A peer-to-peer protocol for prioritized software updates on Wireless Sensor Networks

Mr. Natchanon Nuntanirund

A Thesis Submitted in Partial Fulfillment of the Requirements

for the Degree of Master of Engineering in Computer Engineering

Department of Computer Engineering

FACULTY OF ENGINEERING

Chulalongkorn University

Academic Year 2019

โปรโตคอลเพียร์ทูเพียร์สำหรับการปรับปรุงซอฟต์แวร์ไร้สายแบบหลายระดับความสำคัญบนเครือข่ายเซ็นเซอร์ไร้สาย

นายณัฐชนน นันทนิรันดร์

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2562

Thesis Title	A peer-to-peer protocol for prioritized software updates on Wireless Sensor Networks
By	Mr. Natchanon Nuntanirund
Field of Study	Computer Engineering
Thesis Advisor	Assistant Professor NATAWUT NUPAIROJ, Ph.D.

Accepted by the FACULTY OF ENGINEERING, Chulalongkorn University in Partial Fulfillment of the Requirement for the Master of Engineering

..... Dean of the FACULTY OF ENGINEERING

(Professor SUPOT TEACHAVORASINSKUN, Ph.D.)

THESIS COMMITTEE

..... Chairman

(Associate Professor KRERK PIROMSOPA, Ph.D.)

..... Thesis Advisor

(Assistant Professor NATAWUT NUPAIROJ, Ph.D.)

..... Examiner

(Associate Professor KULTIDA ROJVIBOONCHAI, Ph.D.)

..... External Examiner

(Assistant Professor Putchong Uthayopas, Ph.D.)

ณัฐชนน นันทนิรันดร์ : โปรโตคอลเพียร์ทูเพียร์สำหรับการปรับปรุงซอฟต์แวร์ไร้สายแบบหลายระดับความสำคัญบนเครือข่ายเซ็นเซอร์ไร้สาย. (A peer-to-peer protocol for prioritized software updates on Wireless Sensor Networks) อ.ที่ปรึกษาหลัก : ผศ. ดร.ณัฐวุฒิ หนูไพโรจน์

การปรับปรุงซอฟต์แวร์เป็นสิ่งสำคัญสำหรับอุปกรณ์ในเครือข่ายเซ็นเซอร์ไร้สาย สำหรับการเพิ่มความสามารถใหม่ ปรับปรุงประสิทธิภาพ หรืออุดช่องโหว่ความปลอดภัย แต่เนื่องจากอุปกรณ์ที่ติดตั้งใช้งานอยู่บางอุปกรณ์ไม่สามารถเข้าถึงได้โดยตรง โปรโตคอลการกระจายข้อมูลจึงถูกใช้ในการกระจายข้อมูลซอฟต์แวร์เข้าไปให้กับอุปกรณ์เหล่านั้น อย่างไรก็ตามการปรับปรุงซอฟต์แวร์แต่ละครั้งอาจมีความสำคัญที่แตกต่างกัน เช่น การปรับปรุงซอฟต์แวร์เพื่อเพิ่มความสามารถเพียงเล็กน้อยอาจไม่จำเป็นต้องได้รับการปรับปรุงอย่างเร่งด่วน เมื่อเทียบกับการปรับปรุงซอฟต์แวร์เพื่อการอุดช่องโหว่ร้ายแรง งานวิจัยนี้นำเสนอโปรโตคอลการกระจายข้อมูลซึ่งทนต่อความผิดพร่องและสามารถปรับแต่งเพื่อแลกเปลี่ยนระหว่างการบริโภคพลังงานและความเร็วได้ โดยนำแนวคิดต่าง ๆ จากโปรโตคอลบิตทอร์เรนต์มาประยุกต์ใช้ เช่น การจับมือ, การแบ่งส่วนรับส่ง, และ Choking Algorithm เป็นต้น

สาขาวิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่อนิสิต .....

ปีการศึกษา 2562 ลายมือชื่อ อ.ที่ปรึกษาหลัก .....

5970430921 : MAJOR COMPUTER ENGINEERING

KEYWORD: Wireless Sensor Networks, Data Dissemination, BitTorrent, Software updating

Natchanon Nuntanirund : A peer-to-peer protocol for prioritized software updates on Wireless Sensor Networks. Advisor: Asst. Prof. NATAWUT NUPAIROJ, Ph.D.

Software updating is essential for devices in wireless sensor networks for adding new features, improving performance, or patching vulnerabilities. But since some deployed devices are unable to be accessed directly, data dissemination protocol is used for distributing the update to those devices. However, each software updating may have different priority, for instance, an update that adds an optional feature does not have to be applied as fast as an update that patches severe vulnerabilities. This research presents a reliable data dissemination protocol which is configurable for energy-speed trade-off deriving some concepts from BitTorrent such as Handshaking, Segmented File Transfer, and Choking Algorithm.

Field of Study: Computer Engineering Student's Signature .....

Academic Year: 2019 Advisor's Signature .....

ACKNOWLEDGEMENTS

I would like to express my appreciation to my advisor, Asst. Prof. Natawut Nupairoj, for his help and guidance that kept me constantly engaged with my research. Without his support, this research would not have been possible.

The Scholarship from the Graduate School, Chulalongkorn University to commemorate 72nd Anniversary of his Majesty King Bhumibol Adulyadej is gratefully acknowledged.

Finally, I would like to thank my friends and family for believing in me through the difficult times.

Natchanon Nuntanirund

TABLE OF CONTENTS

Page

..... iii

ABSTRACT (THAI)..... iii

..... iv

ABSTRACT (ENGLISH)..... iv

ACKNOWLEDGEMENTS.....v

TABLE OF CONTENTS..... vi

List of figures..... ix

1 INTRODUCTION 1

1.1 Motivations 1

1.2 Objectives.....2

1.3 Scope of Work.....2

1.4 Research Procedure 2

1.5 Expected Benefits.....2

1.6 Published Paper.....2

2 TECHNICAL BACKGROUNDS AND RELATED WORKS 3

2.1 Technical Backgrounds.....3

2.1.1 6LoWPAN 3

2.1.2 IPv6 Routing Protocol for LLNs (RPL) 4

2.1.3 IEEE 802.15.4.....6

2.1.4 BitTorrent.....7

2.2 Related Works.....8

2.2.1 IPv6 Multicast Forwarding in RPL-Based Wireless Sensor Networks (MPL) 8

2.2.2 Stateless Multicast RPL Forwarding (SMRF) 9

2.2.3 Enhanced Stateless Multicast RPL Forwarding (ESMRF) 11

2.2.4 TinyTorrents 13

3 PROPOSED ALGORITHM 14

3.1 Definitions 14

3.2 Algorithm 18

3.2.1 Normal Mode 18

3.2.2 Leecher Mode 18

3.2.3 Endgame Mode 20

3.2.4 Seeder Mode 21

3.2.5 Periodic Tasks 21

3.3 Differences between our algorithm and BitTorrent 22

3.3.1 Peer list 22

3.3.2 Choking Algorithm 22

3.3.3 Integrity Verification 22

4 EVALUATION AND DISCUSSION 23

4.1 Evaluation Setup 23

4.2 Result and Discussion 24

4.2.1 Comparison with multicast 25

4.2.2 Trade-off 26

4.2.3 Intermittent Failures 29

4.2.4 Communication Overhead 31

4.2.5 Code size 32

4.2.6 Implementation Challenges 32

5 CONCLUSION 33

REFERENCES 34

VITA 37

List of figures

Page

Figure 1 6LoWPAN Logo 3

Figure 2 An RPL network creating its topology 4

Figure 3 6LoWPAN over IEEE 802.15.4 protocol stack.....6

Figure 4 BitTorrent Logo 7

Figure 5 Graphs showing packet delivery ratio of multicast forwarding with MPL at increasing distance from source with different sending interval [11] 9

Figure 6 A flowchart showing operation of SMRF 10

Figure 7 Graph showing transmissions end-to-end delay and packet delivery ratio comparison of multicast forwarding with MPL (TM) and SMRF using different configurations over ContikiMAC 11

Figure 8 A flowchart showing operation of ESMRF 12

Figure 9 Graph showing comparison of Average Network Delay and Network Delivery Ratio in random topologies between ESMRF, MPL (TM), and SMRF 12

Figure 10 An overview of a TinyTorrents deployment 13

Figure 11 Demonstration of communication between node i and node j 16

Figure 12 Transition between operating modes.....18

Figure 13 Screenshot of Cooja Network Simulator running a simulation of our algorithm on a 30-node random topology 24

Figure 14 Code size comparison between Blink, Ping, and Surge applications which are compiled into different formats [16].....24

Figure 15 Graph showing comparison of average time used in the disseminations using our algorithm, ESMRF, and MPL as number of nodes increasing 25

Figure 16 Graph showing comparison of average power consumption per node in the disseminations using our algorithm, ESMRF, and MPL as number of nodes increasing.....26

Figure 17 Graph showing comparison of average time used to in the disseminations with different T_s values as number of nodes increasing 27

Figure 18 Graph showing comparison of average power consumed per node in the disseminations with different T_s values as number of nodes increasing 28

Figure 19 Graph showing trendlines of average early off time percentage 29

Figure 20 Graph showing comparison of average time used in the disseminations using our algorithm with multiple rates of intermittent faults as number of nodes increasing.....30

Figure 21 Graph showing comparison of average power consumption per node in the disseminations using our algorithm with multiple rates of intermittent faults as number of nodes increasing 30

1 INTRODUCTION

1.1 Motivations

Firmware updating is essential for any devices such as personal computers, smartphones, and even small devices in wireless sensor networks. It always aims to make the devices work at their best by adding new features, improving performance, replacing outdated components, and patching vulnerabilities. The update process on personal computers or smartphones may be relatively easy because the update can be downloaded directly from the Internet but it may be more complicated on deployed devices in wireless sensor networks since they may unable to download the update directly from the internet.

