OREGON STATE UNIVERSITY

Energy-Aware Gossip Techniques for Wireless Broadcasting

by

Tingzhi Li

A thesis submitted in partial fulfillment for the degree of Master of Science

in the

Bechir Hamdaoui
School of Electrical Engineering and Computer Science

November 2016

Declaration of Authorship

I, TINGZHI LI, declare that this thesis titled, ‘Energy-Aware Gossip Techniques for Wireless Broadcasting’ and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: ____________________________________

Date: ____________________________________

“Imagination is more important than knowledge. Knowledge is limited. Imagination encircles the world. ”

Albert Einstein

OREGON STATE UNIVERSITY

Abstract

Bechir Hamdaoui
School of Electrical Engineering and Computer Science

Master of Science

by Tingzhi Li

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.

Acknowledgements

I would like to thank Dr. Bechir Hamdaoui for his advising. We also would like to thank Sherif Abelwahab for providing great suggestion and answering to my questions in many discussions.

Our team of three members had different responsibilities in fullfilling the project's goal. All of us did reasearch in the field of IoT, Sensor Networks and distributed algorithms. Each of us wrote different sections of the paper and helped improving all other sections by proof-reading.

The two graduate students – Marco Falke and Tingzhi Li – wrote the section about related work. Marco Falke wrote about Virtual Networks and Tingzhi Li covered WiFi Direct.

The different coding parts were assigned as follows. Jinming Mu did the initial setup of ns-3 and coded the topology creationg part of the code, he also helped Marco to implement the Gossip protocol. Marco Falke had the responsibility to implement the Gossip protocol in form of a ns-3 application. Furthermore, he wrote the topology file parser to read a given topology file. Lastly, he coded the connection between the ICMP level of ns-3 and application level. Tingzhi Li did modifications to the ICMP part of ns-3 to allow transmission of modified control messages. Moreover, he wrote code to process the produced data and create plots.

Contents

List of Figures

List of Tables

Abbreviations

LAH	**L**ist **A**bbreviations **H**ere

Physical Constants

Speed of Light	c	$=$	$2.997\ 924\ 58 \times 10^{8}$ $\mathrm{ms}^{-\mathrm{S}}$ (exact)

Symbols

a	distance	m
P	power	W (Js^{-1})
ω	angular frequency	rads^{-1}

For/Dedicated to/To my…

Chapter 1

Introduction

Along with the development of the information technology, the price of the broadband connectivity becomes affordable. 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 status of them. Devices such as smartphone, smartwatch, and smart thermostats are able to connect to a network and these devices could communicate with each other. If these devices are connected to the Internet as well, we call them the *Internet of Things*. There is no doubt that IoT is an innovative paradigm [1] because this idea combined the Internet with our everyday gadgets. Either from the perspective of private user or from the perspective of business user, there are infinite possible ways to exploit IoT. As of today, research in the area of IoT is emerging rapidly and many open questions remain to be answered.

Remote sensing is one of the possible services of IoT. Remote sensing allows users to get the data from the device through the network instead of physically read the data. Remote sensing involves the search and selection of IoT devices to form a virtual sensor network. Subsequently, the selected devices estimate the wanted property collaboratively and report the result to the remote cloud agent. IoT assigns a unique MAC address to physical object which do not have an identification, so people could track the specific objects status and control it remotely. The advantage of remote control is that it requires less manpower needed to monitor and manage the object. And it could further increase devices usage rate.

However, the main challenge here is that IoT devices are often resource limited meaning they have limited bandwidth and power, and restricted by their mobility. Due to the characteristics of IoT devices, topology of physical network formed from IoT is dynamic. The distributed network protocol which virtualizing the physical network has to deal with this dynamic nature of IoT networks.Those dynamic networks require a robust way to spread information across the whole network while minimizing the burden placed on the network. An efficient protocol is the gossip protocol [2], which spreads information similar to how an infection spreads in a population.

(563) There is no doubt that the *Internet of Things (IoT)* is an innovative paradigm, [1] which is gaining popularity in our modern society. With the development of information technology, digital devices are getting smaller and yet more powerful. The basic ideal of IoT is that with "unique addressing schemes", various of *things* or *objects* such as smart phones, watches, thermostats, Radio-Frequency IDentification (RFID) tags, sensors are able to communicate, cooperate with each other to perform tasks [1].

Remote sensing is one of the promising services of IoT. With remote sensing, the users could retrieve collected data through the network instead of physically retrieving data. Remote sensing involves the search and selection of IoT devices to form a virtual sensor network. Afterwards, sensing task is sent to the virtual sensor network. The selected devices then perform sensing collaboratively and report the result back to the remote cloud agent.

There is one step during remote sensing process that I am particularly interested in, which is sending tasks to the virtual sensor network. There are two aspects of this area. One question is that what kind of network would IoT devices form? The other question is that what kind of protocol is efficient and robust for broadcasting the sensing task? As we know, IoT devices often have limited bandwidth and power. Their mobility further impacted the topology of the selected virtual sensor network. Due to the characteristics of IoT devices, ad-hoc network is often used to communicate among devices. Flooding was a simple algorithm to broadcast messages in wired network but was prove to be unsuited for wireless environment due to excessive overhead, media contention, and packets collision. Gossip technique instead is often used to quickly broadcast messages with lower overhead. Gossip technique is inspired by the form of gossip seen in social network. In a network, a node with a new message would randomly pick another node and gossip the message. The other node then would do the same thing. This is refer to as classic gossip technique. A variation of that would only randomly pick a node that is its neighbor. But regardless of how a node pick aother node or choose where to pick from, gossip technique could broadcast a new message in a timely and robust manner.

