [3]. If we denote the node's degree as n_b, then

$$p_g = \frac{k}{n_b} \quad \text{where } k \text{ is the propagation factor}$$

The k is manipulated to allow the maximum and minimum p_g to be adjusted [3]. The basic idea behind this approach is that for a node with higher node's degree (meaning it has more neighbors, thus indicating this area is more dense), a lower p_g will be sufficient to spread out the new message. While for a sparse area, a higher p_g is more desirable. Some papers [19][26] suggest schemes that dynamically adjust p_g based on Received Signal Strength (RSS) or euclidean distance. In [26], the authors denoted the relative distance between node i and node j by D_{ij} and the average transmission range by r. The p_g is calculated by the following equation:

$$p_g = \frac{D_{ij}}{r}$$

For a given D_{ij}, wider average transmission range will result in a lower p_g. On the other hand, for a given average transmission range, p_g will increase when the distance between node i and node j gets greater.

In counter-based schemes, nodes keep track of the number of received copies of a given broadcasted message and use it to determine its broadcasting state [21]. Similar to non-counter-density-based schemes, Lee et. al [13] uses each node's degree in conjunction with a counter. The equation used to calculate p_g is as follows:

$$p_g = \frac{p_i}{n_b} \quad \text{where } p_i \text{ is the initial Probability of Gossip}$$

The initial probability is set to be 1. If we denote the copy of messages threshold as m_{th} and the number of received copies of a given broadcasted message as m_r, then whenever $m_r \geq m_{th}$, the above equation start to take effect.

Similar to non-counter-distance-based schemes, some papers [11][15] use the distance between nodes as a metric combined with a counter to determine the broadcasting state a node should be in.

3.2 Our Basic Push-Pull Gossip Protocol

When each node in the network forwards a new broadcast message as it receives one, it is called a *push* gossip protocol. Similarly, when each node only requests for new broadcast messages from other nodes, it is called a *pull* gossip protocol. Our gossip protocol combined both mechanisms thus it is called a *push-pull* gossip protocol.

Our basic push-pull gossip protocol utilizes three packet types to communicate. They are:

- Data packet
- ACK packet
- Request packet

Data packets carry the actually payload (the broadcast message). ACK packets and

Request packets are used to control the gossip process. The ACK packet is used to acknowledge to the sender that receiver node previsouly received that message. The Request packet is used by a node to ask for the latest message from another node. There are several rules in our push-pull gossip protocol.

- Rule 1: A node can only be in one of two states – sleep state or gossip state.
- Rule 2: Periodically, a node will request a new message from one randomly selected neighbor regardless of its state.
- Rule 3: When a node receives a new broadcast message, it will enter the gossip state.
- Rule 4: When a node is in the gossip state, it will periodically randomly select $min(f, n_b)$ number of neighbors and forward its latest message to them.
- Rule 5: When a node receives an ACK packet from any of its neighbor, it will enter sleep state which mean it will stop gossiping its latest message.
- Rule 6: When a node receives a duplicate message from anther node, it will return an ACK packet.

The pseudo code of our push-pull gossip protocol is given in Figure 3.1. All nodes in the network follow the same rules described above. For the sake of discussion, we assume that $f = 1$ and there is no isolated node in the network. In the background, every node in the network will run a request process every 5 seconds regardless of its state. During the request process, it will randomly select a neighbor and request the neighbor's latest message. Initially, every node is in the sleep state. Now let's assume that a new broadcast message is generated by node 1. Then node 1 immediately switches from the sleep state to the gossip state and starts sending out the new message to one of its neighbors. This gossip process runs every 5 seconds unless the node is switched to the sleep state. When a node is switched to the sleep state, it will do nothing. When a node receives a Data packet, it will check for duplication. If it is indeed a new message, the node will store the message and switch to the gossip state. If it is not a new message, the node will send an ACK packet back to the sender. If a node receives an ACK packet, it will switch to the sleep state. Finally, if a node receives a Request packet, it will send its latest message back to the sender.

```
// Periodic request
if state == GOSSIP or state == SLEEP:
        every 5 seconds:
                find a random neighbor N
                send a Request packet to N

// Periodic gossip
if state == GOSSIP:
        every 1 second:
                find min(f, node's degree) random neighbors N<vector>
                send Data packet to N<vector>

if state == SLEEP:
        Do nothing

// Handle packets
if receive a Data packet:
        if it is a new one:
                store the message
                state <- GOSSIP
        else
                send an ACK back

if received an ACK packet:
        state <- SLEEP

if received a Request packet:
        send the latest message back
```

Figure 3.1: The pseudo code of our push-pull gossip protocol

3.3 Proposed Energy-Aware Adaptive Gossip Protocol

As stated previously, the current state of research on gossip techniques for wireless broadcasting focuses very little on energy efficiency and network lifetime. Far too much research focuses on dynamically adjusting p_g based on global or local network information (e.g. global: number of nodes, local: node's degree). Our objective is to develop a new energy-aware gossip protocol that can extend a network's lifetime while still achieving a fast and reliable broadcasting performance. We focused on manipulating *Fanout* instead of *Probability of Gossip*.

Our observations show that a higher *Fanout* setting will result in a shorter broadcasting time for a new message at the expense of higher energy consumption. A lower *Fanout* setting conserves energy but results longer broadcasting times. First, we argue that each node's battery life should be maximized in order to extend network lifetime. Since for a broadcasting protocol, any node that is disconnected from the network due to energy depletion renders a situation that broadcasting can no longer be complete. In order to maximize each node's battery life, a high constant *Fanout* setting is undesirable when a node's battery is very low. Similarly when a node's battery is very high, a low constant *Fanout* setting can hinder the message broadcasting time. Therefore, we propose that *Fanout* should be dynamically adjusted based on each node's remaining energy fraction.

Let's denote the *Remaining Energy Fraction* as E_{frac}. The function that used to calculate the *Fanout* is defined as follow:

$$f = \begin{cases} 5 & \text{if } 0.8 < E_{frac} \leq 1, \\ 4 & \text{if } 0.6 < E_{frac} \leq 0.8, \\ 3 & \text{if } 0.4 < E_{frac} \leq 0.6, \\ 2 & \text{if } 0.2 < E_{frac} \leq 0.4, \\ 1 & \text{if } 0.0 \leq E_{frac} \leq 0.2. \end{cases}$$

The function is plotted in Figure 3.2. We divided the remaining energy fraction interval into 5 equal smaller intervals. The maximum *Fanout* is 5 and the minimum *Fanout* is 1. The general idea of our fanout function is that the *Fanout* of a node will gradually step down as its battery energy is drained. We believe this new fanout function can combine the advantages of both low and high constant *Fanout*. When a node has plenty of energy remaining, it will reach out to more neighbors and facilitate

the message broadcasting process. As a node's energy gets lower, it will conserve its battery energy by contacting less neighbors thus extending network lifetime.

