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.

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

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.

1.1 A Section

1.1.1 A Subsection

1.2 Another Section

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. [2] 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. [2] 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 [2] 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. [2] 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. [3] 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. [3] Moreover, virtualization on top of IP is fixed to the network layer protocol and cannot deploy IP independent mechanisms. [3] 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. [4] 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. [4] 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[5][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.

2.4 A Section

2.4.1 A Subsection

2.5 Another Section

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 adjust 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 remain to have a fast and reliable broadcasting performance. The parameter that we focused on shifted from *Probability of Gossip* to *Fanout* as well.

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 will conserve energy but result 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 disconnected to 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:

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

The function is plotted in Figure 3.2.

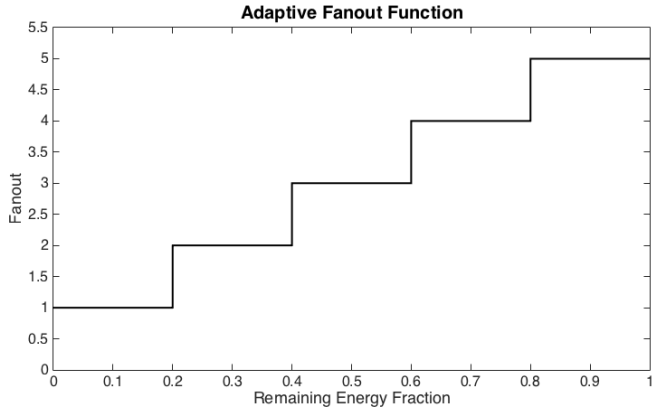

FIGURE 3.2: Adaptive Fanout Function Plot

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). There are two folds in our implementation. First fold is that we implemented the basic push-pull gossip protocol. The second fold is how we implemented proposed adaptive fanout scheme.

4.1 Basic Push-pull Gossip Protocol Implementation

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: 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: Gossip Protocol Control Messages

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.

In order to 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
- collect time stamps from gossip nodes.

Every time when a new broadcast message is generated, it will send it to one of the gossip nodes thus start the whole process. Except the first broadcast message, every other new broadcast message will only be generated and sent out when it received n time stamps from all gossip nodes for the previous broadcast message. Because in the program, we make sure that each gossip node will only send the time stamp of a specific broadcast messageonce, it is a good indication that this message has been successfully broadcast. In order to support these roles that a source node play, we deployed an UDP server application so that every gossip node could connect to it and report its broadcast message time stamps.

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 **??**.

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

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 Environment Setting

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 $100m \times 100m$ 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:

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

Because of randomly 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. Therefore, we applied Depth First Search (DFS) algorithm to ensure that all nodes in the network are connected in some way. Even if the above situation does not happen, a case that shown in Figure 4.1 can occur where a network is divided into two separated subnets.

calculate neighbor list for each node range x1y1 x2y2 distance ¿ range

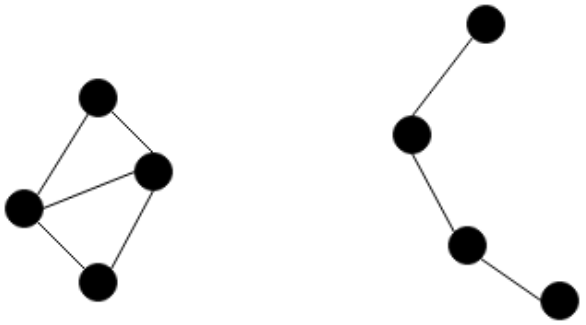

FIGURE 4.1: An example when two separate subnets formed

Every node's coordinates are used to calculate neighbors list for each node. In practice, the global access of this information is usually not easy to obtain. Thus, Hello packets are used to compile neighbors list for each node.

Because nodes are randomly placed in a square area, with fixed WiFi range there could be a case that each node has at least one neighbor but the network is separated into two subnets unconnected. [twoSepNet.png]

Therefore, I used an algorithm that uses depth-first search algorithm to determine whether all nodes get visited during the recursive search. If the algorithm successfully traverse all nodes, it is considered a complete graph which means the network is connected. If the algorithm yield a failure, this trial will be rejected and the simulation will move on to next trial.