Over the years, researcher have been focusing on how to improve gossip technique's reliability while lowering overhead. Many proposed gossip schemes have been proposed. Some porposed to apply gossip probability on nodes. The idea behind that is that a message could be broadcasted successfully without every nodes' participation. In this scheme, a lower overhead could be achieved because a portion of the nodes participated in gossipping but a message is still being broadcasted. Some proposed a event counter based scheme to combat this issue. The gist of that shceme is that if a node overheard the same messages a times and $a > b$ were b is the threshold, it would not gossip its latest message this time. Some even went a step further, they tried to identify the dependency among a node and its neighbors and dynamically adjust gossip probability based on collected information.

However, none of them focused directly on how to conserve energy while gossipping. Some might argure that the effort on lowering overhead is equivlent to conserving energy but (!!!need to think about this). As we are moving to an IoT and mobile devices dominated world, energy conservation become more and more important as to overall user experience or network survival time. In this thesis, I proposed a gossip technique that dynamically adjust gossip fan-out based on each node's remaining energy. The results showed that my proposed approach performed as well as constant gossip fan-out approach while still conserving significant amount of energy.

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

two category: fixed probability schemes, and adaptive probability schemes.

fixed probability:

adaptive probability: counter-based, non-counter-based

counter-based: density, distance, energy

non-counter-based: density, speed, distance, energy

PS: smart gossip,

outliers: trickle algorithm

(from 563 paper).

2.1 Virtual Sensor Networks

The ongoing technological progress further and further improves the computation, connectivity and sensing capabilities of various devices, sometimes mobile ones. [3] This enables a huge variety of opportunities in sensor networks. For example, devices in a sensor network could be assigned tasks based on their constraints in computation, power usage or networking potential. In contrast to dedicated sensor networks, where the participating nodes serve a single application, Virtual Sensor Networks (VSN) take advantage of the nodes technological progress. When a VSN is formed on top of a Wireless Sensor Network, only a subset of all available nodes is part in the VSN. Furthermore, several VSNs can exist simultaneously in on Wireless Sensor Network. [3] That is, one subset of the nodes forms a VSN and relies on the remaining nodes to communicate between its nodes. In some cases, physical nodes of one VSN even could be completely cut off from communication due to their spatial distribution and must rely on the other nodes. Usually the different VSNs pursue completely unrelated sensing tasks and the nodes in each VSN behave like they are on their independent Sensor Network. Figure **??** based on [3] depicts a visualization of two VSNs formed on top of an Wireless Sensor Network. This logical separation helps to simplify the implementation of applications significantly. [3] Further advantages of VSNs are enhanced performance and better scalability.

The development of algorithms and protocols to support the grouping of VSNs on top of Sensor Networks, is still an ongoing research topic. Those need to consider how the available time and frequencies should be fairly distributed for intra network communication. Moreover, it should be possible for nodes to change their membership in VSNs.

2.2 Virtual Networks on Top of the Internet

It is important to realize that the Internet, due to so many different participants with sometimes opposing interests, is hard to modify and only possible small and slow steps, if at all. Therefore, Virtual Networks are often the only way to realize innovation. To implement a Virtual Network using the existing Internet, several things need to be considered. First, the characteristics of the networking technology determine the attributes of the Virtual Network. For instance, a wired network yields a more scalable and bandwidth flexible Virtual Network than a wireless network would do. [4] Second, the layer of virtualization (referring to the OSI layer model) impacts the flexibility of the Virtual Network. That is, the lower the layer of virtualization, the more flexibility will be possible. Specifically, so-called overlay networks, mostly realized in the application layer, are limited in their ability to support fundamentally different architectures. [4] Moreover, virtualization on top of IP is fixed to the network layer protocol and cannot deploy IP independent mechanisms. [4] Lastly, an important consideration in the non-comprehensive list is also about security and privacy in virtual networks. Thus, attack vectors such as denial-of-service or distributed denial-of-service against the underlying physical network will have impact on all simultaneously virtualized networks.

2.3 Virtualization Algorithm

Though, it is possible to form a VSN of mobile IoT devices by having access to all relevant data such as availability, sensor capabilities or sensor mobility, a more efficient solution is to assume the managing cloud agent does not have full knowledge of every sensors properties. [5] The cloud agent even may not be connected to all nodes but only to a subgroup of them. The presented algorithm also takes into account mobility of the devices which sometimes leads to nodes being unavailable for some time. [5] This virtualization algorithm will search and select appropriate sensors from the whole network to form the virtual network which then executes the sensing task.

The project goal is to evaluate gossip protocol performance in the wireless ad-hoc network since several papers[2][6] claimed that gossip protocol is an efficient, scalable, reliable message dissemination approach. I would like to implement gossip protocol in a wireless ad-hoc network in ns-3 and study its reliability, scalability and efficiency. Based on the progress of implementation gossip protocol in a wired peer-to-peer network in ns-3, my first step is to switch the network environment from wired peer-to-peer to wireless ad-hoc network. The wireless ad-hoc network setting are as following:

- WLAN Standard: IEEE 802.11b
- MAC layer: wifi ad-hoc mode
- Add non-QoS upper mac layer
- Modulation: DSSS
- Data Rate: 1Mbps
- RTS/CTS: On
- Receiver Gain: 0dB
- Delay Mode: Constant Speed Propagation Delay Mode
- Loss Mode: Friis Propagation Loss Mode
- Ipv4 address base: 10.1.0.0
- Ipv4 address netmask: 255.255.0.0

For the topologies that I used to evaluate gossip protocol, the number of nodes are 100, 250, 400, 550, 700, 850, and 1000. For each case, there are 100 random generated topology files defining the connectivity among those nodes. In the ad-hoc network, each node usually has multiple edges and we assume that there is no isolated node in the network. For the allocation of those nodes, I used random grip allocator in ns-3. The distance between two adjacent nodes is 5 meters. I assume that the connectivity remained regardless of the distance between two nodes. A simple example of 9 nodes random grip allocation would look like fig. **??**. Implementation of gossip protocol will be presented in section **??**.