Data Dissemination is a core component of firmware updating in wireless sensor networks. Since the architecture of wireless sensor networks is distributed, the dissemination will be forwarding the data from one node to another until all nodes receive it. This behavior allows it to be categorized as peer-to-peer communication. Deluge [1] and MNP [2] was data dissemination protocols in wireless sensor networks in the early days before standardized network protocols such as 6LoWPAN [3] or LoRA [4] become more popular. These standardized protocols provide multicast communication methods for data dissemination out of the box, but they do not guarantee packet delivery while firmware updating requires reliable communication. Also, different firmware updates have different purposes and priorities, for instance, an update that adds an optional feature does not have to be applied as fast as an update that patches severe vulnerabilities. Thus, enabling energy-speed trade-off.

Instead of disseminating the data with multicast communication methods, one of the most popular data sharing protocols over a peer-to-peer networks is BitTorrent [5]. It distributes each part of the data over the network and make it available to be downloaded from multiple sources, thus enabling parallel downloading. This leads to faster downloading and higher scalability compared to traditional multicast communication methods.

This research introduces a decentralized energy-speed trade-off configurable peer-to-peer data dissemination algorithm for wireless sensor networks that adopted several features from BitTorrent for being highly scalable and efficient such as Handshaking and Have Messages for exchanging possession information among nodes, Segmented File Transfer to enable parallel

downloading, Choking Algorithm to limit resources usage while uploading, Optimistic Unchoking to improve data distribution in the network, and Endgame mode to accelerate the download process.

1.2 Objectives

To develop a reliable peer-to-peer protocol for software updating in distributed wireless sensor networks which enables software updating in different purposes and priorities

1.3 Scope of Work

1. This research developed using 6LoWPAN over IEEE 802.15.4 protocol on Contiki-NG
2. This research developed using Storing Mode of RPL Routing Protocol
3. All nodes in this research are Zolertia Z1 boards
4. This research simulated using Cooja Network Simulator

1.4 Research Procedure

1. Study of related works
2. Analysis of research problems and possible solutions
3. Development of the proposed solution
4. Experiment and evaluation
5. Result Conclusion
6. Writing of research papers and thesis

1.5 Expected Benefits

A reliable peer-to-peer protocol for software updating in distributed wireless sensor networks which enables software updating in different purposes and priorities

1.6 Published Paper

High Performance Peer-to-peer Data Dissemination for Decentralized Wireless Sensor Networks Firmware Updating. Published at 2020 2nd International Electronics Communication Conference (IECC). Held on 8-10 July 2020 in NTU@one-north Executive Centre, Singapore.

2 TECHNICAL BACKGROUNDS AND RELATED WORKS

In the section, we are going to discuss about technical backgrounds of the technologies used in this research including 6LoWPAN, RPL, and BitTorrent. We will also discuss about multicast communication modes in 6LoWPAN as they are mechanisms to disseminate the data over RPL networks which are related to our work.

2.1 Technical Backgrounds6LoWPAN

Figure 1 6LoWPAN Logo

6LoWPAN (IPv6 over Low-Power Wireless Personal Area Network) (RFC4919) [3] is a protocol for adapting IPv6 packets to be able to be sent wirelessly over IEEE 802.15.4. The devices in a 6LoWPAN network will be assigned IPv6 addresses to enable them accessible to each other and to the Internet.

6LoWPAN is an open standard by Internet Engineering Task Force (IETF) initially defined as a protocol for sending packets wirelessly over IEEE 802.15.4 at 2.4GHz but later has been improved to be able to work with sub-1GHz protocols such as Bluetooth Smart, Power Line Control (PLC), and Low-Power Wi-Fi.

To adapt IPv6 packets to be able to be sent over IEEE 802.15.4, 6LoWPAN will perform 3 tasks as follows.

1. Header compression

8-byte UDP Header and 40-byte IPv6 Header will be compressed by removing redundant information to reduce the packet size. Even 6LoWPAN also supports TCP protocol, but TCP Headers will not be compressed.

2. Fragmentation and reassembly

IEEE 802.15.4 allows maximum 127-byte frame size for each transmission but Maximum Transmission Unit (MTU) of IPv6 is 1280 bytes. Thus, requiring packet fragmentation before sending and reassembly on receiving.

3. Stateless auto configuration

A device in a 6LoWPAN network will create an IPv6 address for itself without waiting for an assignment from its router. There is also a mechanic that prevents 6LoWPAN devices from creating duplicates IPv6 addresses called Duplicate Address Detection (DAD).

6LoWPAN supports AES-128 encryption over IEEE 802.15.4 and supports Transport Layer Security (TLS) and Data Transport Layer Security (DTLS) over TCP and UDP, respectively. However, TLS and DTLS uses intensive computation so it may require microcontrollers that supports hardware-accelerated encryption.

2.1.2 IPv6 Routing Protocol for LLNs (RPL)

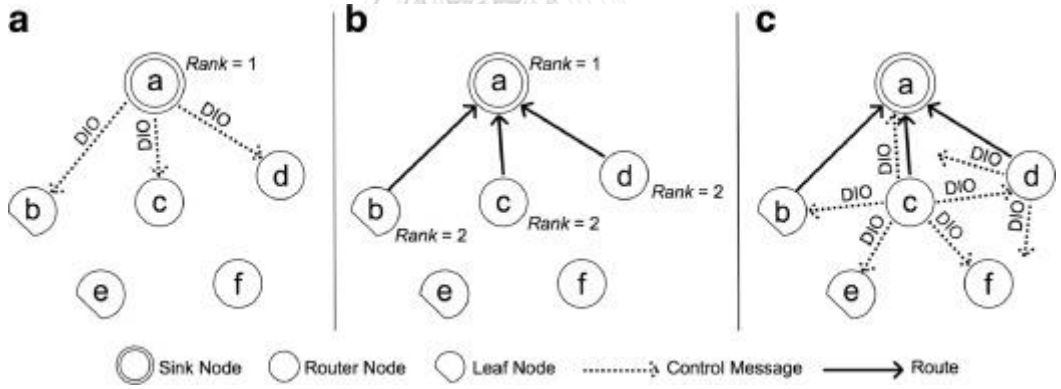

Figure 2 An RPL network creating its topology

Reference: https://doi.org/10.1016/j.comnet.2016.03.018

RPL (IPv6 Routing Protocol for Low-Power and Lossy Networks) (RFC6550) [6] is a routing protocol for wireless networks that consist of low-power devices and lossy communications, optimized for sending data to one mutual sink.

RPL will create a topology of a network in Directed Acyclic Graph (DAG) which contains states of nodes in the network and costs of accessing each node. The states of each node may contain its workload, residual energy, latency, or reliability which will be calculated as its cost for forwarding a packet through it.

There can be multiple sinks in an RPL network. For routing to each sink, Destination Oriented Directed Acyclic Graph (DODAG) will be created from the DAG. A DODAG will contain RPLInstanceID to identify which RPL network it belongs to, DODAGID to identify to DODAG itself, DODAGVersionNumber to control the version of the DODAG, and ranks for each node in the DODAG to perform routing. For updating the information of a DODAG, each node will broadcast a DODAG Information Option (DIO) message to exchange information of the DODAG among themselves while the communication flow is controlled by Trickle Algorithm [7] to prevent redundant transmissions. The root of the network will be the first to broadcast DIO message and set the rank of itself to be 1. When a node received DIO message, it will update its rank corresponding to the received information and select its parent node which has lower rank to forward its packets to. A node can send DODAG Information Solicitation (DIS) to request a DIO message and Destination Advertisement Object (DAO) to its parent to request DODAG rebuild.