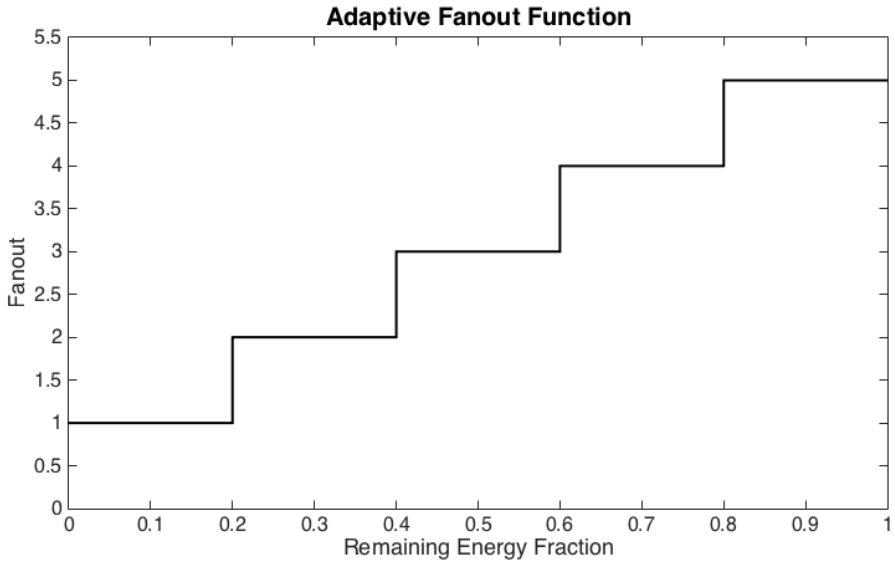

Figure 3.2: Adaptive fanout function plot

The pseudo code of our adaptive fanout push-pull gossip protocol is given in Figure 3.3. Each time a node tries to gossip a new message, it will first calculate the *Fanout* using the adaptive fanout function. One thing worth mentioning is that the *Fanout* cannot exceed its node's degree. We have to take the minimum number between the calculated the *Fanout* and the node's degree. For example, if a node only has 3 neighbors but the result from the fanout function is 5, the actual *Fanout* will be 3.

```
// Periodic request
if state == GOSSIP or state == SLEEP:
        every 5 seconds:
                find a random neighbor N
                send a Request packet to N

// Periodic gossip
if state == GOSSIP:
        every 1 second:
                calculate the fanout f based on its energy fraction
                find min(f, node's degree) random neighbors N<vector>
                send Data packet to N<vector>

if state == SLEEP:
        Do nothing

// Handle packets
if receive a Data packet:
        if it is a new one:
                store the message
                state <- GOSSIP
        else
                send an ACK back

if received an ACK packet:
        state <- SLEEP

if received a Request packet:
        send the latest message back
```

Figure 3.3: The pseudo code of our adaptive fanout push-pull gossip protocol

Chapter 4: Implementation

In order to evaluate our proposed energy-aware gossip broadcasting protocol, we implemented the protocol in an open-source software called Network Simulator 3 (NS-3). From the system point of view, this implementation consists of 4 major parts. They are:

- The ICMP extension
- The adaptive fanout push-pull gossip protocol
- The UDP server and client application
- The simulation control program

The ICMP extension is the necessary backbone gossip communication infrastructure developed to support adaptive fanout gossip protocol in the application layer. The adaptive fanout gossip protocol is the protocol entity that we are interested in studying and the pseudo code is shown in Figure 3.3. The adaptive fanout push-pull gossip protocol utilizes the underlying ICMP extension to communication among the gossip nodes. The UDP server and UDP client are installed on the source node and the gossip nodes respectively. The UDP server and client provide a channel to collect simulation data. Finally, the simulation control program is developed to handle simulation environment set up, to start and stop simulation, and to process and output collected data.

4.1 Basic Push-Pull Gossip Protocol Implementation

4.1.1 ICMP Extension

For the basic push-pull gossip protocol implementation, we first started building the 3 types of packets (Data packets, ACK packets, and Request packets) to extending the existing Internet Control Message Protocol (ICMP). An ICMP message contains two parts: an 8-byte header and a data section. The first 4 bytes of the header have a

fixed format. However, the last 4 bytes vary and depend on the type or code of the ICMP packet [6]. The first and second byte of the header is the type field and code field respectively. And the third and fourth byte are checksum field. The format of the header is shown in Table 4.1. Table 4.2 presents some of the selected ICMP message types.

Table 4.1: ICMP Header Structure

Octet	0	1	2	3
	Type	Code	Checksum	
Octet	4	5	6	7
	Rest of Header			

Table 4.2: ICMP Control Messages

Type	Code	Description
0	0	Echo reply
8	0	Echo request
9	0	Router Advertisement
10	0	Router discovery/selection/solicitation
42 to 255		Reserved

Since types 42 to 255 are reserved for further development, we decided to extend ICMP by defining type 42, 43, and 44 to represent ACK packets, Request packets, and Data packets respectively. The detail is shown in Table 4.3.

Table 4.3: Our Gossip Protocol Extension

Type	Code	Description
42	0	Send Acknowledgment
43	0	Send Request
44	0	Send Data

Based on these new control message type extensions, we could further develop our basic push-pull gossip protocol in NS-3. ICMP is an Internet layer protocol, but the actual control logic of our gossip protocol is developed in the application layer.

4.1.2 Source Node Implementation

In order to generate and collect simulation results, the system consists of a source node and n gossip nodes. The source node is responsible for the following duties:

- Generating new broadcast messages
- Storing time stamps for each generated message

Every time when a new broadcast message is generated, the source node will send the message to one of the gossip nodes via a wireless ad-hoc network channel thus starting the broadcasting process. Except the first broadcast message, the source node will only generate new message when it receives n *Acknowledgment packets* (different from the ACK packets) from all gossip nodes for the previous broadcast message. For data collection, we make sure that each gossip node will only send this special *Acknowledgment packet* once per broadcast message as it will indicate that a message has been successfully broadcasted. In order to support this feedback mechanism, we deployed an UDP server application which connects to every gossip node and is used to send the *Acknowledgment packets*. It is worth noting that the actual implementation of a source node is a restricted gossip node implementation that can only send out packets but cannot emit ACK packets or Request packets.

4.1.3 Gossip Nodes Implementation

For gossip nodes, as shown in Figure 3.1, there are two main processes. The periodic request process and periodic gossip process. At the start of the simulation, these two processes will be initialized. Most of the functionalities for a gossip node belong to either the receiving end or the transmitting end. In the receiving end, we developed functions to handle ACK packets, Request packets, and Data packets. In the transmitting end, we developed functions to send ACK packets, Request packets, and Data packets. Additionally, we also deployed a UDP client application on these gossip nodes so we can collect time stamps of each broadcast message. In order to make all these functionalities work across different layers, the functions in the gossip nodes (application layer) call the corresponding functions in ICMP (Internet layer). For example, if a gossip node is trying to send a new broadcast message to another gossip node, it would first call the

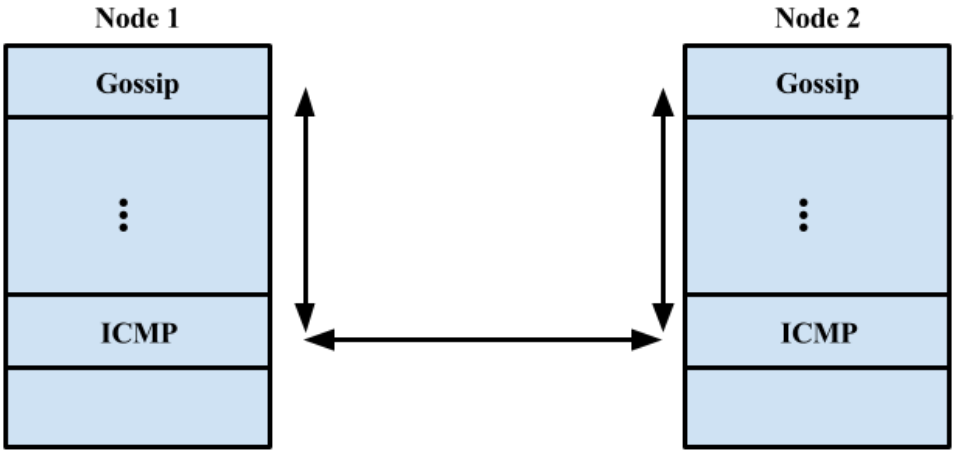

Figure 4.1: An example of how two gossip nodes communicate

function *sendPayload()* in the application layer. Then *sendPayload()* calls the function *sendMessage()* in ICMP which is in the Internet layer. On the receiving end, a node first receives the new broadcast message in the Internet layer. The message is handled by a function in ICMP called *handleData()*. In turn, this function calls the corresponding function in the application layer. The process is illustrated in Figure 4.1.