There are three essential performance metrics I would like to measure.

- Average number of data packets sent per node
- Average hops per node needed to spread the message
- Maximum time needed until the message is spread

The average number of data packets sent per node is a key metrics that measure the efficieny of gossip protocol comparing to other popular multicast protocol like MAODV. It mostly emphysis on the efficiency or workload of sender's side. Theoretically, the average number of data packets sent per node would remain constant regradless of the scale of the network.

The average hops per node needed to spread the message is a metrics that indicates the efficiency on the receiver's side. It is a metrics that represents how many times the message is forwarded before the node received it. Generally, lower average hops per node is preferred.

The maximum time needed to spread the message could be used to evaluate the time complexity of gossip protocol. Baically, this metrics indicates how fast a message can be spred across the whole network.

In this project, randomness is shown in three different aspects: (1) network topologies are random generated. (2) The node that get the initial message is randomly chosen. (3) For each node during simulation, it randomly chooses neighbour to perform ”gossipping.”

After evaluation the performance of gossip protocol, we hope to verify the following assumptions.

- Time complexity of the gossip protocol is $O(\log N)$, where N is the number of nodes.
- Average number of data packets sent per node will remain constant regardless of network scale.

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 built 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 had a new message, it will send it to multiple randomly picked nodes in the network. Every node that received the new messages then will each randomly select multiple nodes and share the message with them. After a couple rounds of gossiping, majority of the nodes in the network have received this new message. The number of nodes a node tried to contacted is defined as the *Fanout* of 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}. In the rest of this thesis, I will refer to *Probability of Gossip* as p_g. Once a node received a new message, the number of times it will contact other nodes is defined as the *Message Live Time* of gossip protocol. It is denoted as T_l.

In a wired network setting , the *Probability of Gossip* of classic gossip protocol is set to be 1 and *Fanout* is usually set to be 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 *broadcast storm* problem [7]. Due to overlapping radio signals in a geographical area, flooding often cause excessive redundancy, serious contention, and 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, or node's degree (number of neighbors). Their goal is to reduce 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 that define the behavior of 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 flooding broadcast scheme which is not suitable for wireless ad hoc network. 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 related to a node's memory constrain. A large T_l setting will increase the message broadcasting successful rate at the expense of higher memory requirement and greater protocol overhead.

3.1.3 Variations of Gossip Protocol

It is more clear when we category different variations of 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 *Probability of Gossip* can be set to a fixed value or be 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 has been paid to another gossip protocol parameter *Fanout*. Fixed *Probability of Gossip* approaches can calculate its probability based on network density, distance among nodes, and speed [8]. In this scheme, nodes forward an incoming message with a fixed p_g, and the probability of not forwarding the incoming packet is $(1-p_g)$ [8]. The major challenge of fixed scheme is determining the optimal p_g. Due to the dynamic nature of wireless ad-hoc network, even an optimal initial global p_g could become sub-optimal overtime.

Adaptive *Probability of Gossip* approaches uses local or global network information such as density and speed to adjust individual or global probability. In adaptive scheme, there are adaptive non-counter-based schemes and adaptive counter-based schemes [8]. Adaptive density-based schemes usually utilize node's degree metrics. In (nb-scheme) the p_g has an inverse relationship with the number of neighbors of a node [9]. If we denote 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 used so that the maximum and minimum probability can be adjusted [9]. 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 an sparse area, higher *Probability of Gossip* would be more desirable. Some paper [10][11] suggested schemes that dynamically adjust *Probability of Gossip* based on Received Signal Strength (RSS) or euclidean distance. In [11], the authors denoted the relative distance between node i and node j by D_{ij} and the average transmission range by r. The *Probability of Gossip* is calculated using the following equation:

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

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 [8]. Similar to non-counter-density-based schemes, some paper [12] used node's degree in conjunction with a counter. The equation used to calculate *Probability of Gossip* 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 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 paper [13][14] used the distance between nodes as a metrics combining 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 only forward new broadcast messages when it receive one, it is called a *push* gossip protocol. Similarly, when each node only request for new broadcast messages from other nodes, it is called a *pull* gossip protocol. Our gossip protocolcombined both mechanisms thus it is called a *push-pull* gossip protocol.