RPL supports 2 modes of operation: Non-Storing and Storing. In Non-Storing mode, each node will be able to send packets only to the root of the network. If a node in Non-Storing mode wants to send a packet to another node, the packet must be forwarded by the root. In Storing mode, each node will have its own routing table to store routing information to other nodes nearby. Thus, enabling them to be able to communicate with each other without sending to the root first but requiring more memory usage.

2.1.3 IEEE 802.15.4

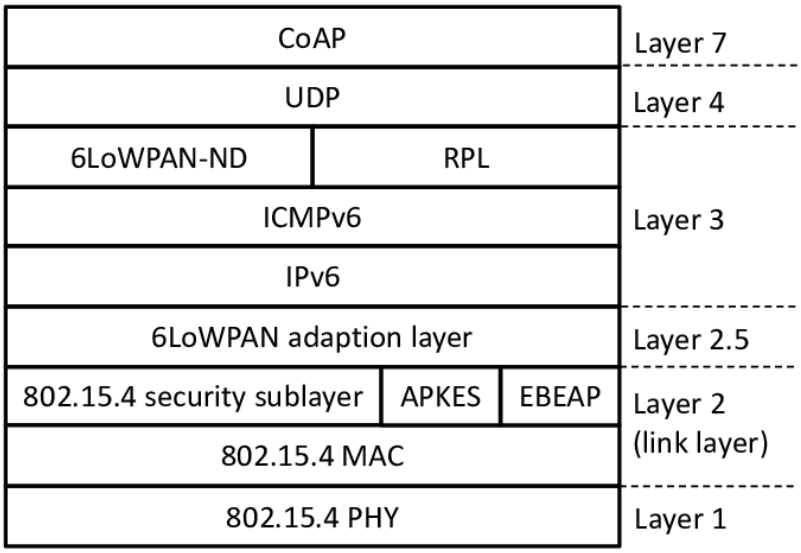

Figure 3 6LoWPAN over IEEE 802.15.4 protocol stack

Reference: https://www.doi.org/10.1145/2523501.2523502

IEEE 802.15.4 [8] is a standard for low-rate wireless personal area networks (LR-WPANs) containing standard for physical layer and medium access control for LR-WPANs. IEEE 802.15.4 is developed and maintained by IEEE 802.15 working group. It is the basis of ZigBee, 6LoWPAN, and Thread protocol.

IEEE 802.15.4 is designed to provide low-data rate and low-power communication to embedded devices. It offers 250 kbit/s transfer rate at 10-meter communication, but it can be configured to 20, 40, and 100 kbit/s rate for lower power consumption.

In the physical layer (PHY), IEEE 802.15.4 defines how RF (Radio frequency) transceiver perform signal transmissions and receptions. It can operate on one of three unlicensed frequency bands in each region including 868.0-868.6 MHz in Europe, 902-928 MHz in North America, and 2400-2483.5 MHz worldwide.

In the medium access control layer (MAC), IEEE 802.15.4 manages access to the physical channel, validate data frames and guarantees time slots.

There is no standard defined for higher layers in IEEE 802.15.4 but there are many other specifications that built on top of it including ZigBee or 6LoWPAN which are supported by many operating systems such as RIOT OS, TinyOS, Contiki-OS, and Zephyr OS.

2.1.4 BitTorrent

Figure 4 BitTorrent Logo

BitTorrent [5] is a peer-to-peer data sharing protocol. Instead of downloading data from a single server, BitTorrent allows users to have parts of the data and share among themselves. This provides scalability to the system. Thus, the increasing number of users will not reduce the performance but may improve it instead. Several features of BitTorrent which we adopted for this research are described below.

1. Handshaking and Have Messages

In order for peer A to know which pieces can be requested from peer B, peer A and peer B will exchange handshakes and Have messages containing Bitfield of their possession information.

2. Segmented File Transfer

The data that is to be shared in a BitTorrent will be divided into pieces. These pieces will be distributed over the network to ensure high availability and to make each peer to be able to download multiple pieces from multiple sources simultaneously as known as parallel downloading.

3. Rarest First Piece Selection

A peer will select and download the rarest visible pieces to improve data redundancy and reduce the chance of some pieces missing from the network.

4. Choking

A peer chokes another peer to refuse to upload to it and unchokes to start upload to it again. This is to prevent a peer to upload to too many peers at the same time, a peer will unchoke and upload to its top uploaders in return for their previous uploads as tit-for-tat.

5. Endgame Mode

When a peer is only few pieces away from completion, it will request for the missing pieces from multiple sources ignoring the fact that the pieces may currently be downloading. This is to quickly turn itself to a *seeder* and become more efficient uploader.

6. Optimistic Unchoking

A newly connected peer will not have anything to upload. Thus, will not able to download anything because of the choking addressed above. For optimistic unchoking, a peer will randomly unchoke one peer that want a piece from it and give the peer the piece it wants ignoring its upload rate.

2.2 Related Works

2.2.1 IPv6 Multicast Forwarding in RPL-Based Wireless Sensor Networks (MPL)

MPL (RFC7731) [9] is a multicast communication protocol designed to enable multicast communication for IPv6 without creating a special topology. MPL supports 2 modes: 1. using Trickle Algorithm [7] to control the flow of communication and 2. using classic flooding.

A research [10] states that using Trickle Algorithm to control the flow of communication has lower rate of packet loss in lossy networks which is more suitable for wireless sensor networks that using classic flooding.

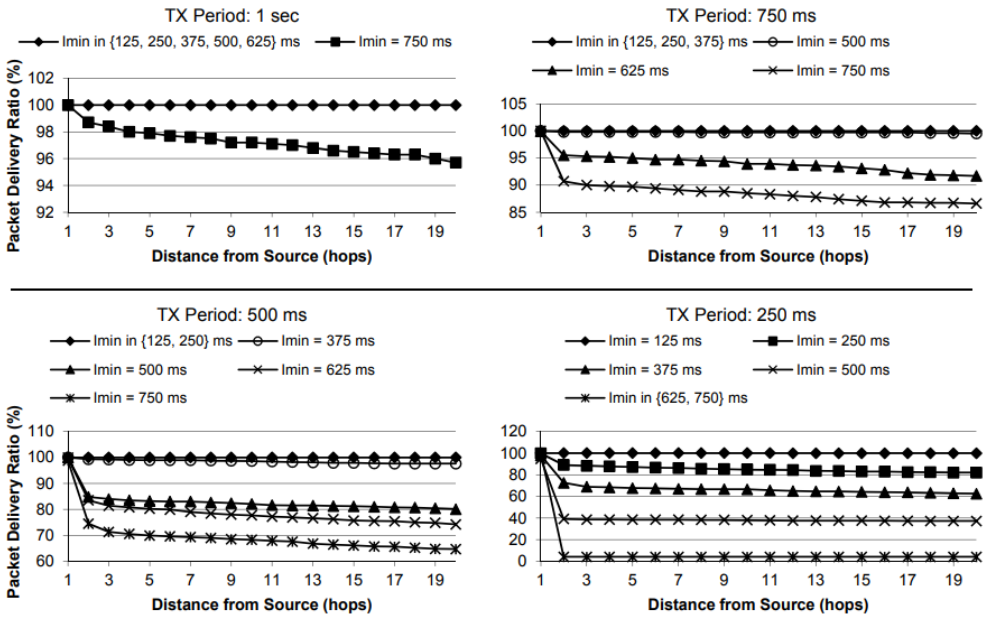

Figure 5 Graphs showing packet delivery ratio of multicast forwarding with MPL at increasing distance from source with different sending interval [11]

2.2.2 Stateless Multicast RPL Forwarding (SMRF)

SMRF [11] is a multicast communication protocol designed to overcome disadvantages of MPL as follows.

1. **Scalability** While MPL does not create a topology to reduce memory usage, MPL has to create custom packets for each node to control the communication state. So there will be several factors that affects its ability to scale such as communication traffic and cache size.
2. **Performance** To prevent redundant communication, MPL will not immediately forward packets. Instead, MPL will cache them and wait for Trickle Algorithm to trigger sending. Thus, causing delay in communication.
3. **Complexity** In MPL, each node will have to maintain 2 Trickle Timers and a Sliding Window for each multicast source and a cache for the most recent multicast packet. MPL also has its own control packets which increase complexity and size of the code and also increase memory usage.

4. **Multicast versus Broadcast** Without using topology, MPL multicast packets have to be sent through the entire network even to the nodes that do not want them causing high unnecessary traffic and power consumption.
5. **Arrival Order** Using custom packets and caching may cause incorrect packet arrival order which is not suitable for some applications such as Network Monitoring System, in which case the packet arrival order verification has to be implemented separately.

SMRF utilizes DODAG information to overcome those downsides. A node that wants to join SMRF multicast group will announce to its parent node using DAO message and a parent node will forward the multicast packets to its children. However, the limitations of RPL are 1. no prevention mechanism for redundant packet reception and 2. nodes have to be sibling to be able to send multicast to each other or they have to send multiple same unicast packets multiple times instead which uses more memory and cause network delay.