In summary, gossip nodes have the following responsibilities:

- Gossiping every new broadcast message
- Storing every broadcast message without duplication
- Storing the time stamps for each new broadcast message received
- Reporting each new broadcast message time stamps back to the source node

4.2 Adaptive Fanout Extension Implementation

In order to add our proposed adaptive fanout scheme to the existing push-pull gossip protocol, we first aggregated a basic energy source into every gossip node. The basic energy source decreases its remaining energy linearly. The initial voltage of the basic energy source is set to 3V. Then we utilized a WiFi radio energy model to simulate the

energy consumption for every gossip node when transmitting or receiving a packet. The WiFi radio energy model has 4 states defined. They are TX, RX, IDLE, and SLEEP. The power consumption of each state in Watts is defined as follow:

- $P_{tx} = 1.14W$ (transmit at 0dBm)
- $P_{rx} = 0.94W$
- $P_{idle} = 0.82W$
- $P_{sleep} = 0.10W$

Since the basic energy source is set to 3V. The currents of each state in Ampere are:

- $I_{tx} = 0.380A$
- $I_{rx} = 0.313A$
- $I_{idle} = 0.273A$
- $I_{sleep} = 0.033A$

In our implementation, we set $P_{idle} = 0$ and $P_{sleep} = 0$ because the majority of the time when a node participates in broadcasting a message, it remains in the IDLE state. Therefore, if we don't disable P_{idle} and P_{sleep}, the network lifetime will be largely determined by P_{idle} which is undesirable. Once we have energy sources and the Wifi radio energy model installed on the gossip nodes, we then calculate the corresponding *Fanout* for every node. One small detail worth mentioning here is that the actual *Fanout* f_{actual} cannot exceed a node's degree (number of neighbors) n_b, thus $f_{actual} = min(f, n_b)$. Once we calculate the *Fanout* information, the rest gossip process works as described in Section 4.1.

4.3 Simulation Control Program

As we stated earlier, the purpose of the simulation control program is to properly set up the simulation environment, to initialize simulation objects (the source node, and the gossip nodes), to start and stop simulations, and to collect, process, and export simulation data.

For the simulation environment set up, the simulation stop time is set to be large enough such that the energy source will be depleted first because we want to collect simulation data regarding the network lifetime. Any gossip node with a depleted energy source will automatically trigger the simulation to stop. Topology wise, we wanted our simulated networks to closely resemble a Wireless Sensor Network (WSN) or a Mobile Ad-hoc Network (MANET). In other words, we wanted to avoid the situation where the gossip nodes cluster in a small area. We achieve that goal by adopting a small maximum WiFi range for every gossip node and scaling up nodes' placement area as the number of the gossip nodes increases. Since a $100m \times 100m$ area with $50m$ maximum gossip node WiFi range can achieve a desirable network density for 10 gossip nodes, we used this ratio to calculate the dimension of the gossip nodes placement area. If we denote the side of a square area by s, then the equation to calculate the size of the gossip nodes placement area is as follows:

$$\frac{10}{100^2} = \frac{n}{s^2} \Longrightarrow s = \sqrt{1000 \times n}$$

where n is number of the gossip nodes. A newly generated topology can contain isolated gossip nodes that no other gossip nodes can contact because of the random placement of gossip nodes and the fixed maximum WiFi range (as illustrated in Figure 4.2). In this case, successfully broadcasting a message to all gossip nodes is not possible. Figure 4.3 shows another possible scenario where a network is divided into two separated subnets. In this case, none of the gossip nodes are isolated but we still cannot broadcast a message successfully. In order to eliminate these undesirable cases, we applied Depth First Search (DFS) algorithm to ensure that every gossip node in the network is connected in some way. Using each gossip node's neighbors list, DFS would try to traverse the entire network starting from a single node. If the algorithm is able to visit every gossip node, we consider this newly generated topology suitable for our simulation. On the other hand, when the algorithm cannot traverse every gossip node successfully, the instance of simulation will be terminated.

As mentioned earlier, DFS algorithm needs every gossip node's neighbors list in order to operate properly. In order to obtain every gossip node's neighbors list after the random gossip node placement, the program accesses every gossip node's coordinates. Let's assume that the coordinate of $node_1$ is $n_1 = (x_1, y_1)$ and $node_2$ has the coordinate

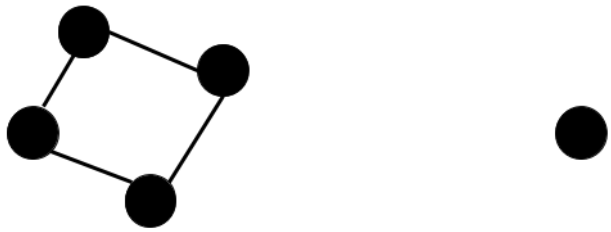

Figure 4.2: An example when there is an isolated gossip node

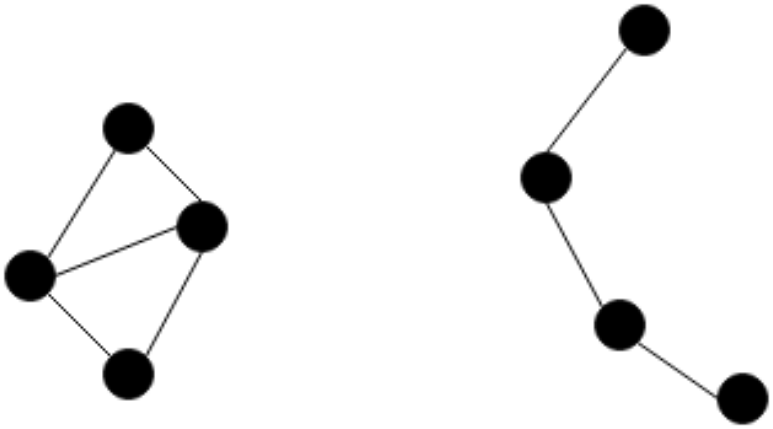

Figure 4.3: An example when two separate subnets formed

of $n_2 = (x_2, y_2)$, the distance between $node_1$ and $node_2$ can be easily computed by the distance formula as shown below:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

where d is the distance between $node_1$ and $node_2$. Then the expression to evaluate whether $node_2$ is $node_1$'s neighbor or not is shown below:

$$\begin{cases} node_2 \text{ is } node_1\text{'s neighbor} & \text{if } d \leq R, \\ node_2 \text{ is NOT } node_1\text{'s neighbor} & \text{if } d > R. \end{cases}$$

where R is every gossip node's maximum WiFi range. We use this expression repeatedly to generate neighbors list for every gossip node. In practice, the global access of each gossip node's coordinate information is usually not easy to obtain. Thus, Hello packets are used to generate neighbors lists for every gossip node.