Our basic push-pull gossip protocol utilizes three packet types to perform. They 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 to the sender that receiver node already received that message before. The Request packet is used for a node to ask for 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 mean it will stop gossiping the new message.
- Rule 6: When a node received a duplicate message from anther node, it will send an Ack packet back.

```
// 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

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 switch from sleep state to gossip state and start sending out this message to one of its neighbors. This gossip process runs every 5 seconds unless the node switched to sleep state. When a node switched to sleep state, it will do nothing. Now when a node received 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 received an Ack packet, it will switch to sleep state. Lastly, if a node received a Request packet, it will send its latest message back to the sender.

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. 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 work. 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 denoted the *Remaining Energy Fraction* as E_{frac}. The function that used to calculate *Fanout* is defined as follow:

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

The function is plotted in Figure 3.2.

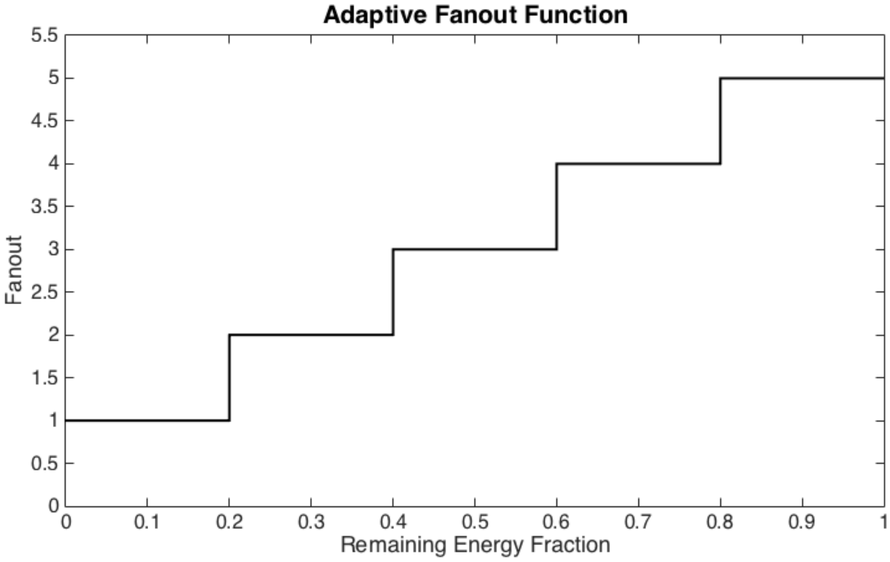

FIGURE 3.2: Adaptive Fanout Function Plot

The basic idea of our fanout function is that the *Fanout* of a node will gradually stepping down as its battery energy being drained. We believe this new fanout function can combine the advantages of both worlds. When a node's has plenty of energy left, it will reach out to more neighbors and facilitate message broadcasting process. As a node's energy gets lower, it will conserve its battery energy by contacting less neighbors thus extend network lifetime.