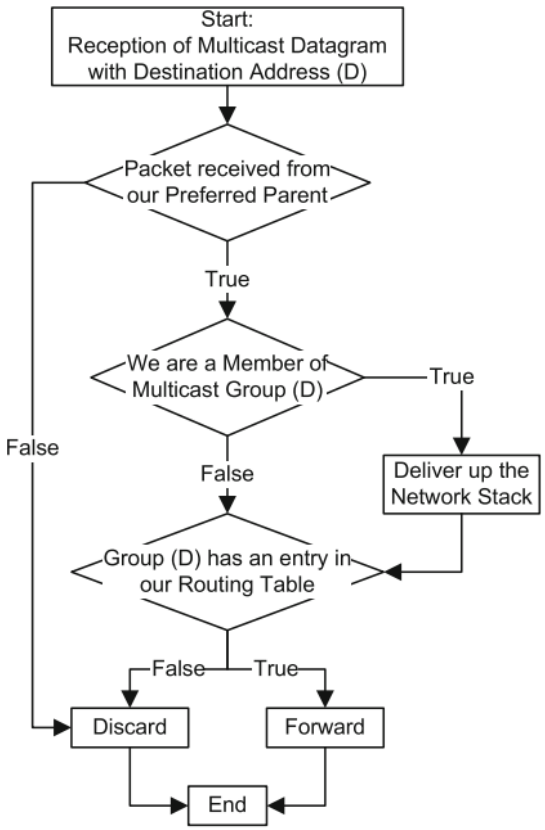

Figure 6 A flowchart showing operation of SMRF

An experiment shows that not using Trickle Algorithm makes SMRF less reliable than MPL. SMRF has more packet loss rate but has higher communication speed and correct order of arrival than MPL. Also, MPL code is smaller and more power efficient. [10]

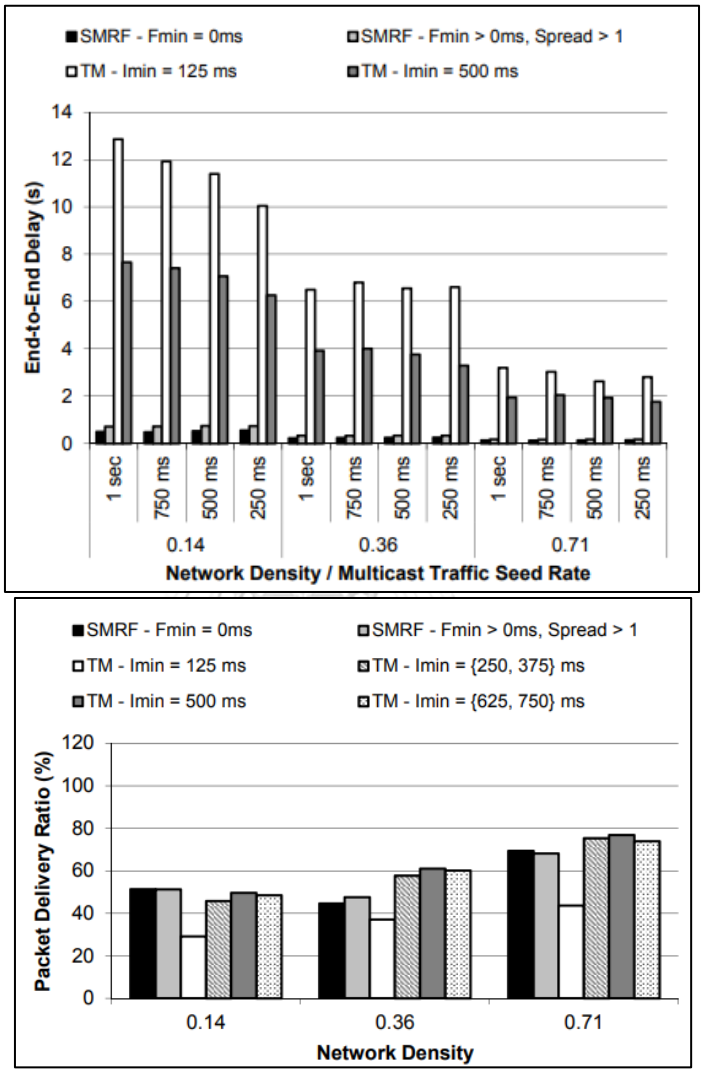

Figure 7 Graph showing transmissions end-to-end delay and packet delivery ratio comparison of multicast forwarding with MPL (TM) and SMRF using different configurations over ContikiMAC

2.2.3 Enhanced Stateless Multicast RPL Forwarding (ESMRF)

ESMRF [12] is a multicast communication protocol developed from SMRF by solving the important downside of SMRF that is not able to send multicast packets upward. SMRF can only send multicast packets from parents to their children but not vice versa. ESMRF overcomes this limitation by sending the packet to the root first and since the root is the parent of all nodes so

ESMRF will be able to send multicast packets in both upward and downward while having the same performance of SMRF in linear topologies and having better performance in random topologies.

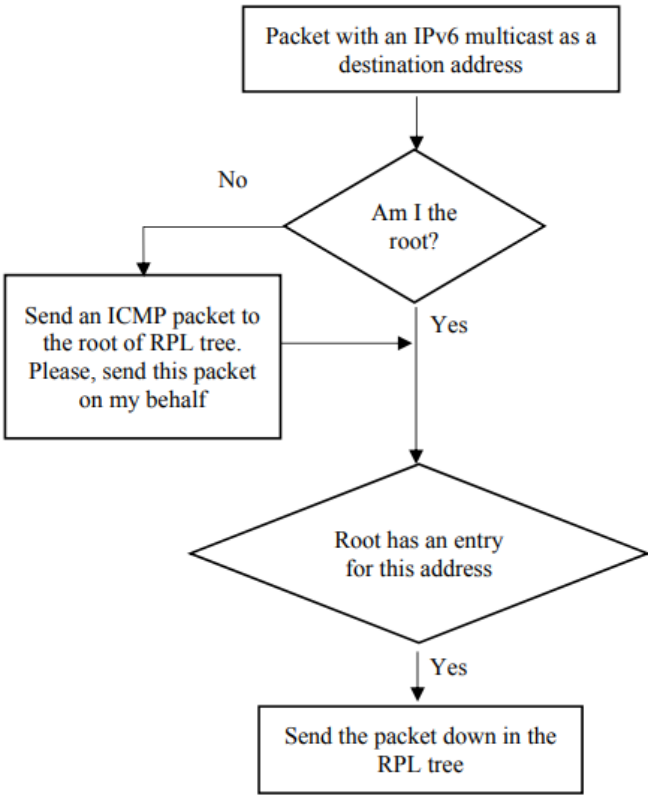

Figure 8 A flowchart showing operation of ESMRF

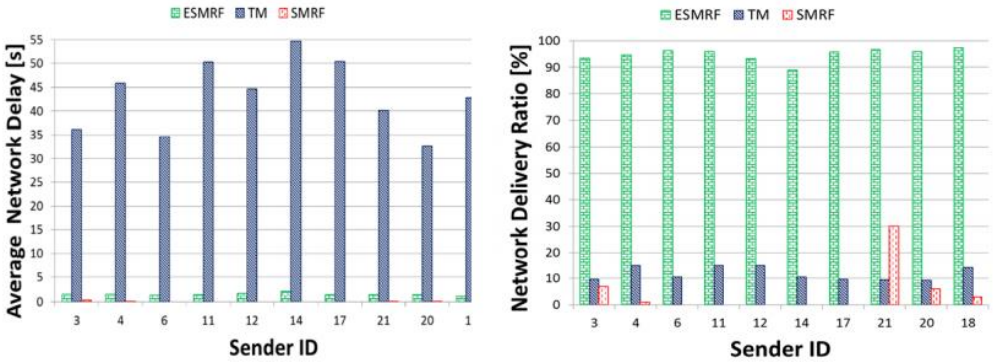

Figure 9 Graph showing comparison of Average Network Delay and Network Delivery Ratio in random topologies between ESMRF, MPL (TM), and SMRF

2.2.4 TinyTorrents

There is also a work [13] called *TinyTorrents* in 2009 which was an attempt to bring BitTorrent to wireless sensor networks. The work aims to pushing the sensed data into an existing BitTorrent network by forwarding its metadata to the root node and a PC which are physically connected and together work as a border router exposed the metadata to the Internet. As the work's intention was not to disseminate the data throughout the network but to get the data ready for retrieving from the Internet, so we did not compare our performance to this work.

