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 adhoc 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

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1  Introduction</td>
<td>1</td>
</tr>
<tr>
<td>2  Related Work</td>
<td>4</td>
</tr>
<tr>
<td>3  Energy-Aware Gossip Protocol</td>
<td>7</td>
</tr>
<tr>
<td>3.1 Classic Gossip Protocol</td>
<td>7</td>
</tr>
<tr>
<td>3.1.1 How It Works</td>
<td>7</td>
</tr>
<tr>
<td>3.1.2 Key Gossip Protocol Control Parameters</td>
<td>8</td>
</tr>
<tr>
<td>3.1.3 Variations of Gossip Protocol</td>
<td>8</td>
</tr>
<tr>
<td>3.2 Our Basic Push-Pull Gossip Protocol</td>
<td>10</td>
</tr>
<tr>
<td>3.3 Proposed Energy-Aware Adaptive Gossip Protocol</td>
<td>13</td>
</tr>
<tr>
<td>4  Implementation</td>
<td>15</td>
</tr>
<tr>
<td>4.1 Basic Push-Pull Gossip Protocol Implementation</td>
<td>15</td>
</tr>
<tr>
<td>4.1.1 ICMP Extension</td>
<td>15</td>
</tr>
<tr>
<td>4.1.2 Source Node Implementation</td>
<td>16</td>
</tr>
<tr>
<td>4.1.3 Gossip Nodes Implementation</td>
<td>17</td>
</tr>
<tr>
<td>4.2 Adaptive Fanout Extension Implementation</td>
<td>18</td>
</tr>
<tr>
<td>4.3 Simulation Control Program</td>
<td>19</td>
</tr>
<tr>
<td>5  Performance Evaluation</td>
<td>24</td>
</tr>
<tr>
<td>5.1 Performance Metrics</td>
<td>24</td>
</tr>
<tr>
<td>5.1.1 Average Network Lifetime</td>
<td>24</td>
</tr>
<tr>
<td>5.1.2 Average Message Broadcast Time</td>
<td>25</td>
</tr>
<tr>
<td>5.1.3 Average Overhead Per Node Per Message</td>
<td>25</td>
</tr>
<tr>
<td>5.1.4 Average Energy Consumption Per Node Per Message</td>
<td>26</td>
</tr>
<tr>
<td>5.1.5 Average Number of Broadcast Messages</td>
<td>26</td>
</tr>
<tr>
<td>5.2 Simulation Environment Settings</td>
<td>27</td>
</tr>
<tr>
<td>5.3 Result Analysis</td>
<td>28</td>
</tr>
<tr>
<td>6  Conclusions and Future Work</td>
<td>35</td>
</tr>
<tr>
<td>Bibliography</td>
<td>36</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>12</td>
</tr>
<tr>
<td>3.2</td>
<td>14</td>
</tr>
<tr>
<td>3.3</td>
<td>14</td>
</tr>
<tr>
<td>4.1</td>
<td>18</td>
</tr>
<tr>
<td>4.2</td>
<td>20</td>
</tr>
<tr>
<td>5.1</td>
<td>29</td>
</tr>
<tr>
<td>5.2</td>
<td>30</td>
</tr>
<tr>
<td>5.3</td>
<td>32</td>
</tr>
<tr>
<td>5.4</td>
<td>33</td>
</tr>
<tr>
<td>5.5</td>
<td>34</td>
</tr>
</tbody>
</table>
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Gossip Protocol Category Matrix</td>
<td>8</td>
</tr>
<tr>
<td>4.1</td>
<td>ICMP Header Structure</td>
<td>16</td>
</tr>
<tr>
<td>4.2</td>
<td>ICMP Control Messages</td>
<td>16</td>
</tr>
<tr>
<td>4.3</td>
<td>Our Gossip Protocol Extension</td>
<td>16</td>
</tr>
<tr>
<td>5.1</td>
<td>Number of Simulations Run</td>
<td>28</td>
</tr>
</tbody>
</table>
Chapter 1: Introduction

Wireless devices are becoming smaller, more diverse and powerful, and enabled more than ever with many different capabilities like sensing, computing, storage, and communication [7, 18]. New technologies (e.g., cognitive radios [21, 46, 25, 27, 10, 20, 49, 22, 41, 37, 26] and cloud computing [16, 15, 11, 12, 13, 36, 3, 14, 2]) are also emerging at a fast pace bringing with them potential solutions to many of the challenges that wireless networks, devices, and applications are facing nowadays and in the future. Billions of so-called Internet of Things (IoT) devices present in our bodies, homes, cities, factories, hospitals, and literally everywhere else are emerging as a driving force and great enablers for a new era of applications impacting nearly all aspects of our lives, from agriculture/environment monitoring, to city traffic/waste/parking management, to disaster relief (e.g., tsunamis, terrorism), to healthcare services (e.g., remote surgery, epidemics control) [1]. IoT is a unique paradigm [5] projected to change substantially how people live, think, and do business by making existing applications and services that were not possible before possible today, and by creating a myriad of new applications covering a very wide range of domains, including healthcare, manufacturing, homes, agriculture, and many others. As a result, IoT has recently been gaining tremendous momentum, attracting lots of attention, opening up new research directions and innovative methods that aim to exploit IoT in the best way possible to make the best societal, economical and environmental impacts [4].