```
// 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>
```

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

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 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, the actual *Fanout* will be 3.

Chapter 4

Implementation

In order to evaluate our proposed energy-aware gossip broadcasting protocol, we implemented the protocol in a open-source software called Network Simulator 3 (ns-3). From 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 application layer. The adaptive fanout gossip protocol is the protocol entity that we are interested in studying. It utilized underlining ICMP extension to communication 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 set up, 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 protocolimplementation, 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 [15]. 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 [16]. 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 4.1: ICMP Header Structure

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

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

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 type 42 to 255 are reserved for further development, we decided to extend 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

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

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 kick 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 packet* (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 running. 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 time stamps of each broadcast message back. 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 *sendPayload()* that is in application layer. Then *sendPayload()* would have to call the function *sendMessage()* in ICMP which is in network layer. On the receiving end, a node first would receive the new broadcast message in network layer. The message is handled by a function in ICMP called *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
- Store every broadcast messagewithout duplication
- Store the time stamps for each received new broadcast message
- Report each new broadcast messagetime stamps back to the source node

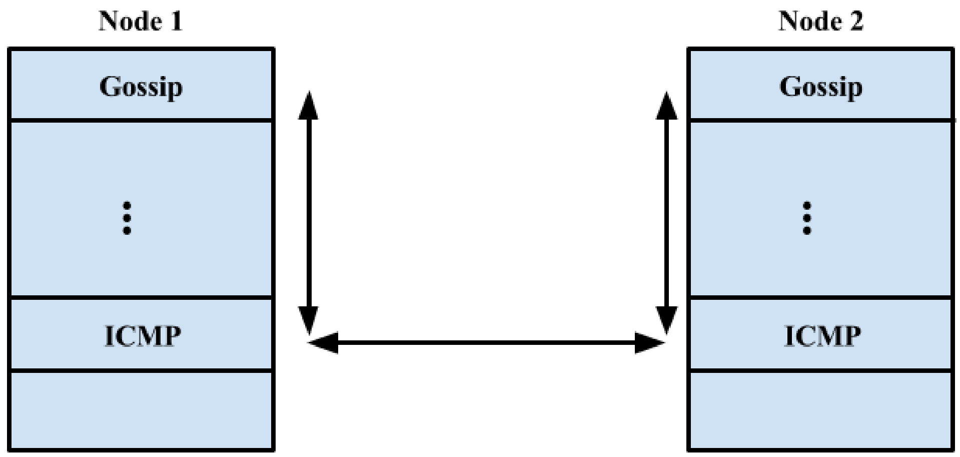

FIGURE 4.1: An example of how two gossip nodes communicate

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 worthing 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 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 set up, because we want to collect simulation data about network lifetime, the simulation stop time is set to be large enough so that the energy source will be depleted first. Any depleted energy source will automatically trigger the simulation to stop. Topology wise, we wanted it to be a close resemble to Wireless Sensor Network (WSN) or MANET. In other words, we wanted to avoid 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 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 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 nodeplacement, the program can access each node's coordinates. Assuming node 1 ($node_1$) has the coordinate of (x_1, y_1) and node 2 ($node_2$) has the coordinate of (x_2, y_2), the distance between $node_1$ and $node_2$ can be easily computed by the distance formula. Therefore, if we denote the distance between $node_1$ and $node_2$ is d, then $d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$.

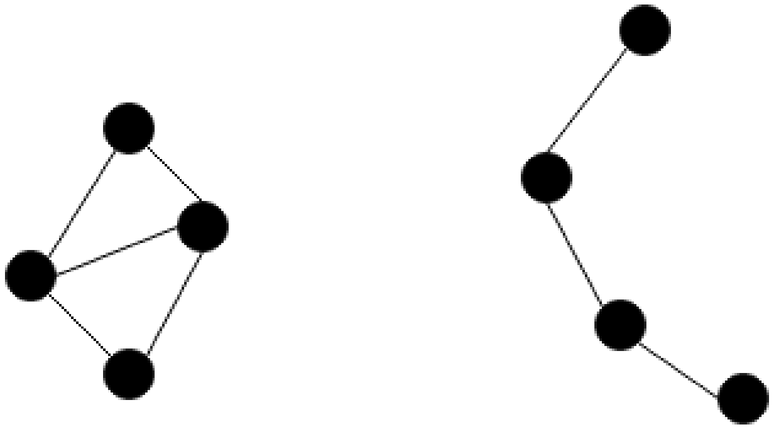

Figure 4.2: An example when two separate subnets formed