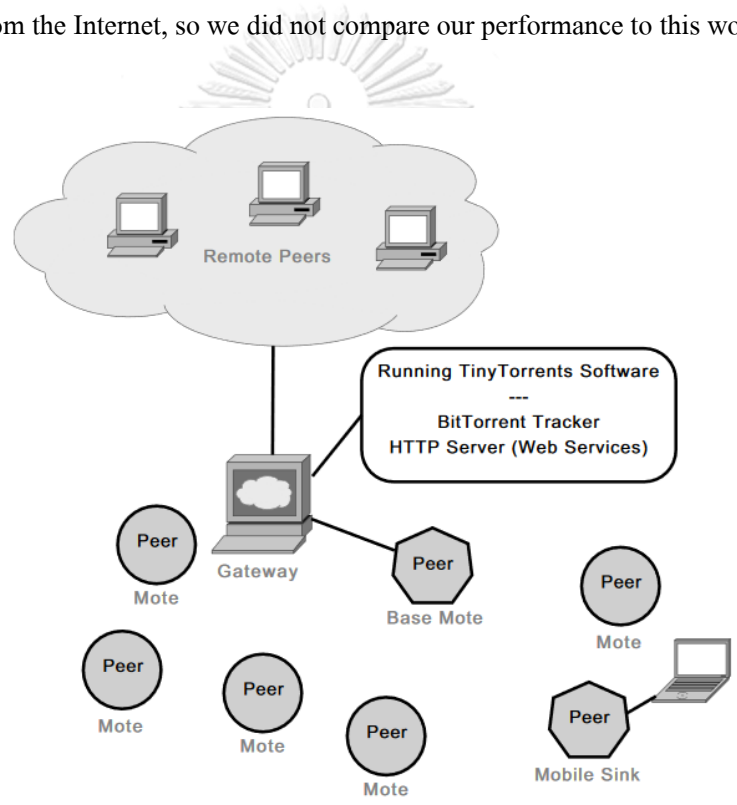

Figure 10 An overview of a TinyTorrents deployment

3 PROPOSED ALGORITHM

3.1 Definitions

We define our wireless sensor network is a network consists of N nodes with id from 0 to N-1. As nodes are distributed, we assume that each node can communicate only to its neighbors. For node i, let Neighbors(i) be a set of neighbors of node i that node i can communicate directly. Due to limited memory, let assume that node i can communicate to at most M nodes simultaneously. Note that the number of nodes in Neighbors(i) can be more or less than M.

Let node 0 is the root node which has the disseminating data of S bytes. The disseminating data is divided into fixed number of X pieces. The size of each piece is S/X bytes. A node is considered to be a seeder if it has all X pieces. A seeder always sends up-to M nodes that request for pieces at the same time. A node is considered to be a leecher if it still does not have all pieces. A leecher always requests for missing pieces from other nodes (up-to M_d nodes simultaneously). It also supplies pieces to up-to M_u nodes that request for the pieces that it has. Obviously, $M_u + M_d <= M$.

For a node requesting or supplying pieces, it uses 5 control messages to control the flow of its communications with other nodes:

1. HANDSHAKE for a node to send its information about how many and which pieces it is possessing and whom the node has downloaded from
2. ACKHANDSHAKE for a node to answer back its information possession information to a HANDSHAKE
3. INTEREST for a node to inform that it is interested in downloading a piece
4. CHOKE/UNCHOKE for a node to tell if it allows the node that is interested to download from it
5. REQUEST for a node to start downloading

Node i keeps track of the communication state for each node in Neighbors(i). Let node j is a node in Neighbors(i) and let State(i, j) is the communication state between node i and node j. The possible values of State(i, j) is described in Table 1.

State(i, j)	Description
IDLE	node i has not handshaked with node j
HANDSHAKING	node i sent HANDSHAKE to node j and waiting for ACKHANDSHAKE from node j
HANDSHAKED	node i has received ACKHANDSHAKE from node j
INTEREST_INFORMING	node i sent INTEREST to node j and waiting for CHOKE or UNCHOKE from node j
INTEREST_INFORMED	node i has received CHOKE or UNCHOKE from node j
DOWNLOADING	node i is downloading a piece from node j
UPLOADING	node i is uploading a piece to node j

Table 1 Table showing the possible values of State(i, j)

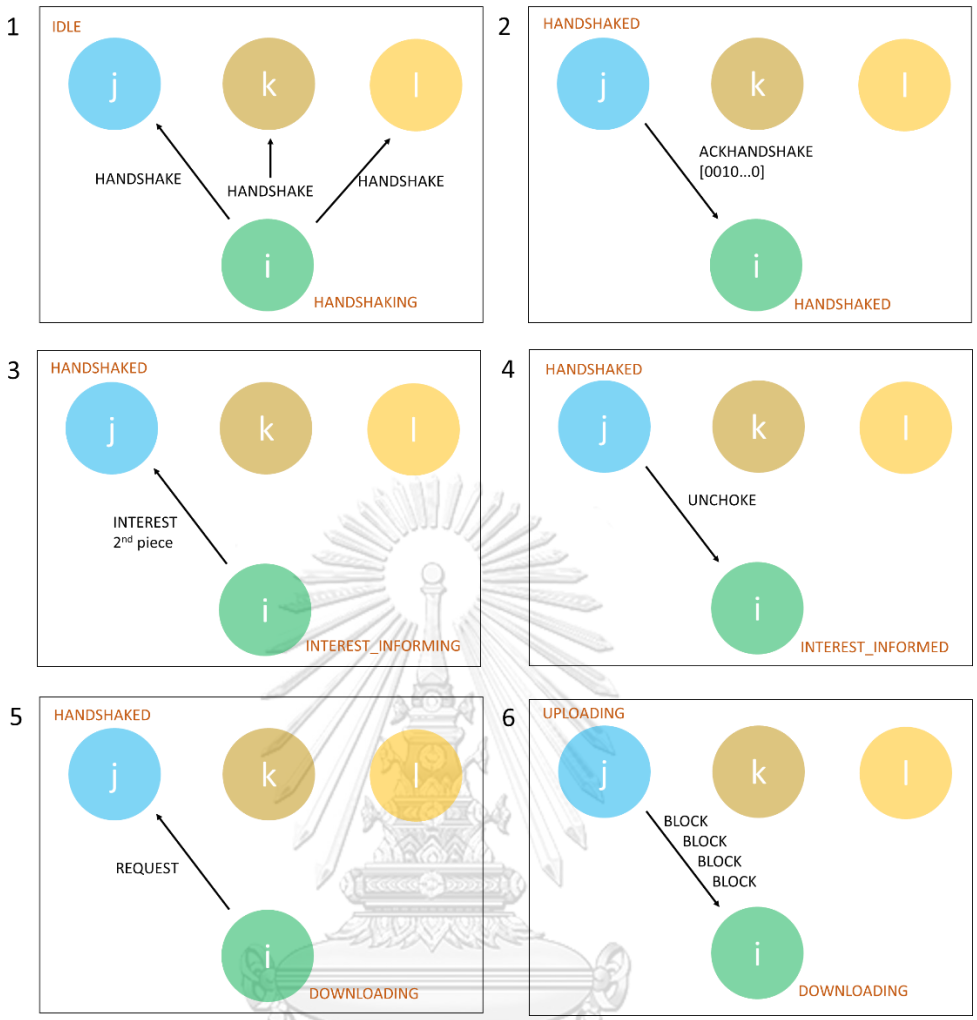

Figure 11 Demonstration of communication between node i and node j

In addition to State(i, j), node i also keeps track of Interest(i, j) indicates if node i is interesting in node j, Choke(i, j) indicates if node i is choking node j, and NumUL(i, j) indicates number of blocks node i has uploaded to node j

To support different priority levels of software updates, this algorithm can be configured to disseminate at different speed and different power consumption. For high priority software updates such as patching vulnerabilities, the disseminations should be as fast as possible even causing high power consumption; on the contrary, for low priority software updates such as adding an optional feature, the disseminations can be slower while keeping low power consumption.

The configuration can be done by setting a variable called T_s. At the beginning of a dissemination process, it is likely for node i that every node in Neighbors(i) does not have any pieces so node i will sleep and wake periodically every 5 seconds for a duration of T_s seconds.

So, If $T_s = 0$, the algorithm will assume that the dissemination needs to be as fast as possible. But if $T_s > 0$, the algorithm will assume that the dissemination is focusing on reducing power consumption more that increasing speed will enable features called *Late On* and *Early Off* therefore which will be described in section 3.2.