Many application services that IoT devices can provide rely on network broadcast protocols for information dissemination [32]. For example, IoT devices can use broadcasting to enable firmware package updates in distributed manners. The main challenge here, however, is that IoT devices are often resource limited, meaning that they have limited processing, bandwidth and energy capabilities, and they are restricted by their mobility in some cases. Due to these inherent IoT characteristics, topologies of physical networks formed from IoT devices are often dynamic and change over time. Therefore, scalable, robust, and fault-tolerant broadcast protocols are needed to be able to cope with the dynamic nature of such networks. Flooding is considered to be the simplest
broadcast protocol but is not suitable as it results in excessive communication overhead, medium contention, and packet collision, etc. [47] which would severely deplete devices' batteries. To address the shortcomings of flooding, gossip techniques emerge instead to offer a relatively simple, robust, fast and broadcast approach.

Besides gossip techniques, several other deterministic approaches have been proposed with the aim to reduce protocol overhead by shifting message-forward responsibility to only a subset of nodes in the network [32]. However, there are two main problems with these approaches. First, if any node in the subset fails, nodes that depend on it will not be able to receive new messages[32]. Second, the energy resources of the nodes belonging to those subsets will be depleted faster than other nodes, leading to shorter network lifetimes [32].

Three key protocol parameters of crucial importance need be to specified: Probability of Gossip, Fanout, and Message Live Time. With gossiping, nodes in the network have to forward the messages with a probability $p_{gossip}$ [32]. The idea is that a message can still be broadcast successfully without needing every node to participate [32], thereby generating lesser overall node overhead. However, it is too difficult to choose the parameter $p_{gossip}$ due to the fact that global topology information is needed. Furthermore, optimal $p_{gossip}$ can become sub-optimal over time, and hence needs to be adaptive [32].

In terms of energy consumption, several adaptive energy-based probabilistic schemes have been proposed, most of which have focused on dynamically adjusting $p_{gossip}$ based on energy level related parameters. The basic idea of these approaches is to adapt gossip protocol based on the nodes’ available energy levels. When a node’s energy level is above threshold, the node gossips the message with a fixed $p_{gossip}$. When that energy is below the threshold, the node drops the message. For a special case where a node has only one neighbor, the Probability of Gossip is set to be 1 regardless of its energy level [44]. 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 [40], the node’s remaining energy fraction is used directly as Probability of Gossip.

However, none of these efforts focused on the Fanout gossip protocol parameter. From our observation, for any given Probability of Gossip, higher Fanout setting allows nodes to contact more neighbors each round, thus achieving 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 powers, but takes longer time to broadcast a message. Our aim in this thesis is to retain the benefit of high Fanout setting (fast message broadcast time) while maintaining increased lifetime of the network. Therefore, we propose a gossip broadcast protocol that can dynamically adjust Fanout parameter based on nodes’ remaining energy levels, thus improving message broadcast time without compromising the lifetime of networks.

The rest of this thesis is organized as follows. We present the 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 the 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. These techniques are often used to design multicast, routing and broadcast protocols [23, 29]. Demers et. al [17] 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 [6] as well as in wireless networks. Many schemes whether proposed for wired or wireless networks all focused on optimizing protocol overhead or energy consumption. The performance metrics used in these approaches include network density and available energy levels.

In wired network domains, it has been used for peer-to-peer networks [23]. In wireless network domains, it has been used for mobile ad-hoc networks [9, 48], and wireless sensor networks [34] [38]. There are many proposed schemes that are designed for wired networks that use network information for adaptive gossiping such as [30, 45], but those approaches are not suitable for wireless networks [32].

Some of the basic gossip techniques that are specifically designed for wireless networks include the fixed forward probability schemes [24], where each node with a probability of $p$ gossips the message to its neighbor and with $1 - p$ probability not to gossip the message. The authors are designing a routing protocol in a wireless ad-hoc network, where each time 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 wireless networks, a 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 [8] proposed a new broadcast scheme for ad-hoc networks that would adjust Probability of Gossip based on the number of neighbors a node has. The equation for calculating Probability of Gossip is $p_{gossip} = \frac{k}{n_b}$ 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 the propagation factor. The idea is that a node with more neighbors will have a lower probability to gossip new messages and vice versa. This scheme reduced overhead by tailoring Probability of Gossip for each node. However, suitable $k$ values for various ad-hoc network topologies still remain to be difficult to determine.

Another interesting proposed gossip broadcast scheme is called "Smart Gossip" [32]. It uses "family classification" method to categorize 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 [44]. Moreover, Probability of Gossip is proportional to the number of children a node has [44]. When a node has no children, the $p_{gossip} = 0$. When a node has no siblings but only one child, the $p_{gossip} = 1$. The advantage of this approach is that it takes nodes dependency into account while gossiping. However, this scheme can be difficult to implement and there is no update once the initial hierarchy is established [44].