In order to collect benchmark of adaptive fanout gossip protocol, one extra node is place in the area. Its functions are generate new packets, collect other nodes' received time of every packets, and store new packets generate time. A trimmed version of adaptive fanout gossip protocol is used to generate new packets. An UDP server application is installed on the node to collect other nodes' received time of every packets.

Adaptive fanout gossip protocol is installed on all other nodes in the network. Besides that, a UDP client application is also installed on them to send received time of every packet. The simulation stops when any one node's energy is depleted. After that, all the data collected is processed to generate our performance metrics.

the work flow

source DSSSRate 11Mbps DsssRate 1Mbps maxRange = 50m num of nodes = 10 simulation time = 100000.0s initialEnergy = 108J

Gossip Interval 1s request interval 5s

- Extend the Internet Control Message Protocol (ICMP) to support transmitting three simple control messages needed for gossip protocol.
- Develop the gossip protocol application to be installed on network nodes.
- Set wireless ad-hoc network attributes and gossip protocol attributes, which has already be presented in section **??**.
- Import nodes connectivity information from topology files and export simulation results for performance evaluation.

4.4 Performance Metrics

Life Span of the Network

Definition: The time when any node's battery die

Because the goal of this protocol is to broadcast any new packet, the network will lose the physical ability whenever any node lost wireless connection to its neighbors due to energy depletion. Therefore, this metric indicates how long a network with size n could stay connected using a broadcast protocol.

Average Packet Broadcast Time

Let's think about broadcasting one packet from one node to (n-1) nodes, the packet broadcast time of this packet for this network is the difference between the time when last node received the packet and the time this packet was first sent.

Because for each independent trial, multiple packets will be sent. The packet broadcast time is calculated for each packet in the way described above. In the end, we average the times for each scenario (e.g. n = 10).

Need to develop a mathematical equation for this

The average packet broadcast time indicates the time needed for a packet to reach every node in the network.

Protocol Overhead

There are three types of protocol packets for the protocol. The ack packet, solicit packet, and the payload packet. The acknowledgment packet is sent to the sender from receiver when a node received a payload that it already received before. The solicit packet is sent to query a random neighbor for its latest payload. For example, in one trial with n nodes, if it broadcaster m packets, then the average protocol overhead is defined as:

```
[fontsize=\small]
#Nodes
0
1
2
#Edges
(0, 1)
(1, 2)
```

FIGURE 4.2: A topology file of a linear topology with three nodes.

$overhead = ()(p1 + p2 + \ + p_n)/n)/m$

This metric indicated how many packets is needed for a node to facilitate broadcasting one packet.

Consumed Energy

The consumed energy per node per packet is defined as

energy = ((e1 + e2 + \ + en)/n)/m

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

4.4.1 Gathering Simulation Data

To evaluate the performance of gossip protocol, we use several randomly generated topology files with the number of nodes as variable. Those topology files are derived from a random geometric graph network, which was created by uniformly and randomly placing nodes into a space and then connect nodes whose distance is smaller than some given radius.

Each topology file contains the number of nodes as well as all the edges, which represents the connections between nodes. As an example, the content of a simple topology file is shown in figure 4.2. A parser written in C++ is developed to create the given number of nodes in NS-3 and installed the gossip protocol application on them. Thereafter, all edges are parsed and created accordingly. Each node holds an NS-3 Ipv4Interface with assigned Ipv4 address and stores the Ipv4 addresses of his neighbors.

For the simulation, we set the link rate for all connections to be 1Mbp. Also, all nodes are instructed to execute the gossip process of sending out data periodically every 5ms. The interval of requesting new data is set to 5s.

To allow the performance analysis, all nodes count the number of data packets they sent. All nodes would track how many hops the data message experienced before reaching them and they record the time when they received the data message as well. For every single simulation, we collect the information from the nodes and determine the average amount of data packets sent per node and average number of hops per node. Moreover, the information about how long it took for the message to reach the "last" node is also recorded.

This information is determined and stored for each of the several hundred topology files. It should be noted that due to time limitations each topology file was only simulated once. Finally, we did statistical analysis upon those collected data in the hope of verifying the assumptions we made in section **??**.