3.2 Algorithm

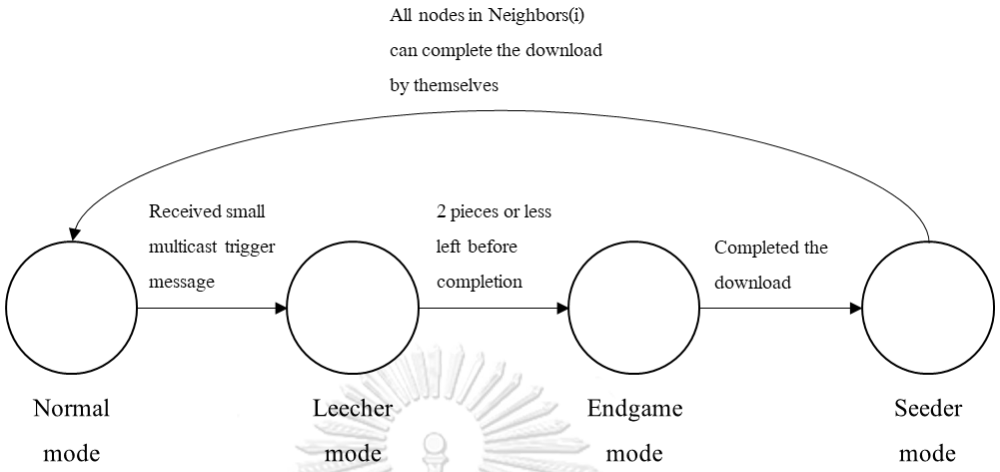

Figure 12 Transition between operating modes

A node can operate in one of 4 modes: Normal Mode, Leecher Mode, Endgame Mode and Seeder Mode. These modes and their operations will be described in this section.

3.2.1 Normal Mode

All nodes start in normal mode in which each node operates normally as a sensor or an actuator, it will switch to leecher mode when it is informed that there is a new firmware update. The root node sends a trigger packet to all nodes using reliable multicast communication [14]. As a trigger packet is small, a standard protocol is effective enough.

3.2.2 Leecher Mode

In leecher mode, node i finds pieces from Neighbors(i). Initially, node i sets State(i, x) = IDLE for each node x in Neighbors(i). The process of node i requesting a piece is described below.

1. Node i sends HANDSHAKE to each node x in Neighbors(i) and sets State(i, x) = HANDSHAKING
2. On receiving ACKHANDSHAKE from a node in Neighbors(i) and let node j be that node, node i sets State(i, j) = HANDSHAKED

3. If all nodes in Neighbors(i) do not have any pieces and $T_s > 0$, node i will sleep for T_s seconds. Otherwise, node i continues handshaking. Note that Ts is the variable that we used to vary among different priority levels of software updates. This is a feature called *Late On* to delay download operation in exchange for lower power consumption
4. If number of nodes being downloaded by node i < M_d, node i will pick a piece P_x which it does not have and has only 1 owner in Neighbors(i) assuming to be the rarest piece, then pick the first node in Neighbors(i) list that has the piece P_x. Suppose that node j is picked, node i sets Interest(i, j) = TRUE
5. Node i sends INTEREST to node j and sets State(i, j) = INTEREST_INFORMING
6. Node i expects for CHOKE or UNCHOKE
    a. If CHOKE received, node i sets State(i, j) = HANDSHAKED and Interest(i, j) = FALSE. After that, node i will stop sending messages to node j for 5 seconds.
    b. If UNCHOKE received, node i sets State(i, j) = INTEREST_INFORMED
7. After 6.b, node i sends REQUEST to node j to start downloading and sets State(i, j) = DOWNLOADING and waits for blocks of the piece P_x
8. Node i will receive Y blocks from B_0 to B_{Y-1} consecutively. For each block received, node i will verify its integrity using CRC16 attached in the message
9. After node i has received the last blocks B_{Y-1} of the piece P_x without problems, node i sets State(i, j) = HANDSHAKED and Interest(i, j) = FALSE

Node i will keep requesting for its missing pieces from not more than M_d nodes at the same time while it also providing its pieces to other nodes that request for them not more than M_u nodes at the same time.

To overcome unreliable communication, node i has recovery procedure when it does not receive a response from node j within 3 seconds. The procedures of the recovery are as follows:

If State(i, j) is HANDSHAKING then set State(i, j) = IDLE. This is when node i has sent HANDSHAKE but does not receive ACKHANDSHAKE in time, node i will set State(i, j) to IDLE to resend HANDSHAKE to node j in the next iteration.

If State(i, j) is INTEREST_INFORMING then set State(i, j) = HANDSHAKED and Interest(i, j) = FALSE. This is when node i has sent INTEREST to node j but does not receive CHOKE or UNCHOKE in time, node i will set Interest(i, j) to FALSE to find a new piece and a new node to request for.

If State(i, j) is DOWNLOADING, node i will stop downloading from node j then set State(i, j) = HANDSHAKED. This is when node i is downloading from node j but does not receive all blocks in time, node i will set State(i, j) = HANDSHAKED to stop downloading.

Node i maintains a list of nodes that is unresponsive to it by keeping the list in Blacklist(i). If node j has not responded to node i for 5 consecutive messages, node i will assume that there may be some issues in the communication between itself and node j so node i will remove node j from Neighbors(i) and push node j to Blacklist(i). To select a node to request from, node i will iterate from the head of Neighbors(i) list. Thus, moving node j to the end of the list making it unlikely to be selected again.

While possessing pieces, node i will receive requests for the pieces from node j. To supply these pieces, node i performs as follows.

On receiving HANDSHAKE from a node j, node i will send ACKHANDSHAKE to node j

On receiving INTEREST from node j, node i will consider if it should upload to node j. If node i finds itself already uploading to M_u nodes at the time or node j has been choked from optimistic unchoking, node i will send CHOKE to node j and set Choke(i, j) = FALSE to stop choking node j in the next request. Otherwise, node i will send UNCHOKE to node j and set Choke(i, j) = FALSE

On receiving REQUEST from node j, node i will set State(i, j) = UPLOADING to prevent requesting from node j while uploading to it. Then divide the requested piece into Y blocks from B_0 to B_{Y-1} and send them along with their CRC16 hashed value to node j consecutively. For each block sent, NumUL(i, j) += 1

3.2.3 Endgame Mode

When node i is 2 or less pieces away from completion, node i will switch to endgame mode. In this mode, node i will request for its missing pieces from Neighbors(i) as in leecher mode but it will request for the same piece from multiple sources ignoring the fact that the piece

may currently be downloading at that time. This is to quickly turn itself to be a seeder and enter seeder mode and also to overcome stuck or inactive connections.

3.2.4 Seeder Mode

After node i completed downloading all X pieces, it will switch to seeder mode. In this mode, node i will stop sending out HANDSHAKE, INTEREST, or REQUEST since it does not need to request for any pieces anymore. But it will still respond to other nodes that request for its pieces. Node i will now change its $M_d = 0$ and $M_u = M$ to upload to more nodes simultaneously. As node i is still receiving HANDSHAKE, it still has knowledge of how many and which pieces the nodes in Neighbors(i) are possessing and whom the nodes have downloaded from which are likely to also be in Neighbors(i). So, if $T_s > 0$, node i will determine if all nodes in Neighbors(i) are able to complete all X pieces without node i itself. If all nodes in Neighbors(i) can complete the download by themselves, node i will stop uploading and turn back to Normal Mode to reduce overall power consumption used for software updating. This is a feature called *Early Off* to stop uploading earlier that may delay the overall download process but significantly reduce power consumption.

3.2.5 Periodic Tasks

Along with the communication, there are 2 tasks node i is performing periodically:

1. Constructing Neighbors(i) – node i will fetch the list of neighbors from the networking stack of the operating system every 7 seconds, node i will add these nodes to Neighbors(i) if they are not existing then assign default values for State(i, j), Interest(i, j), Choke(i, j) to each new node j. Note that if the new node is the old one that was once removed from Neighbors(i) but exists in Blacklist(i) due to unresponsiveness, node i will ignore it in adding it back to Neighbors(i) this time but only remove it from Blacklist(i) which makes them wait for another 7 seconds to be able to be added back.
2. Optimistic unchoking – node i will choke the node that has downloaded from it the most to prevent uploading too many pieces to that node. Thus, improving data distribution throughout the network. Let node j be the node that has downloaded from node i the most (max NumUL()), node i will set Choke(i, j) = TRUE to deny

the next request from node j and randomly pick a new node k when Choke(i, k) is TRUE then set Choke(i, k) = FALSE

3.3 Differences between our algorithm and BitTorrent

Although we derived some ideas from BitTorrent protocol to implement our algorithm, but not all of them have the same intentions as they do in original BitTorrent.

3.3.1 Peer list

In original BitTorrent which is operating on the Internet, a network can be relatively large and can be composed of hundreds or thousands of peers. So, the mechanism of keeping the list of peers is to distribute it over the network using Distributed Hash Table (DHT). But in our algorithm which disseminates the data in wireless sensor networks, it would be more efficient to forwarding the data to neighbor nodes. So, the list of peers will be the list of neighbor nodes provided by the operating system.