Now if we denote each gossip node's WiFi range to be R, then $node_1$ and $node_2$ are neighbors when $d \leq R$. While $node_1$ and $node_2$ are not neighbors 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 work flow 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 nodeand 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 protocoland 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 nodesis 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 that it will 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 represent the time stamps for each generated new broadcast message. Similarly, it will also access n vectors from n gossip nodes. These vectors stored 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 nodeand the protocol successfully broadcast m messages, if we denote the vector on the source node side to be $T_s =< t_{s1}, t_{s2}, \ldots, t_{sm} >$ 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 are $T_{delay} = T_g - T_s =< t_{g1} - t_{s1}, t_{g2} - t_{s2}, \ldots, t_{gm} - t_{sm} >$. For the scenario where n gossip nodesparticipated 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 nodereceived the message and the time the source node generate that messageis 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 need to be done.

For the overhead, we defined it as the total number of packets this protocol sent during the simulation. This includes Ack packet, Request packet, and Data packet. Our simulation control program first will access the *packetSent* counter on each gossip node. Then it will average over total number of broadcast messages(m messages). Finally, it will take that number and average over 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 over number of gossip nodes (n). And finally it will take that number and average over total number of broadcast messages (m).

The network lifetime is defined as the time duration which all gossip nodes have energy to receive and transmit packets. Therefore, as we stated earlier, when any gossip node's energy is depleted, That moment is consider the end time of our simulation. And the program simply output 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 Success Broadcast Messages.*

5.1.1 Average Network Lifetime

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

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 and receive new broadcast messages. Thus, this network is considered to be physically unable to broadcast messages successfully. Now the definition of *Average Network Lifetime* is simply just an average over all *Network Lifetime* under n gossip nodes 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 we jump into the definition of *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.

Because here we are trying to measure the broadcast time of a certain message, it only makes sense when we sample the maximum delay among all gossip nodes because the message received time of the last gossip node determined our message broadcast time for a particular message. Now the *Average Message Broadcast Time* is just an average over all message broadcast time under n gossip node case. Now because the randomness of our proposed gossip protocol, each simulation under n gossip node case can result in different number of success broadcast messages. Therefore, for n gossip node case simulations, we first calculate the *Average message broadcast time* for each simulation. And then we perform the average among those number to obtain this performance metrics data.

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* here is defined as the total number of packets sent by a gossip node. These packets includes Ack packet, Request packet, and Data packet. For every simulation, in order to accurately measure the overhead for each gossip node and for each broadcast message, we simply performed an average over n. And then take that number and further average over number of broadcast messagesbeing 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* 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. For each simulation, first we average among every gossip node's energy consumption. Then we take that number and average among m messages. If we denote E_k as the *Average Energy Consumption Per Node Per Message* for the kth simulation under n gossip nodes. Then

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

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:

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

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. They are $f = 1, 5, 10$. By obtaining performance metrics data from these three constant fanout settings, we can study how different *Fanout* affect protocol performance.