In terms of reducing energy consumption while using gossip techniques, Nitnaware et al [39] proposed a simple scheme by defining and using a Energy Level Threshold. When a node’s energy level drops below the threshold, the node will not gossip new messages it receives. Otherwise, it will gossip the new message with the probability of $k$. One special case is that when a node has only one neighbor, it will gossip the new message with a probability one regardless of its energy level. In [40], the authors proposed to use the remaining energy fraction directly as Probability of Gossip. Here, the gossip probability is defined as $p_{gossip} = \frac{E_{frac}}{100}$. 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 et. al [43]. 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 for a message update. This scheme is similar to "Smart Gossip" in terms of collecting neighbors information instead of focusing on the information a node can itself obtain.

One thing all these papers have in common is that they all focused on adjusting Probability of Gossip to reduce protocol overhead or reduce protocol energy consump-
tion. They did not explore the possibility of adjusting $Fanout$ to achieve longer network lifetime. Because some of the papers that were mentioned here focused on designing a routing protocol, 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.
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 lastly, 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 gossip protocol is to broadcast messages in an efficient manner by mimicking social activities 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 sends it to multiple randomly picked nodes in the network. Every node receiving the new messages then randomly selects multiple nodes and shares the message with them. After a couple of rounds of gossiping, the majority of the nodes in the network will receive this new message. The number of nodes a node tries to contact is termed the Fanout of gossip protocol, and denoted by $f$. Each time a node faces the decision of whether it should send a new message to another node, the probability of doing so is defined as $p_{	ext{gossip}}$. In the rest of this thesis, Probability of Gossip is referred to 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 gossip protocol, and denoted by $T_l$.

In a wired network setting, the Probability of Gossip of classic gossip protocol is set to 1 whereas Fanout is usually set to 1 or 2. Message Live Time could vary depending on the application requirement. In a wireless ad-hoc network setting, a simple broadcasting by flooding would cause a broadcast storm problem [47]. Due to overlapping radio signals in a geographical area, flooding often causes excessive redundancy, serious contention, and heavy collision. Instead Fanout is set to be 1 or 2 as well. However, people often tweak Probability of Gossip based on local or global network information such as total
number of nodes and nodes’ degree (number of neighbors). Their goal is to reduce the protocol overhead by lowering Probability of Gossip while still achieving decent message broadcasting coverage.

3.1.2 Key Gossip Protocol Control Parameters

Four key parameters define the behavior of gossip protocols in wireless ad hoc networks. There parameters are:

- **Probability of Gossip**: \( p_g \)
- **Fanout**: \( f = 1, 2, 3, \ldots \)
- **Message Live Time**: \( T_l = 1, 2, 3, \ldots \)
- **Gossip Interval** \( \Delta T_g \) (applicable when \( T_l > 1 \))

When \( p_g = 1 \) and \( f = \) node’s degree, this protocol closely resembles the flooding broadcast scheme, a scheme that is not suitable for wireless ad hoc networks. When \( p_g = 1 \) and \( f = 1 \) or 2, this protocol is set to be classic gossip protocol. \( T_l \) is a parameter that is closely dependent on the node’s memory limitation. A large \( T_l \) setting increases the message broadcasting successful rate at the expense of higher memory requirement and greater protocol overhead.

3.1.3 Variations of Gossip Protocol

Table 3.1 represents and shows the different categories of gossip protocol as a matrix.

<table>
<thead>
<tr>
<th></th>
<th>Global Network Information</th>
<th>Local Network Information</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Fixed ( p_g )</strong></td>
<td>Quadrant I</td>
<td>Quadrant II</td>
</tr>
<tr>
<td><strong>Adaptive ( p_g )</strong></td>
<td>Quadrant III</td>
<td>Quadrant IV</td>
</tr>
</tbody>
</table>

The **Probability of Gossip** values can be fixed or adaptive. The basis of calculating \( p_g \) can either be local network information such as node’s degree (number of neighbors) or global network information such as 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 is that researchers mainly focused on adjusting *Probability of Gossip*. Very little attention, however, has been paid to the *Fanout* parameter. Fixed *Probability of Gossip* approaches can calculate its probability based on network density, distance among nodes, and speed [44]. In this scheme, nodes forward an incoming message with a fixed $p_g$, and the probability of not forwarding the incoming packet is $(1 - p_g)$ [44]. The major challenge with fixed schemes lies in determining the optimal $p_g$. Due to the dynamic nature of wireless ad-hoc networks, even an optimal initial global $p_g$ could become sub-optimal over time.