The work flow of our simulation control program is described below:

1. Read in the simulation environment parameters.
2. Create a source node and n gossip nodes.
3. Set the wireless ad-hoc network channel speed to 1Mbps for all gossip nodes.
4. Create a special wireless ad-hoc network connection between a source node and a gossip node and setting its speed to 11Mbps.
5. Compute the side size of the square area for the gossip nodes placement.
6. Randomly placing all nodes (the source node, and the gossip nodes) in the square area
7. Install a basic energy source and WiFi radio energy model on the gossip nodes.
8. Assign IP addresses to every node.
9. Install the adaptive fanout gossip protocol and UDP client application on the gossip nodes.

10. Install the gossip generator application and UDP server application on the source node.
11. Generate a neighbors list for every gossip node.
12. Evaluate the topology connectivity using DFS algorithm.
13. If the topology is disconnected somehow, terminate the simulation.
14. If the topology is connected, start the simulation.

The link speed among gossip nodes is set to 1Mbps because it is sufficient to transmit a small Data packet (about 74 Bytes). The link speed between the source node and one of the gossip nodes is set to 11Mbps because we want to ensure that source node does not become the bottleneck of the performance of our proposed gossip protocol. Since all nodes including the source node are randomly placed in this area, we ensure that the WiFi range of the source node is large enough to reach any of the gossip nodes. The WiFi range of each gossip node is set to 50m in order to control every gossip node's degree.

As we stated earlier, the simulation will stop once any gossip node's energy source is depleted. After that, the program will enter the data collection and processing phase. In this phase, the program is designed to calculate the following data:

- Message broadcast time (one output file per simulation)
- Average overhead per node per message (average within each simulation)
- Average energy consumption per node per message (average within each simulation)
- Network lifetime

For the message broadcast time, the program will access the vector stored in the source node that represents the time stamps for each generated broadcast message. Similarly, it will also access n vectors from n gossip nodes. These vectors store the received broadcast message time stamps of each gossip node. Let's take a look at an example where we have one source node and one gossip node and the protocol successfully broadcasts m messages, if we denote the vector on the source node to be $T_s =<$

$t_{s1}, t_{s2}, \ldots, t_{sm} >$ and the vector on the gossip node to be $T_g =< t_{g1}, t_{g2}, \ldots, t_{gm} >$, then the delay for these messages is $T_{delay} = T_g - T_s =< t_{g1} - t_{s1}, t_{g2} - t_{s2}, \ldots, t_{gm} - t_{sm} >$. For the scenario where n gossip nodes participated in the broadcasting process, the program will first calculate the T_{delay} for each gossip node($T_{delay_1}, T_{delay_2}, \ldots, T_{delay_n}$). Then it will loop through the first element in those vectors and store the maximum delay because by our definition for one broadcast message the time difference between the last gossip node received the message and the time the source node generated that message is the broadcast time for that message. The program will repeat this process until it reaches the m_{th} broadcast message. However, we would like to point out that this process is only applied once per simulation. To obtain the data that our performance metrics need, further processing needs to be done.

We defined the overhead as the total number of packets the protocol sent during the simulation. This includes ACK packets, Request packets, and Data packets. Our simulation control program first will access the *packetSent* counter on each gossip node. Then it will average total number of broadcast messages(m messages). Finally, it will take the average overhead per message and average over number of gossip nodes (n gossip nodes). Again, this process is done once per simulation. Further data processing is needed to yield our desired performance metrics.

Similar to the process of calculating the average overhead per node per message, in order to compute the average energy consumption per node per message, the program first collects consumed energy from the gossip nodes, then averages it over number of gossip nodes (n). And finally it will take the average energy consumption per node and average that over the total number of broadcast messages (m).

The network lifetime is defined as the time duration over which all gossip nodes have energy to receive and transmit packets. Therefore, the program simply outputs the simulation stop time to a file.

Chapter 5: Performance Evaluation

In this chapter, we first will introduce our designed performance metrics in order to properly evaluate our proposed adaptive fanout push-pull gossip protocol. Then we will explain our simulation environment settings. Finally, we will analyze the simulation results.

5.1 Performance Metrics

As we stated in Section 3.1.1, the objective of our proposed approach is to achieve the balance between fast broadcasting time and long network lifetime. So obviously, the first two performance metrics that we proposed are *Average Network Lifetime* and *Average Message Broadcast Time*. Other performance metrics are *Average Overhead Per Node Per Message*, *Average Consumed Energy Per Node Per Message*, and *Average Number of Successfully Broadcast Messages*.

5.1.1 Average Network Lifetime

Before we define *Average Network Lifetime*, we first need to define *Network Lifetime*.

Network Lifetime: The time duration which a wireless ad-hoc network can physically broadcast messages successfully.

When a gossip node's energy is depleted, this gossip node will no longer be able to transmit or receive new broadcast messages. Thus, this network is considered to be physically unable to broadcast messages successfully. The definition of *Average Network Lifetime* is simply an average over all *Network Lifetimes* for each n gossip node case. For example, if we ran s simulations under $n = 10$ setting, and we denote *Average Network Lifetime* for 10 gossip nodes as L_{avg_10}, then the following equation can be used to calculate the *Average Network Lifetime*.

$$L_{avg\_10} = \frac{L_1 + L_2 + \ldots + L_s}{s}$$

This performance metric measures how long a wireless ad-hoc network with n gossip nodes can stay connected when running our proposed adaptive fanout gossip protocol.

5.1.2 Average Message Broadcast Time

Before defining the *Average Message Broadcast Time*, we first need to clearly define *Message Broadcast Time*.

Message Broadcast Time: The maximum delay among gossip nodes for each broadcast message.

Message Broadcast Time only measures the length of time of a single message. Therefore, we sample the maximum delay among all gossip nodes because the message received time of the last gossip node determines each message broadcast time. The *Average Message Broadcast Time* is an average over all message broadcast times for each n gossip node case. Due to the randomness of our proposed gossip protocol, each n gossip node simulation case can result in a different number of success broadcast messages. Therefore, for n gossip node case simulations, we first calculate the *Average message broadcast time* of each simulation. Then we average the average message broadcast time per simulation across each n gossip node case.

For example, let's denote the *Average Message Broadcast Time* for ith simulation as T_i and the *Average Message Broadcast Time* for n nodes as T_{avg_n}. If we performed s simulations, then the following equation can be used to calculate this metric:

$$T_{avg\_n} = \frac{T_1 + T_2 + \ldots + T_s}{s}$$

This metric indicates the time needed for a broadcast message to reach every gossip node in the network.

5.1.3 Average Overhead Per Node Per Message

The *overhead* is defined as the total number of packets sent by a gossip node. These packets include ACK packets, Request packets, and Data packets. In every simulation, in order to accurately measure the overhead for each gossip node and for each broadcast message, we simply performed an average over number of gossip nodes n. Then we take the average overhead per node and further average over the number of broadcast

messages being sent. So for kth simulation, if we denote the overhead of node i for message j as O_{ij}, the number of gossip nodes are n, and the system broadcast m messages. Then for this simulation, the *Average Overhead Per Node Per Message* can be computed using the following equation:

$$O_k = \frac{(O_{11} + O_{21} + \ldots + O_{n1}) + \ldots + (O_{1m} + O_{2m} + \ldots + O_{nm})}{n \times m}$$

If we performed s simulations under n gossip nodes setting, then the *Average Overhead Per Node Per Message* of these simulations is:

$$O_{avg\_n} = \frac{O_1 + O_2 + \ldots + O_s}{s}$$

This performance metric indicates how many packets are sent out by a gossip node to facilitate broadcasting one message.