3.3.2 Choking Algorithm

Choking Algorithm was initially created in order to provide more data to good uploaders in the network which will improve overall performance of the dissemination process. This is because a BitTorrent network operates on the Internet and its peers can be anywhere in the world so there are many factors that some peers are good uploaders and some aren't. But in our algorithm which is for disseminating important data as software updates, nodes in a network are physically close to each other and equally in need of the same data, so the only reason to choke others is incapability to serve the data to them.

3.3.3 Integrity Verification

A BitTorrent network would store the metadata of its data in DHT so after one of its peers downloaded a piece, the peer will verify the integrity of the piece by comparing the SHA-1 hash of the piece to the metadata. But in our algorithm, instead of storing in metadata, the hash will be attached to each BLOCK message sent to the destination. And instead of using CPU intensive SHA-1 hash, our algorithm uses CRC16 for hashing which is more suitable for microcontrollers.

4 EVALUATION AND DISCUSSION

4.1 Evaluation Setup

The implementation of our algorithm was developed using Contiki-NG operating system. We implemented the algorithm for 6LoWPAN using RPL storing mode since it has been used by many commercial deployments for its being decentralized which offers higher scalability for large networks [15]. We assume that only application part is disseminated. The application part of a node in a wireless sensor network can be varied in size as shown in Figure 13, so we defined that the size of the disseminating data would be (S = 4096) bytes and as 6LoWPAN operates on IEEE 802.15.4 which limits its packet size to 127 bytes, so the parameter for the evaluation is configured as follows: the disseminating data will be divided into (X = 32) pieces, a piece will be divided into (Y = 4) blocks, thus making a block for each transmission will be 32 bytes. To limit resources usage, a node will upload to (M_u = 2) nodes and download from (M_d = 2) nodes at the same time.

We defined 3 priority levels of software updates: Critical, Balanced, and Minor. For Critical Updates, we set T_s value to 0 to disseminate the updates with the highest speed while allowing high power consumption. For Balanced Updates, we set T_s value to 5 to disseminate the updates with moderate speed while maintaining moderate power consumption. For Minor Updates, we set T_s value to 10 to disseminate the updates with low speed while keeping low power consumption. We evaluate the support of different priority levels of software updates by varying T_s values in 5 random 50-node topologies and evaluate the performance against different number of nodes in 3 random topologies for each 10, 30, and 60 nodes using Cooja Network Simulator. All nodes are deployed as Zoleria Z1 boards. The simulations started with loading the data onto the root node while the other nodes will start empty and the download process will start after 1 minute.

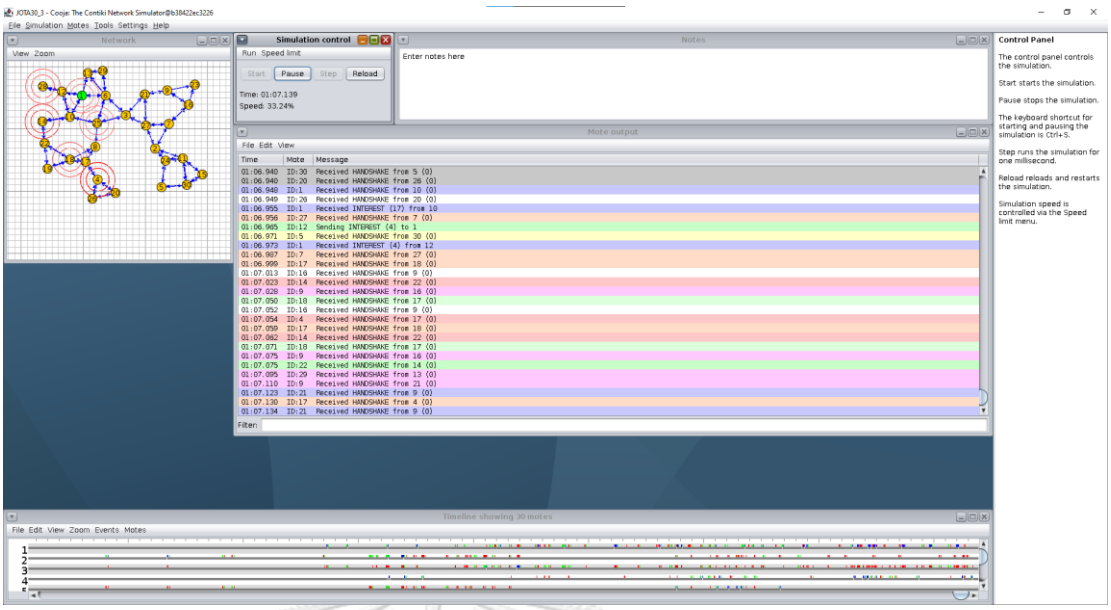

Figure 13 Screenshot of Cooja Network Simulator running a simulation of our algorithm on a 30-node random topology

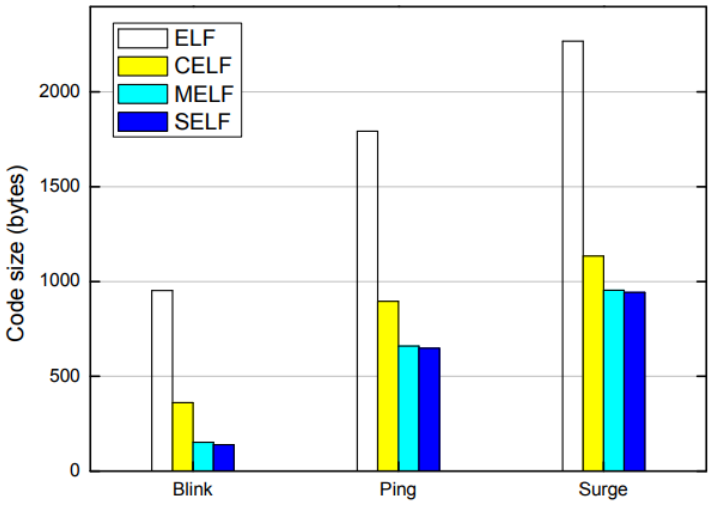

Figure 14 Code size comparison between Blink, Ping, and Surge applications which are compiled into different formats [16]

4.2 Result and Discussion

We evaluate the performance of our algorithm using two metrics: Dissemination Time and Power Consumption.

4.2.1 Comparison with multicast

We compared our algorithm to two multicast communication protocols: ESMRF and MPL. On our algorithm and ESMRF, the transmissions were sent every 100 milliseconds since they are not based on timers and able to handle fast transmissions well. Therefore, MPL is Trickle-based and it can ensure higher delivery ratio on slow transmissions [11] so its transmissions were sent every 1 second.

ESMRF and MPL multicast do not ensure packet discovery so the root node will continue sending the data sequentially. Thus, a node which missed one of the last pieces will have to wait longer than a node which missed one of the first.

Figure 15 and Figure 16 show the comparison of average time used and average power consumed in the dissemination between our algorithm, ESMRF, and MPL. In 10-node topologies, our algorithm spends less time than ESMRF and MPL for 33.60% and 39.58% and consumes less power for 50.60% and 57.97% respectively. But as increasing number of nodes, our algorithm performs even better. It spends around 40%-60% less time and consumes around 60%-80% less power than both ESMRF and MPL. The result also shows that the increasing number of nodes significantly takes both ESMRF and MPL dissemination time but not our algorithm.

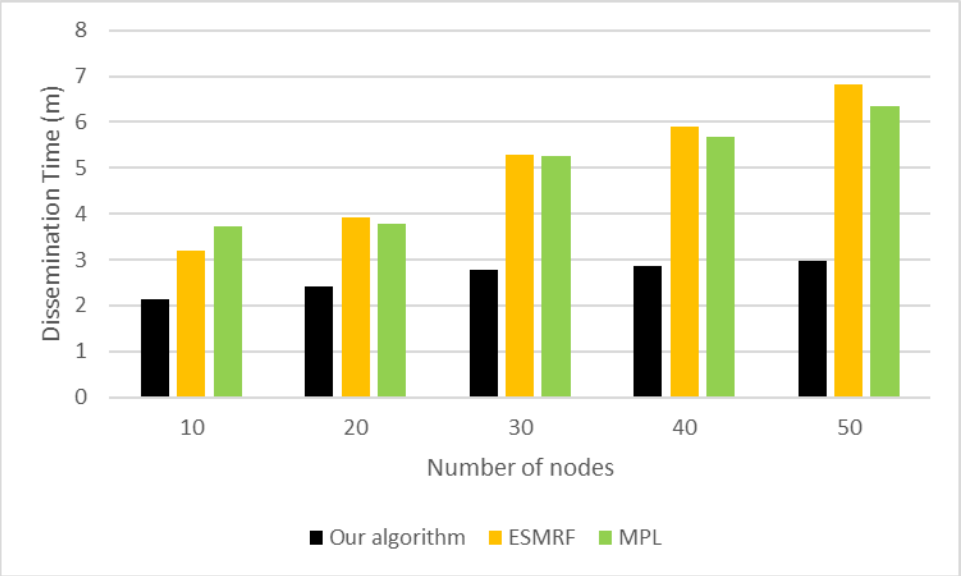

Figure 15 Graph showing comparison of average time used in the disseminations using our algorithm, ESMRF, and MPL as number of nodes increasing