Chapter 5

Performance Evaluation

Talk about the settings

In summary, the simulation parameters are set to be as follows:

- Number of Nodes = 10, 20, 30, , 200
- Simulation time = large enough to ensure energy depletion happens first
- Maximum Wifi Range = 50m
- Initial Energy = 108J
- Gossip Interval = 1.0s
- Solicit interval = 5.0s

The number of nodes, simulation time, maximum wifi range, initial battery energy, gossip interval, and solicit interval are all the parameters I could control in the simulation. The number of nodes is ranging from 10 to 200 with step 10. The simulation time is set to be large enough to ensure any node's battery will die before the simulation stops. The maximum wifi range is set to be 50 meters so that no one node is connected to all other nodes. The initial battery energy is set to be 1080 Joule (equivalent to 100mAh) for every node. The gossip interval is set to be 1 second meaning for any node every second the gossip process will wake up and check if it need to gossip a new packet to its neighbors. The solicit interval is set to be 5 seconds which means every 5 seconds, a node will randomly pick a neighbor and query for their latest received packet.

The simulation will stop when any node's energy was depleted. For each scenario (e.g. n = 10), 100 independent trials were simulated in order to gather performance data. Any simulation parameters other than number of nodes stay the same through all simulation trials.

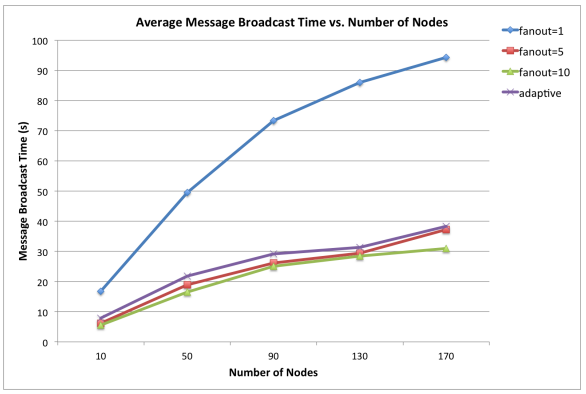

FIGURE 5.1: Average message broadcast time vs. number of nodes

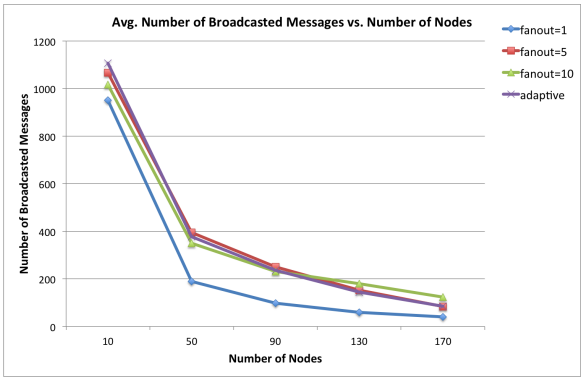

FIGURE 5.2: Average number of broadcasted messages vs. number of nodes

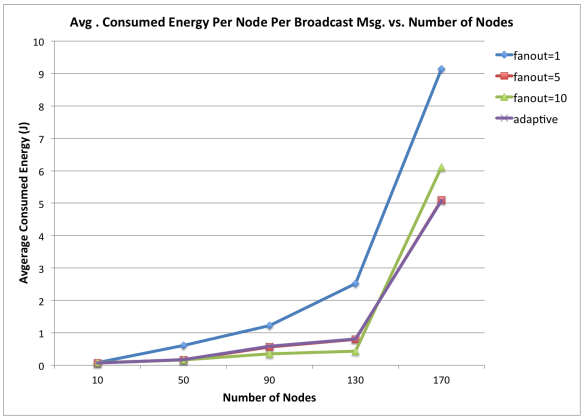

FIGURE 5.3: Average consumed energy per node per message vs. number of nodes

5.1 Results Analysis

Fanout of gossip protocol is defined as the number of neighbors a node contact each time when a new packet is received. It is one of the parameters that control the behavior of gossip protocol. In one scenario, fanout is set to be one. In another scenario, fanout is set to be five. These two scenarios 's trials are conducted to compare to the adaptive fanout scenario.