5.1.4 Average Energy Consumption Per Node Per Message

To measure the *Average Energy Consumption Per Node Per Message*, We first measure the amount of energy consumed by each gossip node during the simulation. Then we take the average energy consumption per node and average it over m messages. E_k is the *Average Energy Consumption Per Node Per Message* for the kth simulation under n gossip nodes.

$$E_k = \frac{E_{node\_1} + E_{node\_2} + \ldots + E_{node\_n}}{n \times m}$$

Now if we ran s simulations, then *Average Energy Consumption Per Node Per Message* under n gossip nodes can be calculated using the following equation:

$$E_{avg\_n} = \frac{E_1 + E_2 + \ldots + E_s}{s}$$

This metric measures the amount of energy needed for a gossip node to facilitate broadcasting one message.

5.1.5 Average Number of Successfully Broadcast Messages

This metric measures how many messages can be successfully broadcasted with limited energy sources. If we ran s simulations for n gossip nodes, this metric can be computed using the following equation:

$$N_{avg\_n} = \frac{N_1 + N_2 + \cdots + N_s}{s}$$

where N_i represents the number of successfully broadcast messages of the ith simulation under n gossip node case. And N_{avg_n} represents the average number of successfully broadcast messages under n gossip nodes.

5.2 Simulation Environment Settings

In order to properly evaluate our proposed push-pull gossip protocol, we need to compare our protocol to push-pull gossip protocols with constant *Fanout* settings. Here we picked three *Fanout* values. They are $f = 1, 5, 10$. By obtaining performance metrics data from these three constant fanout settings, we can study how *Fanout* affect protocol performance.

The simulation environment settings are summarized below.

- Fanout: 1, 5, 10, or adaptive
- Number of nodes: 10, 50, 90, 130, 170
- WiFi speed among gossip nodes: 1Mbps
- WiFi speed between the source node and a gossip node: 11Mbps
- MAC: IEEE 802.11 for gossip nodes and source node
- RTS/CTS: On
- Gossip node maximum WiFi range: 50m
- Source node maximum WiFi range: 500m
- Simulation stop time: 100000.0s

- Initial battery energy: 108.0J (3V)
- Gossip interval: 1.0s
- Request interval 5.0s
- WiFi radio idle current: 0.0A
- WiFi radio sleep current: 0.0A
- WiFi radio transmit current: 0.380A
- WiFi radio receive current: 0.313A
- Gossip nodes IP address: 10.1.1.0/24
- Source node and a gossip node's IP address: 10.1.2.0/24

As we stated earlier, the simulation stop time is set to be large enough so that the energy sources on gossip nodes will be depleted first. Thus, any gossip node with a depleted energy source will trigger the termination of the simulations.

5.3 Result Analysis

In order to obtain simulation data for these settings, we ran 2000 simulations. Because of the random placement of gossip nodes can potentially generate a disconnected topology, some of the simulations will be terminated before starting. Table 5.1 shows the number of successful simulations for each n gossip node setting. It is worth noting that this table holds true for all four *Fanout* settings. Therefore, 1068 out of 2000 simulations are successful.

Table 5.1: Number of Simulations Run

Number of gossip nodes	Total number of simulations	Number of success simulations
10	100	85
50	100	52
90	100	50
130	100	46
170	100	34

By conducting statistical analysis based on the methods in Section 5.1, we were able to extract performance metrics of each simulation.

Figure 5.1 depicts the *Average Message Broadcast Time* over the number of gossip nodes. As we would expect, under any *Fanout* setting, the *Average Message Broadcast Time* will increase as the number of gossip nodes increase. It is especially obvious for $f = 1$ setting. For other *Fanout* settings, their *Average Message Broadcast Time* are much shorter than the setting where $f = 1$ across different number of gossip nodes settings. We can see a huge performance improvement when *Fanout* is increased from 1 to 5. This is because a gossip node can forward the broadcast message to more neighbors compare to the $f = 1$ setting. However, the performance improvement from $f = 5$ to $f = 10$ is marginal. We believe that the marginal improvement is related to the average node's degree since a *Fanout* setting that exceeds the number of neighbors of a gossip node will bring no additional performance boost. Our adaptive fanout approach performs similar to $f = 5$ setting (up to 28% longer average message broadcast time in $n = 10$ case shown in Table 5.2) even though the *Fanout* ranges from 1 to 5 during a simulation.

Table 5.2: Average Message Broadcast Time for n Gossip Nodes

n	$f = 1$	$f = 5$	$f = 10$	adaptive	adaptive vs. $f = 5$
10	16.77s	6.18s	5.53s	7.89s	27.66%
50	49.58s	18.97s	16.53s	21.86s	15.24%
90	73.38s	26.17s	25.11s	29.14s	11.36%
130	85.96s	29.46s	28.40s	31.37s	6.48%
170	94.37s	37.23s	30.97s	38.86s	3.04%

Figure 5.2 presents the *Average Network Lifetime* over different number of gossip nodes. For three constant *Fanout* settings, the *Average Network Lifetime* decreases as *Fanout* increases. This is within our expectations because for every gossip round, a gossip node with a higher *Fanout* setting will have to contact more neighbors and thus consume more energy. The $f = 1$ setting has the best *Average Network Lifetime* but as we see in Figure 5.1, it has the worst *Average Message Broadcast Time*. For the $f = 5$ setting, the *Average Network Lifetime* is reasonably good while the *Average Message Broadcast Time* is shorter. However, our proposed adaptive fanout setting can achieve even better *Average Network Lifetime* compared to $f = 5, 10$ settings while still perform on par with the other two in terms of *Average Message Broadcast Time* (up to 30%