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
- 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, it will trigger the termination of our simulations.

5.3 Results Analysis

In order to obtain simulation data for these $f = 1, 5, 10, adaptive$ settings, we ran 100 simulations for each number of gossip nodes setting. Because of the random placement of gossip nodes can potentially generate a disconnected topology, some of the simulations will be terminated in 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 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 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* over 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 $f = 1$ setting across different number of gossip nodes settings. We can see a huge performance improvement when *Fanout* is switched from 1 to 5. That is because a gossip node can forward the broadcast message to more neighbors comparing to $f = 1$ setting. However, the performance improvement 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

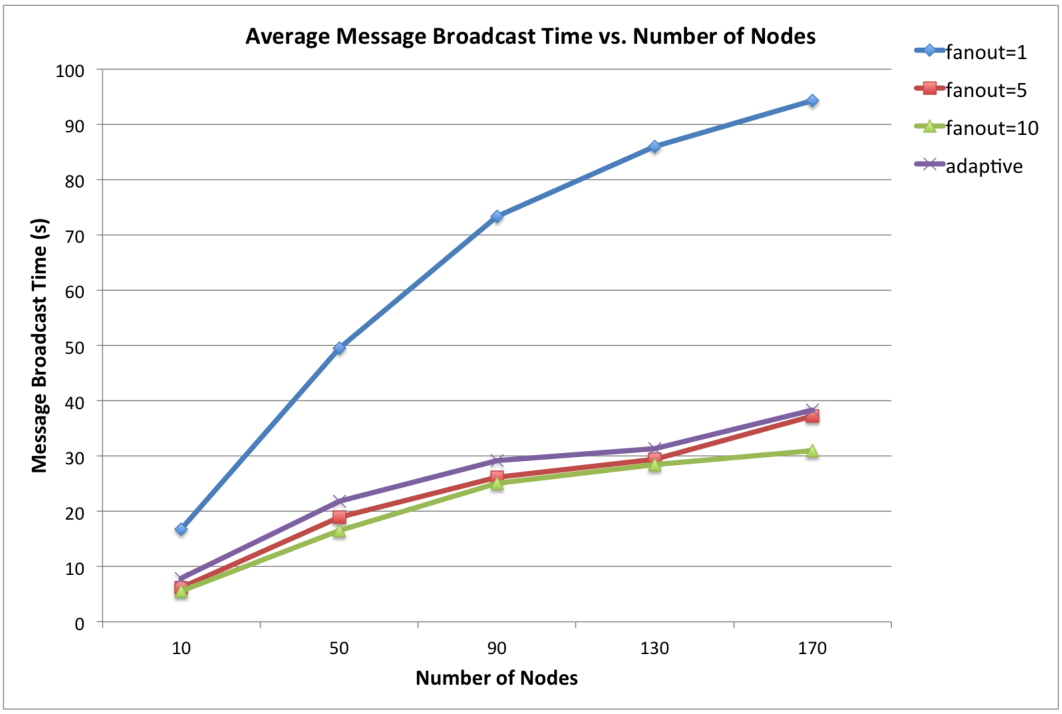

FIGURE 5.1: Average message broadcast time vs. number of nodes

exceeds the number of neighbors a gossip node has will bring no additional performance boost. Our adaptive fanout approach performed 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 consume 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 the *Average Message Broadcast Time* is shorter. However, our proposed adaptive fanout setting can achieve even better *Average Network Lifetime* comparing to $f = 5, 10$ settings while still perform as good as the other two in terms of *Average Message Broadcast Time*.

From Figure 5.3, we can see that as 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 here we would like to point out is that $f = 1$ 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 messagesit can broadcast is less than other *Fanout* settings.

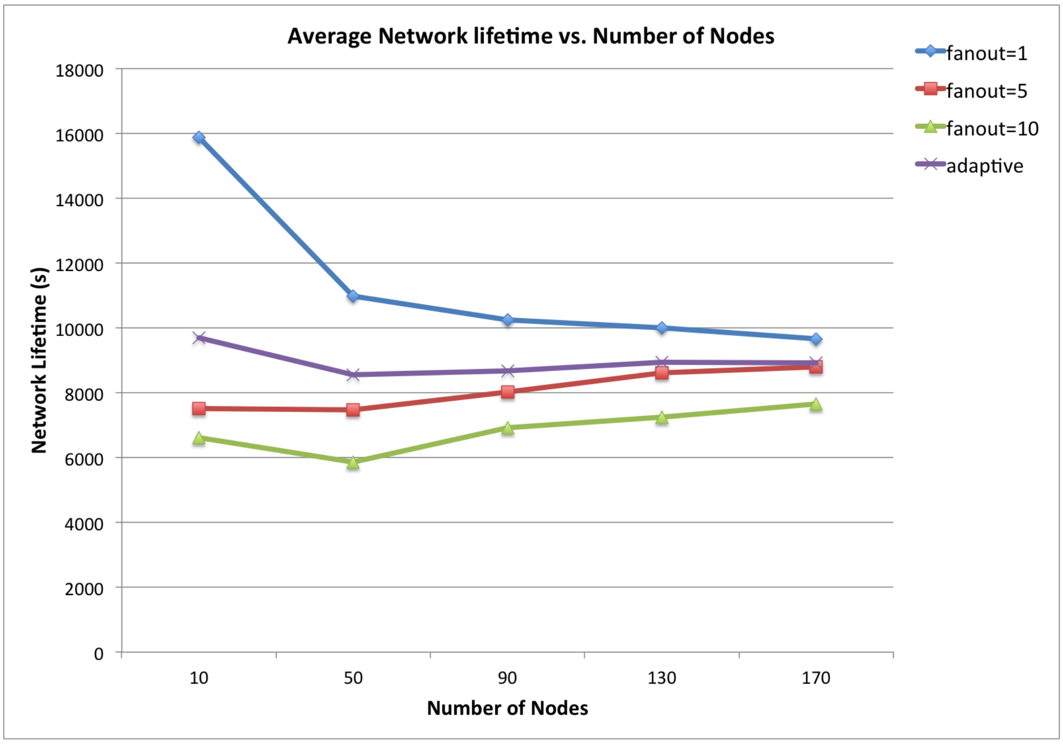

FIGURE 5.2: Average network lifetime vs. number of nodes

Since we computed this metric on a per message basis, it would result in a higher *Average Energy Consumption*. Under $f = 5$ and adaptive fanout setting, their *Average Energy Consumption* plots are almost overlapping each other. The reason is the trade off between *Average Message Broadcast Time* and *Average Network Lifetime*. $f = 10$ 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* over various number 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. That 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 messagesbroadcast over different number of gossip nodes. $f = 1$ setting can deliver the least amount of messagescomparing to other *Fanout* settings. In general, higher *Fanout* setting will increase the number of messagesthe 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 performed as good as $f = 5, 10$ setting while as shown in Figure 5.2 it has longer *Average Network Lifetime*.

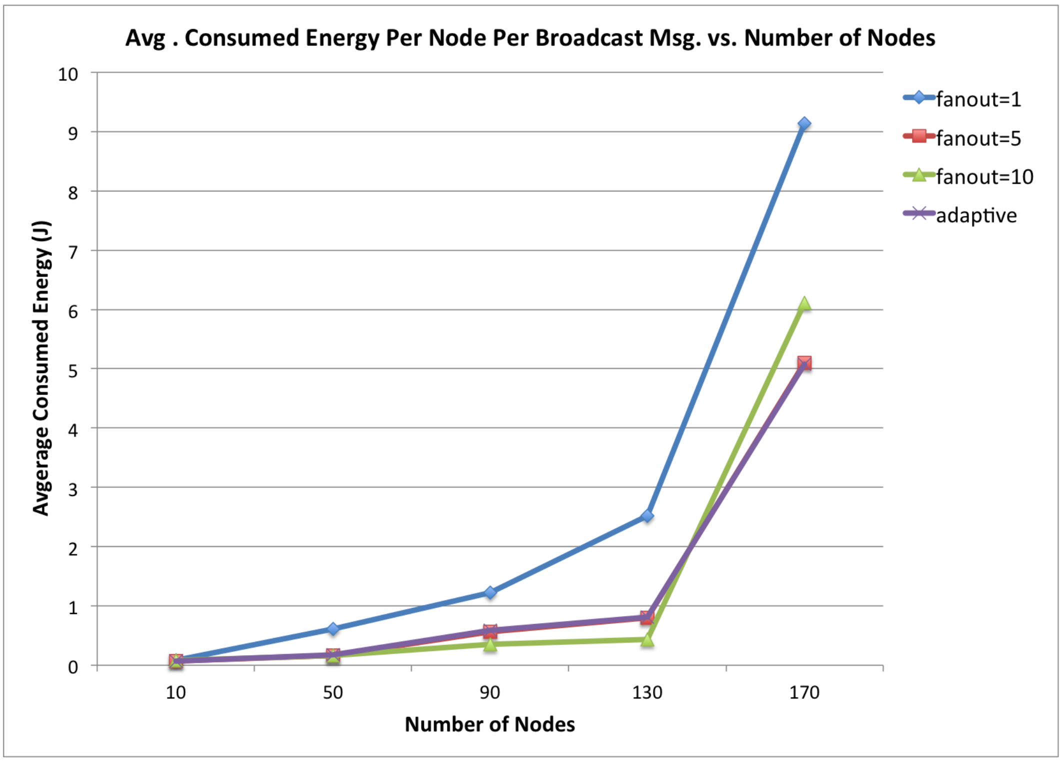

FIGURE 5.3: Average consumed energy per node per message vs. number of nodes

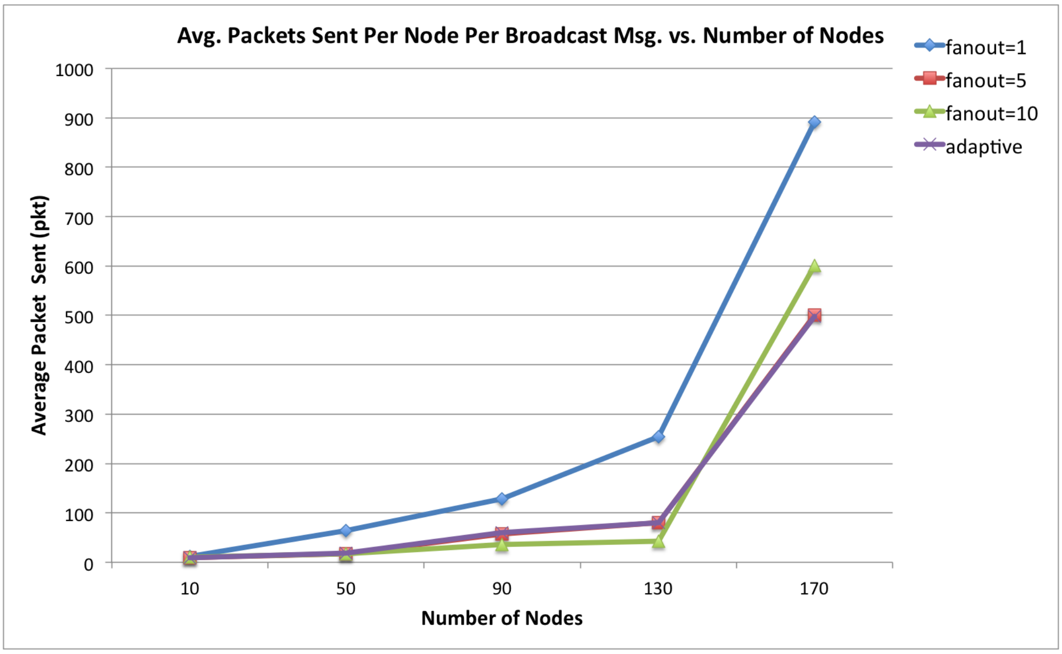

FIGURE 5.4: Average overhead per node per message vs. number of nodes

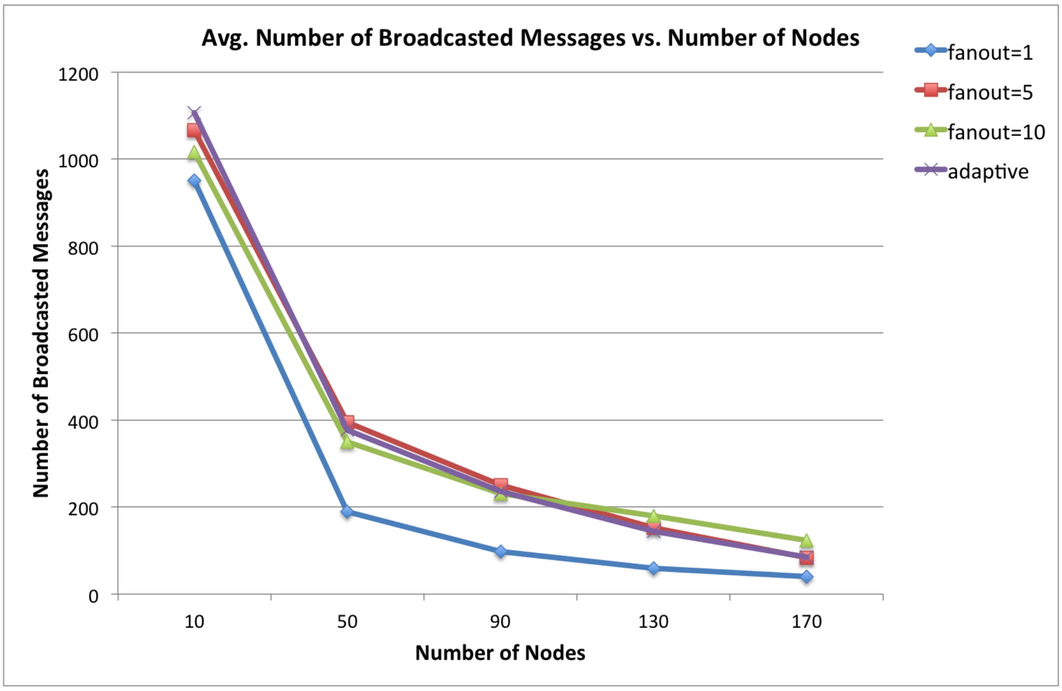

Figure 5.5: Average number of broadcast messages vs. number of nodes

Chapter 6

Conclusions and Future Work

In this thesis, we introduced research progress in gossip techniques for wireless broadcasting in recently years. We pointed out that despite all these efforts dedicated into reduing gossip broadcasting protocol overhead, very little research has focused on energy efficiency and network lifetime. Those aspect used to be not very important when designing new broadcasting protocols because nodes usually have stable power supplys. However, with the emergence of Internet of Things (IoT) devices, broadcasting protocols that take energy consumption into account and optimizing it will be favored over those that do not. 