Adaptive *Probability of Gossip* approaches uses local or global network information such as density and speed to adjust their individual or global probability. In adaptive schemes, there are adaptive non-counter-based schemes and adaptive counter-based schemes [44]. Adaptive density-based schemes usually utilize node’s degree metrics. In (nb-scheme) the $p_g$ has an inversely proportional relationship with the number of nodes’ neighbors [8]. Denoting the node’s degree by $n_b$, we can write:

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

The $k$ is used so that the maximum and minimum probability can be adjusted [8]. The basic idea behind this approach is that for a node with higher node’s degree (meaning it has more neighbors, thus this area is more dense), a lower *Probability of Gossip* will be sufficient to spread out the new message. While for a sparse area, higher *Probability of Gossip* would be more desirable. There have been some works [42][50] that suggest schemes that dynamically adjust *Probability of Gossip* based on Received Signal Strength (RSS) or euclidean distance. In [50], the authors denoted the relative distance between node $i$ and node $j$ by $D_{ij}$ and the average transmission range by $r$, and the *Probability of Gossip* is calculated using the following equation:
For a given $D_{ij}$, wider average transmission range will result in a lower Probability of Gossip. On the other hand, for a given average transmission range, Probability of Gossip will increase when the distance between node $i$ and node $j$ gets greater.

In counter-based schemes, nodes keep track with number of received copies of a given broadcast message and use it to determine its broadcasting state [44]. Similar to non-counter-density-based schemes, some papers [33] used node’s degree in conjunction with a counter. The equation used to calculate Probability of Gossip is as follows:

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

The initial probability is set to be 1. If we denote the copy of messages threshold by $m_{th}$ and number of received copies of a given broadcast message by $m_r$, then whenever $m_r \geq m_{th}$, the above equation starts to kick in.

Similar to non-counter-distance-based schemes, some papers [31][35] used the distance between nodes as a metrics 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 new broadcast messages when 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 combines both mechanisms thus it is called a push-pull gossip protocol.

Our basic push-pull gossip protocol utilizes three packet types to perform, which are:

- Data packet
- Ack packet
- Request packet
Data packet carries the actually payload (broadcast message). Ack packet and Request packet are used to control gossip process. There are several rules in our push-pull gossip protocol. The Ack packet is used to acknowledge the sender that the receiver node already received the message. The Request packet is used for a node to request the latest message from another node.

- Rule 1: A node can only be in two states – sleep state, and gossip state.
- Rule 2: Periodically, a node will request for a new message from one randomly selected neighbor regardless of its state.
- Rule 3: When a node received a new broadcast message, it will enter the gossip state.
- Rule 4: When a node is in gossip state, it will periodically randomly select \( \min(f, n_b) \) neighbors and forward the message to them.
- Rule 5: When a node received an Ack packet from any of its neighbor, it will enter sleep state which means it will stop gossiping the new message.
- Rule 6: When a node received a duplicate message from another node, it will send an Ack packet back.

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 it to send its latest message. Initially, every node is in sleep state. Now let’s assume that a new broadcast message is generated by node 1. Then node 1 immediately switches from sleep state to gossip state and starts sending out this message to one of its neighbors. This gossip process runs every 5 seconds unless the node switches to the sleep state. When a node switches to sleep state, it will do nothing. Now when a node receives a Data packet, it will check for duplication. If it is indeed a new message, it will store the message and switch to gossip state. If it is not a new message, it will send an Ack packet back to the sender. If a node receives an Ack packet, it will switch to sleep state. Lastly, 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 we stated previously in the thesis, the current state of research on gossip techniques for wireless broadcasting focused very little on energy efficiency and network lifetime. Far too many researches focused on dynamically adjusting $p_g$ based on global or local network information (global: number of nodes, local: node's degree, overhearing). Our objective here is to develop a new energy-aware gossip protocol that could extend network lifetime while still achieving a fast and reliable broadcasting performance. The parameter that we focused on shifted from **Probability of Gossip to Fanout**.

Our observation tells us that a higher **Fanout** setting will result in a shorter broadcasting time for a new message at the expense of higher energy consumption. While a lower **Fanout** setting conserves energy, it results in a longer broadcasting time. First of all, we argue that each node’s battery life should be maximized in order to extend network lifetime. In order to maximize each node’s battery life, a constant high **Fanout** setting is undesirable when battery is very low. Similarly when battery is very high, a constant low **Fanout** setting can hinder the message broadcasting time. Therefore, we proposed 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 used to calculate **Fanout** is plotted in 3.2.

The basic idea of our fanout function is that the **Fanout** of a node will gradually decrease as its battery energy drains. We believe this new fanout function can combine the advantages of both worlds. When a node has a plenty of energy left, 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.

Now we could tweak our basic push-pull gossip protocol that we explained in Section 3.2 based on the proposed fanout function. The pseudo code of adaptive fanout push-pull gossip protocol is given in Figure 3.3. Every time when a node tries to gossip a new message, it will first calculate the **Fanout** using the adaptive fanout function. One thing that is worth mentioning here is that **Fanout** cannot exceed its node’s degree. So here we have to take the minimum number between the calculated **Fanout** and node’s degree. For example, if a node only has 3 neighbors but the result from the fanout function is 5,
Figure 3.2: Adaptive fanout function plot

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

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

the actual \textit{Fanout} will be 3.
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 a system point of view, this whole implementation consists of 4 major parts. They are the ICMP extension, the adaptive fanout push-pull gossip protocol, the UDP server and client application, and 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. It utilized underlining ICMP extension to communicate among gossip nodes. It controls our proposed adaptive fanout gossip protocol’s logic and behavior. The UDP server and UDP client are installed on source node and gossip nodes respectively. This provides a channel to collect simulation data. Lastly, the simulation control program is developed to handle simulation environment setup, start and stop simulation, and 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 those 3 types of packets (Data packet, Ack packet, and Request packet) by extending the existing Internet Control Message Protocol (ICMP). The most common use of ICMP is for error reporting [28]. An ICMP message contains two parts: 8-byte header and 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 [19]. The first and second byte of the header is the type field and code field respectively. And the third and fourth byte are the checksum field. The format of the header is shown in Table 4.1.