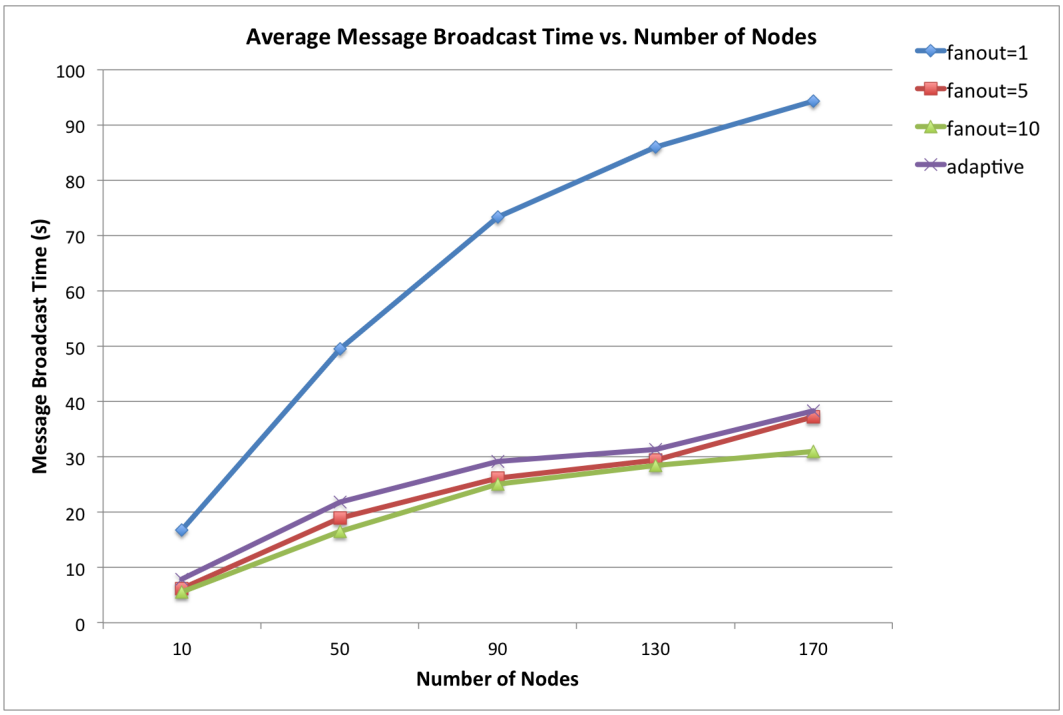

Figure 5.1: Average message broadcast time vs. number of nodes

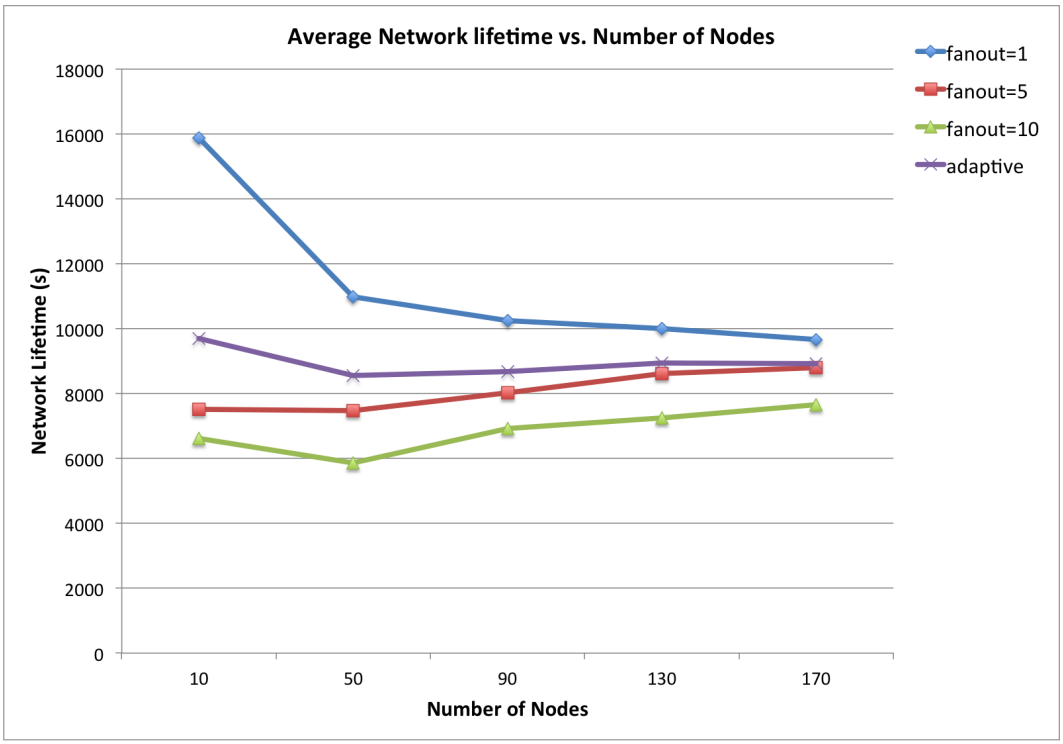

Figure 5.2: Average network lifetime vs. number of nodes

longer network lifetime when compare to $f = 5$ setting as shown in Table 5.3).

From Figure 5.3, we can see that as the number of gossip nodes increases, the *Average Energy Consumption Per Node Per Message* increases as expected. For the ease of discussion, in the following chapter we will call it *Average Energy Consumption*. The interesting part we would like to point out is that the $f = 1$ setting actually has the highest energy consumption. The reason for this is that even though it has the longest *Average Network Lifetime*, it also has the worst *Average Message Broadcast Time*. So given these two constrains, the number of messages it can broadcast is less than other *Fanout* settings. Since we computed this metric on a per message basis, it results in a higher *Average Energy Consumption*. Using the $f = 5$ and adaptive fanout setting, their *Average Energy Consumption* plots almost overlap each other. This is due to the trade off between *Average Message Broadcast Time* and *Average Network Lifetime*. The $f = 10$ setting has the lowest *Average Energy Consumption* except when $n = 170$, but

Table 5.3: Average Network Lifetime for n Gossip Nodes

n	$f = 1$	$f = 5$	$f = 10$	adaptive	adaptive vs. $f = 5$
10	15873.94s	7504.42s	6615.16s	9688.12s	29.10%
50	10983.04s	7464.01s	5860.44s	8554.85s	14.61%
90	10256.03s	8030.57s	6914.40s	8676.51s	8.04%
130	10001.87s	8620.81s	7253.99s	8946.59s	3.78%
170	9663.50s	8803.94s	7650.12s	8923.41s	1.36%

as shown in Figure 5.2, it has the worst *Average Network Lifetime*.

Figure 5.4 shows the results of *Average Overhead Per Node Per Message* over various number of gossip nodes. For the ease of discussion, from now on we will call it *Average Overhead*. Comparing Figure 5.4 and Figure 5.3, we can see that the shape of each plot is almost identical even though the units on the y-axis are different. This is because energy consumption is closely related to overhead since overhead is defined as the number of packets sent by each gossip node. Therefore, the same analysis on Figure 5.3 can be applied here as well.

Finally, Figure 5.5 presents the number of messagesbroadcast over different numbers of gossip nodes. The $f = 1$ setting can deliver the least amount of messages compared to other *Fanout* settings. In general, a higher *Fanout* setting will increase the number of messages the protocol can deliver. However, as we see in Figure 5.1, switching *Fanout* from 5 to 10 would result in a very limited performance boost. Our proposed adaptive fanout approach is similar to $f = 5, 10$ settings while having a longer *Average Network Lifetime*, as shown in Figure 5.2.