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 performance of our proposed approach, we designed several metrics and we developed the protocol in open-source software ns-3. Simulation results showed that comparing to constant $f = 5$ setting, our adaptive approach significantly extended network lifetime while only performed slightly slower in term of message broadcasting time. We suspect the casue for marginal performance improvements for $f = 10$ setting comparing to $f = 5$ 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, $f = 1$ setting has the worst message broadcasting time. But on the flip side, it would result in a longest network lifetime which could 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 perform as good as high *fanout* setting in term of broadcasting time. Constant $f = 1$ setting is recommended for networks that consists of nodes with strict energy constrain.

For future work, we would like to use multicast instead of multiple unicast for each node to send the new message. 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 consum XJ while five unicast will consum $5XJ$. Another interesting scenerio would be to set very different initial energy but same battery capacity for each node. Thus each node would be operating at different *fanout* setting from the begining due to different remaining energy fraction. We believe this would better capture the advantage of our proposed appraoch over constant *fanout* setting.

Appendix A

An Appendix

Bibliography

[1] L. Atzori, A. Iera, and G. Morabito, “The internet of things: A survey,” *Computer networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[2] K. Jenkins, K. Hopkinson, and K. Birman, “A gossip protocol for subgroup multicast,” in *Distributed Computing Systems Workshop, 2001 International Conference on*, pp. 25–30, IEEE, 2001.

[3] A. P. Jayasumana, Q. Han, and T. H. Illangasekare, “Virtual sensor networks-a resource efficient approach for concurrent applications,” in *Information Technology, 2007. ITNG’07. Fourth International Conference on*, pp. 111–115, IEEE, 2007.

[4] N. M. K. Chowdhury and R. Boutaba, “A survey of network virtualization,” *Computer Networks*, vol. 54, no. 5, pp. 862–876, 2010.

[5] S. Abdelwahab, B. Hamdaoui, and M. Guizani, “Cloud-assisted remote sensor network virtualization for distributed consensus estimation,” *arXiv preprint arXiv:1501.03547*, 2015.

[6] R. Chandra, V. Ramasubramanian, and K. P. Birman, “Anonymous gossip: Improving multicast reliability in mobile ad-hoc networks,” in *Distributed Computing Systems, 2001. 21st International Conference on.*, pp. 275–283, IEEE, 2001.

[7] Y.-C. Tseng, S.-Y. Ni, Y.-S. Chen, and J.-P. Sheu, “The broadcast storm problem in a mobile ad hoc network,” *Wireless networks*, vol. 8, no. 2-3, pp. 153–167, 2002.

[8] D. Reina, S. Toral, P. Johnson, and F. Barrero, “A survey on probabilistic broadcast schemes for wireless ad hoc networks,” *Ad Hoc Networks*, vol. 25, pp. 263–292, 2015.

[9] J. Cartigny and D. 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*, pp. 10–pp, IEEE, 2003.

[10] W. Qing-wen, S. Hao-shan, and Q. Qi, “A dynamic probabilistic broadcasting scheme based on cross-layer design for manets,” *International Journal of Modern Education and Computer Science*, vol. 2, no. 1, p. 40, 2010.

[11] N. Wisitpongphan, O. K. Tonguz, J. S. Parikh, P. Mudalige, F. Bai, and V. Sadekar, "Broadcast storm mitigation techniques in vehicular ad hoc networks," *IEEE Wireless Communications*, vol. 14, no. 6, pp. 84–94, 2007.

[12] A. Lee and I. 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*, pp. 1131–1135, ACM, 2010.

[13] I. A. Khan, A. Javaid, and H. L. 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)*, pp. 1–6, IEEE, 2008.

[14] H. Ling, D. Mossé, and T. Znati, "Coverage-based probabilistic forwarding in ad hoc routing," in *Proceedings. 14th International Conference on Computer Communications and Networks, 2005. ICCCN 2005.*, pp. 13–18, IEEE, 2005.

[15] F. K. James and W. R. Keith, *Computer networking a top-down approach featuring the internet.* Addison-Wesley, Reading, 2004.

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