Table 4.2 here presented some of the selected ICMP message types.

Since type 42 to 255 are reserved for further development, we decided to extend
### Table 4.1: ICMP Header Structure

<table>
<thead>
<tr>
<th>Octet</th>
<th>Type</th>
<th>Code</th>
<th>Checksum</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Rest of Header</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Table 4.2: ICMP Control Messages

<table>
<thead>
<tr>
<th>Type</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Echo reply</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>Echo request</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>Router Advertisement</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>Router discovery/selection/solicitation</td>
</tr>
<tr>
<td>42 to 255</td>
<td></td>
<td>Reserved</td>
</tr>
</tbody>
</table>

ICMP by defining type 42, 43, and 44 to represent Ack packet, Request packet, and Data packet respectively. The detail is shown in Table 4.3.

### Table 4.3: Our Gossip Protocol Extension

<table>
<thead>
<tr>
<th>Type</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>42</td>
<td>0</td>
<td>Send Acknowledgment</td>
</tr>
<tr>
<td>43</td>
<td>0</td>
<td>Send Request</td>
</tr>
<tr>
<td>44</td>
<td>0</td>
<td>Send Data</td>
</tr>
</tbody>
</table>

Based on these new control message types extension, we could further develop our basic push-pull gossip protocol in ns-3. ICMP is a layer 3 protocol, but the actual control logic of our gossip protocol is developed in 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:

- Generate new broadcast messages
- Store time stamps for each generated new messages
Every time when a new broadcast message is generated, it will send it to one of the gossip nodes via a special pair of wireless ad-hoc network thus kicking start the broadcasting process. Except the first broadcast message, every other new broadcast message will only be generated and sent out when it received \( n \) Acknowledgment packets (different from the Ack packet) from all gossip nodes for the previous broadcast message. Because in the program we make sure that each gossip node will only send this special Acknowledgment packet once per broadcast message, it is a good indication that this broadcast message has been successfully broadcast. In order to support this feedback mechanism, we deployed an UDP server application so that every gossip node can connect to it and send its Acknowledgment packet. It is worth noting that the actual implementation of a source node is a streamlined gossip node implementation with functions in the receiving end and the ability to send Ack packets and Request packets disabled.

4.1.3 Gossip Nodes Implementation

For gossip nodes, as shown in Figure 3.1, there are two main processes. The periodic request processes and periodic gossip processes. At the start of the simulation, these two processes will be initialized. Most of the functionalities for a gossip node belong to either receiving end or transmitting end. In receiving end, we developed functions to handle Ack packet, Request packet, and Data packet. In the transmitting end, we developed functions to send Ack packet, Request packet, and Data packet. Besides, we also deployed an UDP client application on these gossip nodes so that it can send back timestamps of each broadcast message. To make all these functionalities work, these functions actually call the corresponding functions in ICMP as we described earlier. For example, if a gossip node is trying to send a new broadcast message to another gossip node, it would first call the function \( \text{sendPayload()} \) running at the application layer. Then \( \text{sendPayload()} \) would have to call the function \( \text{sendMessage()} \) in ICMP which runs at the network layer. On the receiving end, a node first receives the new broadcast message in network layer. The message is handled by a function in ICMP called \( \text{handleData()} \). Then in turn, this function will call the corresponding function in the application layer. The whole process is illustrated in Figure 4.1.

In summary, gossip nodes has the following responsibilities:

- Gossip every new broadcast message
Figure 4.1: An example of how two gossip nodes communicate

- Store every broadcast message without duplication
- Store the time stamps for each received new broadcast message
- Report each new broadcast message time stamps back to the source node

4.2 Adaptive Fanout Extension Implementation

To add our proposed adaptive fanout scheme into the existing push-pull gossip protocol, we first aggregated a basic energy source to each gossip node. Then we utilized WiFi radio energy model to simulate the energy consumption for each gossip node when transmitting or receiving a packet. The basic energy source increase or decrease its remaining energy linearly. The WiFi radio energy model has 4 states defined. They are TX, RX, IDLE, and SLEEP. The power consumption of each state in Watts are defined as follow:

- $P_{tx} = 1.14$
- $P_{rx} = 0.94$
- $P_{idle} = 0.82$
- $P_{sleep} = 0.10$
In our implementation, we actually set \( P_{idle} = 0 \) and \( P_{sleep} = 0 \) because majority of the time when a node participated in broadcasting a message, it stays in the IDLE state. Therefore, it 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 Wifi radio energy model installed on the gossip nodes, we then can calculate the corresponding Fanout for each 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 \( f_{actual} = \text{min}(f, n_b) \). Once we have the Fanout information, the rest gossip process works as described in Section 4.1.