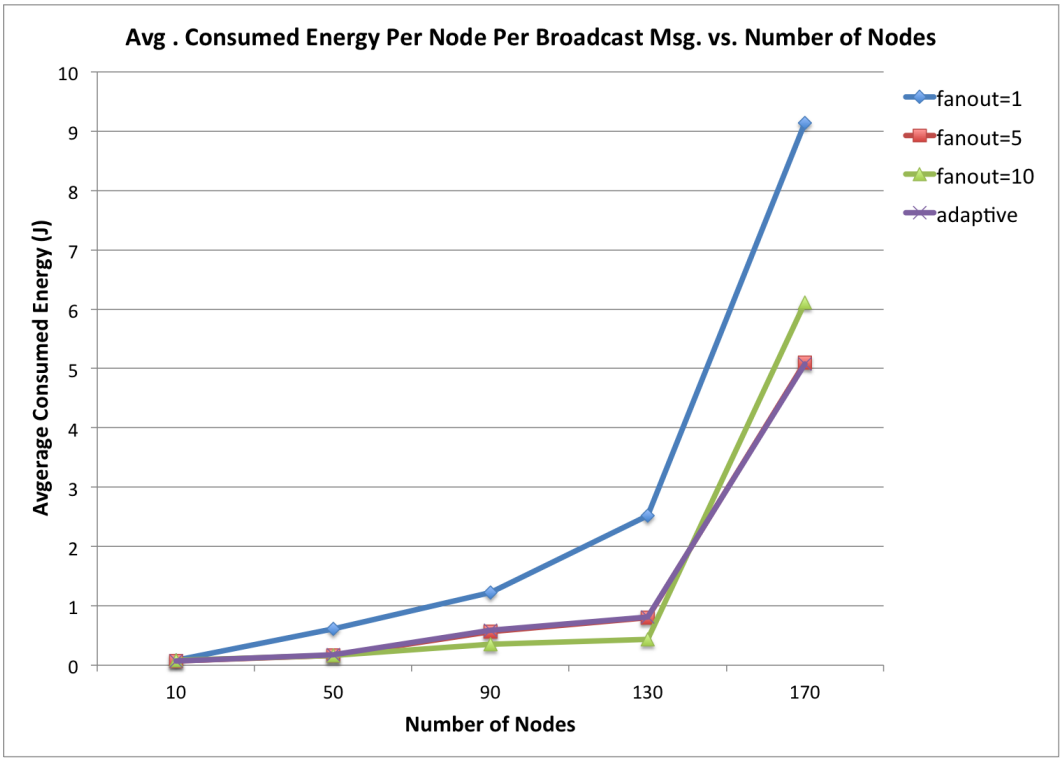

Figure 5.3: Average consumed energy per node per message vs. number of nodes

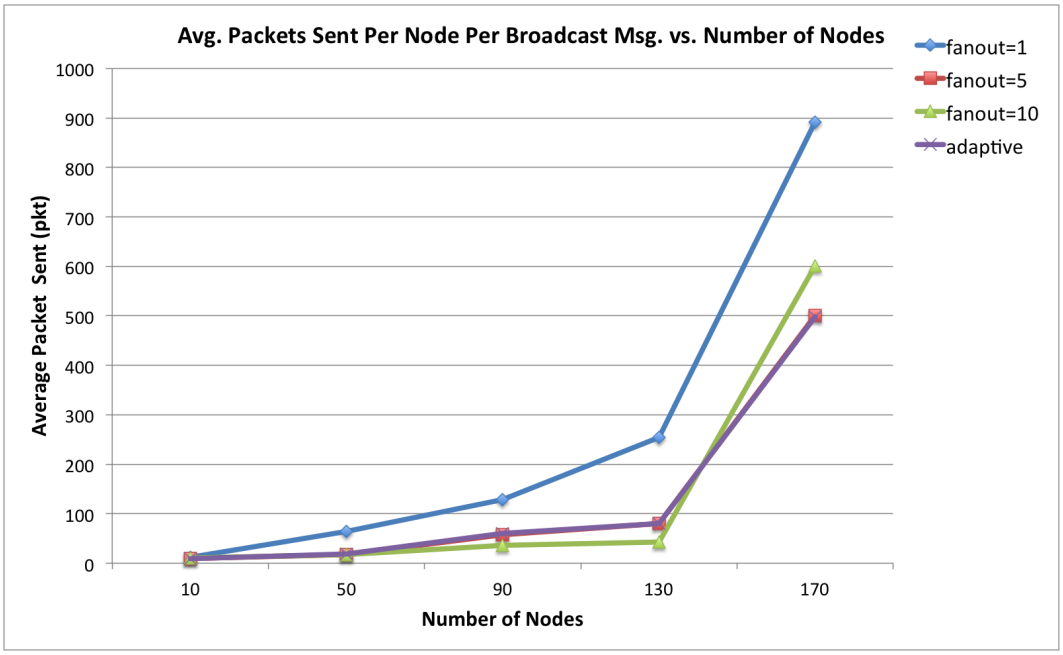

Figure 5.4: Average overhead per node per message vs. number of nodes

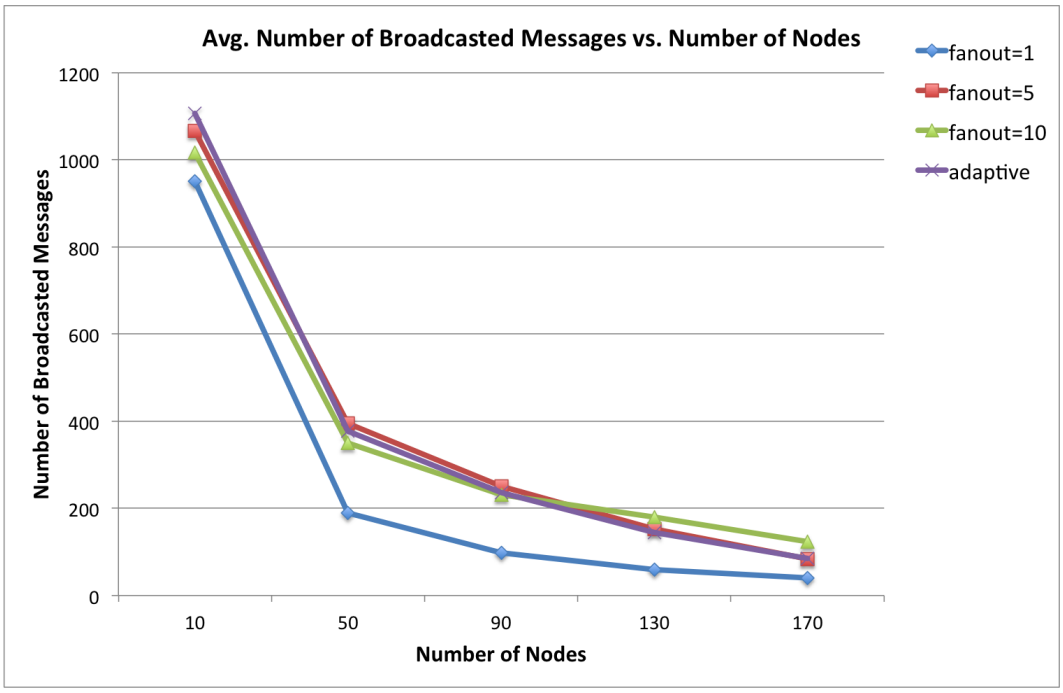

Figure 5.5: Average number of broadcast messages vs. number of nodes

Chapter 6: Conclusions and Future Work

In this thesis, we introduced research progress in gossip techniques for wireless broadcasting in recently years. We pointed out that despite all the efforts dedicated into reducing gossip broadcasting protocol overhead, very little research has focused on energy efficiency and network lifetime. Those aspects used to be not very important when designing new broadcasting protocols because nodes usually had stable power. However, with the emergence of Internet of Things (IoT) devices, broadcasting protocols that take energy consumption into account and optimize it will be favored over those that do not. Based on our observations of the trade off between battery life and broadcasting time regarding the *Fanout* parameter, we proposed a new energy-aware gossip broadcasting protocol that can balance network lifetime and broadcasting time. In order to evaluate the performance of our proposed approach, we designed several metrics and we developed the protocol in the open-source software NS-3. Simulation results showed that when compared to a constant $f = 5$ setting, our adaptive approach significantly extended network lifetime while only performing slightly slower in terms of message broadcasting time. We suspect the cause for marginal performance improvements for the $f = 10$ setting comparing to the $f = 5$ setting is due to the average node's degree. In other words, very little performance boost can be observed when the *Fanout* is set beyond the average node's degree since a node simply cannot reach out to 10 neighbors when it only has 5. As we discussed earlier, the $f = 1$ setting has the worst message broadcasting time. However, it results in a longest network lifetime which can be desirable for some applications.