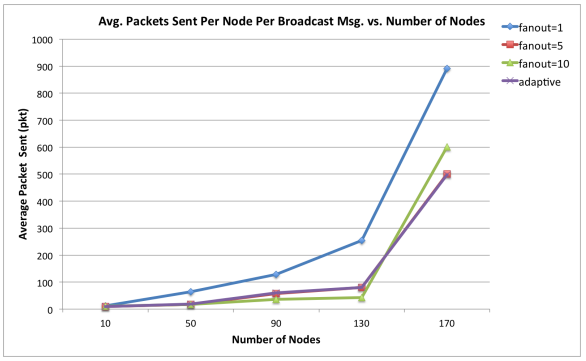

FIGURE 5.4: Average packets sent per node per message vs. number of nodes

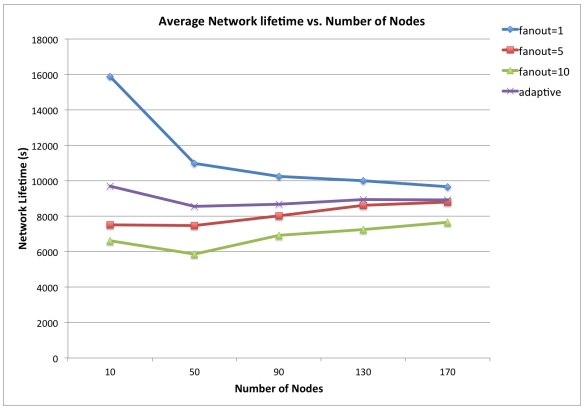

FIGURE 5.5: Average network lifetime vs. number of nodes

(from 563 paper).

The random generated topology files are the input of our simulations. There are 7 different cases where the number of nodes vary from 100 to 1000 , increasing with 150 nodes step. And for each case, we sampled 100 topology files to run the simulations.

First, I analyzed the average number of data packets each node sent, depicted in fig. **??**. It is important to note that the the collected average number for one topology file was again averaged over all reported values produced by topology files with the same number of nodes. Thus, the error bar is an indicator how consistent the average number is. It can be deduced from fig. **??** that this value is ranging from 190 data packets per node to 280 data packets per node depending on the scale of the network. But considering the network scale, average number of data packets per node didn't increase proportionally as we can see in fig. **??**. This metrics actually decreased exponentially.

Since when I collected the different data (average ICMP messages per node) from wired peer-to-peer network environment, here I could not perform a fair comparasion between these two evnironment. But as you can see in fig. **??**, the average ICMP messages per node is significantly lower than in the wireless ad-hoc network. I believe the reason behind this is the shared medium for wireless communication. With shared medium,

collision is prone to occur thus RTS/CTS plays an important role during the whole communicating process. Thus higher average data packets sent per node is expected.

Second, the average number of hops per node is analyzed. This is different from what we collected in the P2P environment which is maximum number of hops. Thus fair comparison between these two environment can not be performed here. In the wireless ad-hoc network, as we can see in fig. **??**, the average number of hops per node mostly concentrated around 2.3 hops regardless of the growing number of nodes. For wired P2P network, the maximum hops is around 16.5. But the standard deviation has the tendency to decrease which is a positive sign since we hope the gossip protocol performance metrics would converge as the network grows. Nonetheless, the overall impression for both network environment is that the number of hops either average or maximum are more or less constant. But why is that the average hops per node could remaine 2 to 3 in a wireless ad-hoc network? My explaination is that since the topology of the network is almost a complete graph as you can see in fig. **??** showing a simple 10 nodes case, with gossip interval 5ms and request interval 5s, before any node send out request packets, chances are the starting node already gossipped with most of the node in the network result to a low average hops per node.

Moreover, figure **??** illustrates the time needed to spread the message across the whole network. For the P2P network environment, the average time needed to spread the message is found to be more or less constant and slightly less than 15s. Due to the huge difference in the gossip-interval-time (5ms) and solicit-interval-time (5s), only the influence of the solicit-interval can be deduced from the results. One can see, that the time needed to spread the message is fluctuating due to the random nature of the gossip protocol, especially for the case of a number of nodes of 100, where the standard deviation is the largest. However, when we compare this result with ad-hoc network simulation result, the different between them is significant. For the latter case, the spread time starts around 190s and grows almost linearly with the number of nodes in the network. In term of absolute maginitude, wireless network perform much worse than P2P network. However, this result is total within our expectation since wireless communication often encounter collision problem and thus result in longer spread time.

Section **??** proposes further work which can be done to gain a more thorough evaluation.

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. I 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] 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.

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

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

[5] 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.

[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.