4.3 Simulation Control Program

As we stated earlier in this chapter, the simulation control program is here to properly set up simulation environment, initialize simulation objects, start and stop simulation, and collect, process, and export simulation data.

For the simulation environment setup, because we want to collect simulation data about network lifetime, the simulation stop time is set to be large enough so that the energy resource will be depleted first. Any depleted energy resource will automatically trigger the simulation to stop. Topology-wise, we wanted it to closely resemble a Wireless Sensor Network (WSN) or MANET. In other words, we wanted to avoid having gossip nodes cluster in a small area. We achieve that goal by adopting a small maximum WiFi range for each gossip node and scale up nodes’ placement area as the number of nodes increases. Since an area with dimension of \( 100m \times 100m \) with \( 50m \) maximum WiFi range can achieve a desirable network density for 10 gossip nodes, we used this ratio to calculate the dimension of nodes placement area. If we denote the side of a square area by \( s \), then the equation to calculate the size of the nodes placement area is as follows:

\[
s = \sqrt{1000 \times n} \quad \text{where } n \text{ is number of gossip nodes}
\]

Because of random gossip nodes placement and a fixed maximum WiFi range, a newly generated topology could contain isolated nodes that no other nodes can contact. In this case, we cannot achieve successful broadcasting no matter what we do. Another possible case is shown in Figure 4.2 where a network is divided into two separated
Figure 4.2: An example when two separate subnets formed

subnets unconnected. In this case, none of the gossip nodes are isolated but we still cannot successfully broadcast a message. Therefore, we applied Depth First Search (DFS) algorithm to ensure that all nodes in the network are connected in some way. Based on each gossip node’s neighbor list, DFS would try to traverse all gossip nodes starting from any gossip node. If the algorithm is able to visit all gossip nodes, we consider this newly generated topology suitable for our simulation. On the other hand, when the algorithm cannot traverse all gossip nodes successfully, this simulation instance will be terminated.

As mentioned earlier, DFS algorithm need each gossip node’s neighbors list in order to properly perform. To obtain this information, after the random gossip node placement, the program can access each node’s coordinates. Assuming node 1 (node\textsubscript{1}) has the coordinate of (x\textsubscript{1}, y\textsubscript{1}) and node 2 (node\textsubscript{2}) has the coordinate of (x\textsubscript{2}, y\textsubscript{2}), the distance between node\textsubscript{1} and node\textsubscript{2} can be easily computed by the distance formula. Therefore, if we denote the distance between node\textsubscript{1} and node\textsubscript{2} by d, then 

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

Now if we denote each gossip node’s WiFi range by R, then node\textsubscript{1} and node\textsubscript{2} are neighbors only when d ≤ R. We used this process to generate neighbors list for each 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 list for each gossip node.

The workflow of our simulation control program is described below.
• Read in simulation environment parameters.
• Create a source node and $n$ gossip nodes.
• The wireless ad-hoc network channel speed is set to be 1Mbps for all gossip nodes.
• Create a special wireless ad-hoc network connection between a source node and a gossip node and set its speed to be 11Mbps.
• Compute the side size of a square area for nodes placement.
• Randomly place all nodes in the square area.
• Install basic energy source and WiFi radio energy model on the gossip nodes.
• Assign IP addresses to all nodes.
• Install adaptive fanout gossip protocol and UDP client application on the gossip nodes.
• Install gossip generator application and UDP server application on the source node.
• Generate neighbors list for each gossip node.
• Check topology connectivity using DFS algorithm.
• If the topology is disconnected somehow, terminate this simulation.
• If the topology is connected, start the simulation.

The link speed among gossip nodes is set to be 1Mbps because it is sufficient to transmit a very small Data packet quickly. The link speed between the source node and one of the gossip node is set to be 11Mbps because we want to make sure that source node does not become the bottleneck for the performance of our proposed gossip protocol. Since all nodes including the source node are randomly placed in this area, we make sure that the WiFi range of the source node is large enough to always be able to reach any gossip nodes. The WiFi range of gossip nodes is set to be 50m in order to control node’s degree.
As we stated earlier, the simulation will stop once any gossip node’s energy is depleted. From that point, the program will enter the data collection and process stage. At this stage, the program is trying 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 timestamps for each generated new broadcast message. Similarly, it will also access \( n \) vectors from \( n \) gossip nodes. These vectors stored the received broadcast message timestamps 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 broadcast \( m \) messages; if we denote the vector on the source node side by \( T_s = < t_{s1}, t_{s2}, \ldots, t_{sm} > \) and the vector on the gossip node side 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, because we are interested in message broadcast time, 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 generate that message is the broadcast time for that message. And then the program will repeat that process until it reaches the \( m_{th} \) broadcast message. However, we would like to point out that this process is only applied on a per simulation basis. To obtain the data that later our performance metrics need, further process needs to be done.