In conclusion, our proposed energy-aware adaptive gossip broadcasting protocol can leverage the advantages of low *Fanout* setting and high *Fanout* setting. Thus, we can extend network lifetime while still performing close to high *Fanout* setting in terms of broadcasting time. A constant $f = 1$ setting is recommended for networks that consists of nodes with strict energy constraints.

For future work, we would like to use multicast instead of multiple unicasts for each node to send a new message. The reason is that if we assume $XJoules$ is the amount of

energy a sender uses to transmit a packet, and $f = 5$, one multicast will only consume $XJoules$ while five unicast will consume $5XJoules$. Another interesting scenario would be to set different initial energy but same battery capacity for each node. Thus each node would be operating at different *Fanout* settings at start of the simulation due to different remaining energy fraction. We believe this would better capture the advantage of our proposed adaptive fanout approach over a constant *Fanout* setting.

Bibliography

[1] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer networks*, 54(15):2787–2805, 2010.

[2] Kenneth P Birman, Mark Hayden, Oznur Ozkasap, Zhen Xiao, Mihai Budiu, and Yaron Minsky. Bimodal multicast. *ACM Transactions on Computer Systems (TOCS)*, 17(2):41–88, 1999.

[3] Julien Cartigny and David Simplot. Border node retransmission based probabilistic broadcast protocols in ad-hoc networks. In *System Sciences, 2003. Proceedings of the 36th Annual Hawaii International Conference on*, pages 10–pp. IEEE, 2003.

[4] Ranveer Chandra, Venugopalan Ramasubramanian, and Kenneth Birman. Anonymous gossip: Improving multicast reliability in mobile ad-hoc networks. In *Distributed Computing Systems, 2001. 21st International Conference on.*, pages 275–283. IEEE, 2001.

[5] Alan Demers, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry. Epidemic algorithms for replicated database maintenance. In *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, pages 1–12. ACM, 1987.

[6] A Behrouz Forouzan. *Data Communications & Networking (sie)*. Tata McGraw-Hill Education, 2006.

[7] Indranil Gupta, Anne-Marie Kermarrec, and Ayalvadi J Ganesh. Efficient epidemic-style protocols for reliable and scalable multicast. In *Reliable Distributed Systems, 2002. Proceedings. 21st IEEE Symposium on*, pages 180–189. IEEE, 2002.

[8] Zygmunt J Haas, Joseph Y Halpern, and Li Li. Gossip-based ad hoc routing. *IEEE/ACM Transactions on Networking (ToN)*, 14(3):479–491, 2006.

[9] Kate Jenkins, Ken Hopkinson, and Ken Birman. A gossip protocol for subgroup multicast. In *Distributed Computing Systems Workshop, 2001 International Conference on*, pages 25–30. IEEE, 2001.

[10] David Kempe, Jon Kleinberg, and Alan Demers. Spatial gossip and resource location protocols. *Journal of the ACM (JACM)*, 51(6):943–967, 2004.

[11] Imran Ali Khan, Akmal Javaid, and Hua Lin Qian. Distance-based dynamically adjusted probabilistic forwarding for wireless mobile ad hoc networks. In *2008 5th IFIP International Conference on Wireless and Optical Communications Networks (WOCN'08)*, pages 1–6. IEEE, 2008.

[12] Pradeep Kyasanur, Romit Roy Choudhury, and Indranil Gupta. Smart gossip: An adaptive gossip-based broadcasting service for sensor networks. In *2006 IEEE International Conference on Mobile Ad Hoc and Sensor Systems*, pages 91–100. IEEE, 2006.

[13] Ahyoung Lee and Ilkyeun Ra. Adaptive-gossiping for an energy-aware routing protocol in wireless sensor networks. In *Proceedings of the 6th International Wireless Communications and Mobile Computing Conference*, pages 1131–1135. ACM, 2010.

[14] Philip Levis, Neil Patel, David Culler, and Scott Shenker. Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks. In *Proc. of the 1st USENIX/ACM Symp. on Networked Systems Design and Implementation*, 2004.

[15] Hui Ling, Daniel Mossé, and Taieb Znati. Coverage-based probabilistic forwarding in ad hoc routing. In *Proceedings. 14th International Conference on Computer Communications and Networks, 2005. ICCCN 2005.*, pages 13–18. IEEE, 2005.

[16] Matthew J Miller, Cigdem Sengul, and Indranil Gupta. Exploring the energy-latency trade-off for broadcasts in energy-saving sensor networks. In *25th IEEE International Conference on Distributed Computing Systems (ICDCS'05)*, pages 17–26. IEEE, 2005.

[17] Dhiraj Nitnaware, Pravin Karma, and Ajay Verma. Performance analysis of energy constraint gossip based routing protocol under stochastic traffic. In *2009 Second International Conference on Emerging Trends in Engineering & Technology*, pages 1110–1114. IEEE, 2009.

[18] Dhiraj Nitnaware and Ajay Verma. Energy based gossip routing algorithm for manets. In *Recent Trends in Information, Telecommunication and Computing (ITC), 2010 International Conference on*, pages 23–27. IEEE, 2010.

[19] WANG Qing-wen, Shi Hao-shan, and Qian Qi. A dynamic probabilistic broadcasting scheme based on cross-layer design for manets. *International Journal of Modern Education and Computer Science*, 2(1):40, 2010.

[20] Daniel Reina, Princy Johnson, and Federico Barrero1and Sergio Toral. Optimization of network lifetime through energy-efficient broadcast scheme using dynamic random

walk. In *Power Electronics and Motion Control Conference (EPE/PEMC), 2012 15th International*, pages LS4e–2. IEEE, 2012.

[21] DG Reina, SL Toral, P Johnson, and F Barrero. A survey on probabilistic broadcast schemes for wireless ad hoc networks. *Ad Hoc Networks*, 25:263–292, 2015.

[22] Luis Rodrigues, Sidath Handurukande, José Pereira, Rachid Guerraoui, and A-M Kermarrec. Adaptive gossip-based broadcast. In *International Conference on Dependable Systems and Networks (DSN-2003)*. IEEE Computer Society, 2003.

[23] Yu-Chee Tseng, Sze-Yao Ni, Yuh-Shyan Chen, and Jang-Ping Sheu. The broadcast storm problem in a mobile ad hoc network. *Wireless networks*, 8(2-3):153–167, 2002.

[24] Amin Vahdat, David Becker, et al. Epidemic routing for partially connected ad hoc networks, 2000.

[25] Axel Wegener, Horst Hellbruck, Stefan Fischer, Christiane Schmidt, and Sándor Fekete. Autocast: An adaptive data dissemination protocol for traffic information systems. In *2007 IEEE 66th Vehicular Technology Conference*, pages 1947–1951. IEEE, 2007.

[26] Nawaporn Wisitpongphan, Ozan K Tonguz, Jayendra S Parikh, Priyantha Mudalige, Fan Bai, and Varsha Sadekar. Broadcast storm mitigation techniques in vehicular ad hoc networks. *IEEE Wireless Communications*, 14(6):84–94, 2007.