For the overhead, we defined it as the total number of packets this protocol sends during the simulation. This includes Ack packet, Request packet, and Data packet. Our simulation control program first will access the \( \text{packetSent} \) counter on each gossip node. Then it will average over the total number of broadcast messages (\( m \) messages). Finally,
it will take that number and averages it over the number of gossip nodes ($n$ gossip nodes). Again, this process is done in a per simulation basis. Further data process is needed to yield our desired performance metrics data.

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

The network lifetime is defined as the time during 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 will first introduce the performance metrics we used to evaluate the effectiveness of the proposed adaptive fanout push-pull gossip protocol. Then, we will describe our simulation environment settings. Finally, we will analyze and present the simulation results.

5.1 Performance Metrics

As we stated in Section 3.1.1, the objective of our proposed approach is to achieve a good balance between fast broadcasting time and long network lifetime. Therefore, the first two performance metrics that we proposed to study are Average Network Lifetime and Average Message Broadcast Time. In addition and for completeness, we also investigated the following performance metrics: Average Overhead Per Node Per Message, Average Consumed Energy Per Node Per Message, and Average Number of Success Broadcast Messages.

5.1.1 Average Network Lifetime

The Network Lifetime is this thesis is defined as the time period during which the wireless ad-hoc network remain operational in that it can broadcast messages successfully. When a gossip node’s energy is depleted, this gossip node can no longer be able to transmit or receive new broadcast messages. When this happens, the network is considered to be physically unable to broadcast messages successfully. The Average Network Lifetime is then simply defined as the Network Lifetimes of all simulated gossip nodes averaged over all the simulated scenarios.

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 we jump into the definition of Average Message Broadcast Time, we first need to clearly define Message Broadcast Time.

The Message Broadcast Time is defined as the longest time it takes any node among gossip nodes to receive the broadcast message. Because here we are trying to measure the broadcast time of a certain message, it makes sense to consider the maximum delay among all gossip nodes, since the time it takes for the last gossip node determines our message broadcast time for a particular message. Now the Average Message Broadcast Time is corresponds to the average of the Message Broadcast Times obtained under all of the simulated scenarios. Now because of the randomness of our proposed gossip protocol, each simulation run may result in a different number of success broadcast messages. Therefore, for $n$ gossip node simulations, we first calculate the Average message broadcast time for each simulation, and then compute the average among all runs.

For example, let's denote the Average Message Broadcast Time for $i$th 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 here is defined as the total number of packets sent by a gossip node. These packets include Ack packets, Request packets, and Data packets. For every simulation, in order to accurately measure the overhead for each gossip node and for each broadcast message, we first perform an average over $n$, and then take that number and average it over the number of broadcast messages that were sent. For the $k$th simulation, if we denote the overhead of node $i$ for message $j$ by $O_{ij}$, the number of gossip nodes by $n$, and the number of messages broadcasted by the system by $m$ messages, the Average Overhead Per Node Per Message can be computed as:
$$O_k = \frac{(O_{11} + O_{21} + \ldots + O_{n1}) + \ldots + (O_{1m} + O_{2m} + \ldots + O_{nm})}{n \times m}$$

If we perform $s$ simulations under $n$ gossip nodes setting, then the Average Overhead Per Node Per Message among 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 for a gossip node to facilitate broadcasting one message.

### 5.1.4 Average Energy Consumption Per Node Per Message

This performance metric is quite self-explanatory. We measure the amount of energy consumed by each gossip node during the simulation. Then we take that number and average it over the $m$ messages. Letting $E_k$ denote the Average Energy Consumption Per Node Per Message for the $k$th simulation under $n$ gossip nodes, we have:

$$E_k = \frac{E_{node,1} + E_{node,2} + \ldots + E_{node,n}}{n \times m}$$

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 Broadcast Messages

This metric measures how many messages can be successfully broadcast with limited energy sources. If we ran $s$ simulations under $n$ gossip nodes setting, this metric can be computed using the following equation:
\[ N_{avg,n} = \frac{N_1 + N_2 + \ldots + N_s}{s} \]

5.2 Simulation Environment Settings

In order to properly evaluate our proposed push-pull gossip protocol, we need to compare it to the same basic push-pull gossip protocol but with constant Fanout settings. Here we picked three typical Fanout values: \( f = 1, 5, 10 \).

The simulation environment setting is 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: 0.38A
• 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 triggering the termination of our simulations.

5.3 Result Analysis

In order to obtain simulation data for these settings, we ran 2000 simulations. Because the random placement of gossip nodes can potentially generate disconnected topologies, some of the simulations are eliminated at the beginning. Table 5.1 shows the number of successful simulations under each number of gossip nodes setting. It is worth noting that this table holds true for all four Fanout settings.

<table>
<thead>
<tr>
<th>Number of gossip nodes</th>
<th>Total number of simulations</th>
<th>Number of success simulations</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>100</td>
<td>85</td>
</tr>
<tr>
<td>50</td>
<td>100</td>
<td>52</td>
</tr>
<tr>
<td>90</td>
<td>100</td>
<td>50</td>
</tr>
<tr>
<td>130</td>
<td>100</td>
<td>46</td>
</tr>
<tr>
<td>170</td>
<td>100</td>
<td>34</td>
</tr>
</tbody>
</table>

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

Figure 5.1 depicts the Average Message Broadcast Time under various numbers of gossip nodes. As expected, under any Fanout setting, the Average Message Broadcast Time increases as the number of gossip nodes increases. It is especially obvious for \( f = 1 \) setting. Under other Fanout settings, the Average Message Broadcast Time is much shorter than the case when \( f = 1 \), and this is true regardless of the number of gossip nodes. Observe that the performance improvements are significant when Fanout is switched from 1 to 5. This is because a gossip node can forward the broadcast message to more neighbors when compared to when \( f = 1 \). However, the performance improvement
Figure 5.1: Average message broadcast time vs. number of nodes

from $f = 5$ to $f = 10$ is marginal. We believe that it is related to the average node's degree since a Fanout setting that exceeds the number of neighbors a gossip node has will bring no additional performance boost. Our adaptive fanout approach performs as good as $f = 5$ setting even though its Fanout is ranging from 1 to 5 during a simulation.

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 expectation because for every gossip round, a gossip node with a higher Fanout setting will have to contact more neighbors, thus consuming more energy. $f = 1$ setting has the best Average Network Lifetime but as we see in Figure 5.1, it has the worst Average Message Broadcast Time. Under $f = 5$ setting, the Average Network Lifetime is reasonably good while having a shorter Average Message Broadcast Time. However, our proposed adaptive fanout setting can achieve even better Average Network Lifetime when compared to $f = 5, 10$ settings while still performing as good as the other two in terms of Average Message Broadcast Time.
Figure 5.2: Average network lifetime vs. number of nodes
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 rest of this chapter, we will simply term it: *Average Energy Consumption*. The interesting part that we would like to point out is that the \( f = 1 \) setting actually has the highest energy consumption. The reason 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 that achieved under the other *Fanout* settings. Since we compute this metric on a per message basis, it results in a higher *Average Energy Consumption*. Under the \( f = 5 \) and adaptive fanout setting, the *Average Energy Consumption* plots are almost overlapping with one another. The reason behind this is the tradeoff that exists between *Average Message Broadcast Time* and *Average Network Lifetime*. The \( f = 10 \) setting has the lowest *Average Energy Consumption* except for 170 gossip nodes setting, but 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* under various numbers of gossip nodes. For the ease of discussion, from now on we will call it *Average Overhead*. Comparing Figure 5.4 to Figure 5.3, we can see that the shape of each plot is almost identical even though the unit on the y-axis is 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.

Lastly, Figure 5.5 presents the number of messages broadcast under different numbers of gossip nodes. The \( f = 1 \) setting can deliver the least amount of messages compared to the other *Fanout* settings. In general, 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 performs as good as \( f = 5, 10 \) setting while, as shown in Figure 5.2, having longer *Average Network Lifetime*. 
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 looked at the gossip techniques developed in the recent years for wireless broadcasting. We observed that despite all these efforts mostly dedicated to reducing gossip broadcasting protocol overheads, very little research has focused on energy efficiency and network lifetime. Energy may not be very crucial to consider when designing broadcasting protocols for networks whose nodes do not have power supply limitation. However, with the emergence of Internet of Things (IoT) devices, broadcasting protocols that take energy consumption into account becomes a necessity and of a great importance. Based on our observation of the tradeoff between battery life and broadcasting time regarding Fanout parameter, we proposed a new energy-aware gossip broadcasting protocol that could balance between network lifetime and broadcasting time. In order to evaluate the several performance metrics of our proposed approach, we implemented the proposed protocol in ns-3, which a well-known open-source software that is used to study the performance of networking protocols. Simulation results show that compared to constant $f = 5$ setting, our proposed adaptive approach significantly extends network lifetime while performing only slightly slower in terms of achievable message broadcasting time. We suspect the cause for marginal performance improvements for the $f = 10$ setting compared to the $f = 5$ setting is the average node’s degree. In other words, very little performance boost can be observed when Fanout is set beyond average node’s degree since a node simply cannot reach out to 10 neighbors when it only has 5 neighbors. As we discussed earlier, the $f = 1$ setting has the worst message broadcasting time. But on the flip side, it would result in the longest network lifetime.

In conclusion, our proposed energy-aware adaptive gossip broadcasting protocol can leverage the benefits of both low and high Fanout settings. And as a result, it extends the network lifetime while still performing as well as the Fanout setting case in terms of broadcasting time. Constant $f = 1$ setting can, however, be recommended for networks that consists of nodes with strict energy constraints.

For future work, we would like to use multicast instead of multiple unicast for each node to send new messages. The reason is that if we assume $XJ$ is the amount of energy
used to transmit a packet for a sender and $f = 5$, one multicast will only consume $XJ$ while five unicasts will consume $5XJ$. Another interesting scenario would be to set very different initial energy levels but consider the same battery capacity for each node. Thus each node would be operating at different Fanout setting from the beginning due to the different remaining energy fraction. We believe this would better exploit the advantage of our proposed approach over the case of a constant Fanout setting.
Bibliography