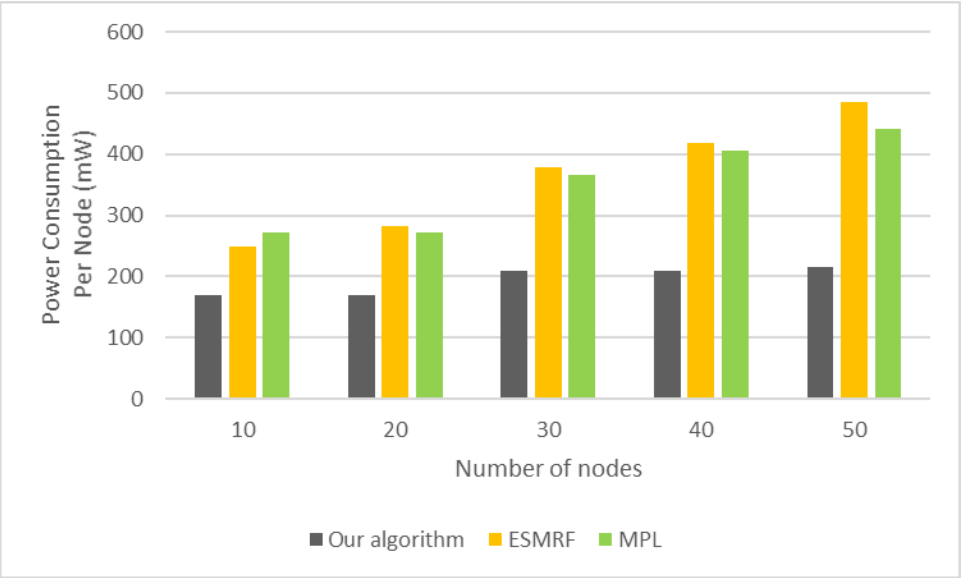

Figure 16 Graph showing comparison of average power consumption per node in the disseminations using our algorithm, ESMRF, and MPL as number of nodes increasing

The results show that our algorithm outperforms both ESMRF and MPL in term of dissemination time. ESMRF can disseminate the data with high transmission rate but also encountered high rate of packet loss. MPL can disseminate the data with 100% delivery ratio to a few first hops but the ratio dropped for the further hops and it took almost twice or three times as much time to wait for the second or third retransmission. The results also show that increasing number of nodes increases dissemination time of our algorithm with lower rate than ESMRF and MPL. So, our algorithm may be considered to be higher scalable than both ESMRF and MPL.

4.2.2 Trade-off

To evaluate the performance of disseminations for different priority levels of software updates, we used 3 T_s value: Critical Updates (T_s = 0), Balanced Updates (T_s = 5), and Minor Updates (T_s = 10).

Figure 17 and Figure 18 show the comparison of average time used and average power consumed in the dissemination between different priority levels of software updates. In 10-node topologies, different T_s values do not cause any different in power consumption because of too low number of nodes, however, the disseminations of Minor Updates (T_s = 10) take more time which may not be a worthy trade-off. In 30-node topologies, the disseminations of Critical

Updates (T_s = 0) and Balanced Updates (T_s = 5) take almost the same amount of time while the disseminations of Balanced Updates (T_s = 5) consumes less power than Critical Updates (T_s = 0) for 10.12%, also, Minor Updates (T_s = 10) take 15.71% more time but consume 7.85% less power compared to Balanced Updates (T_s = 5). In 50-node topologies and 60-node topologies, the disseminations of Balanced Updates (T_s = 5) take more time for around 5%-15% but consume less power for around 15% than the disseminations of Critical Updates (T_s = 0) while the disseminations of Minor Updates (T_s = 10) take more time for around 20-25% but consume less power for around 17-20% compared to Critical Updates (T_s = 0).

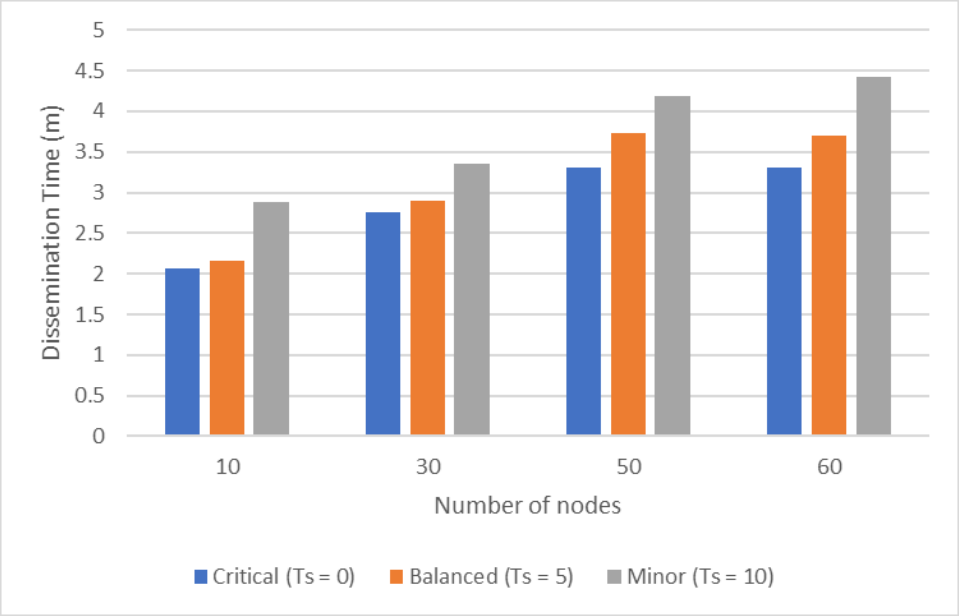

Figure 17 Graph showing comparison of average time used to in the disseminations with different T_s values as number of nodes increasing

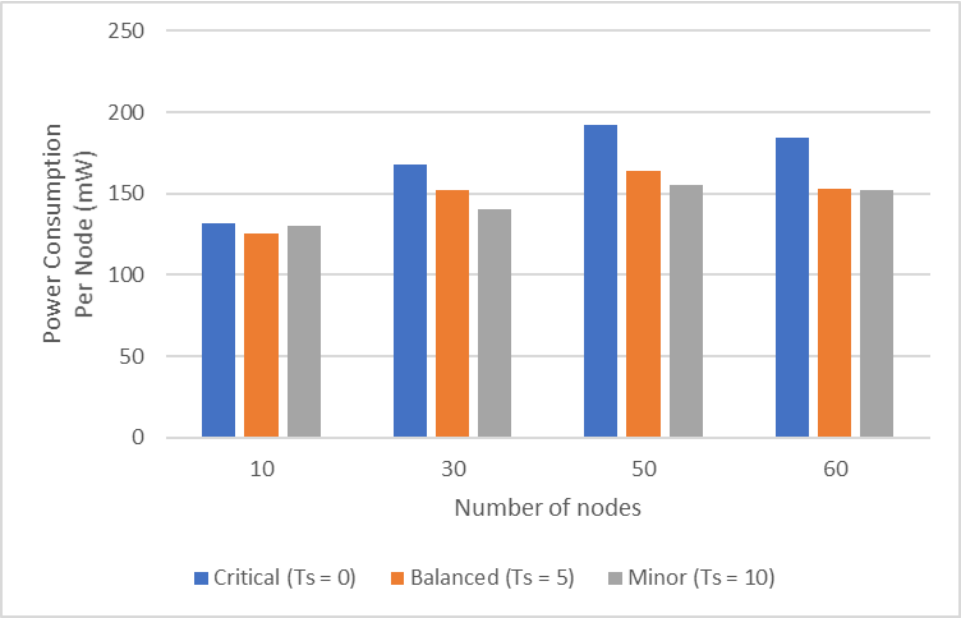

Figure 18 Graph showing comparison of average power consumed per node in the disseminations with different T_s values as number of nodes increasing

As Figure 18 shows that, at 60 nodes, the disseminations have slightly lower average power consumption per node, and we suspect that as increasing number of nodes there may be higher chance that nodes can be switch off earlier. So, we plotted average time that nodes had switched to Normal Mode (off time) before every nodes completed the download in Figure 18.

In Figure 19, at 10 nodes, the disseminations completed too fast, so we are going to discuss about the result of 30 nodes and higher. The percentages of off time at each T_s value are higher as number of nodes are increasing.

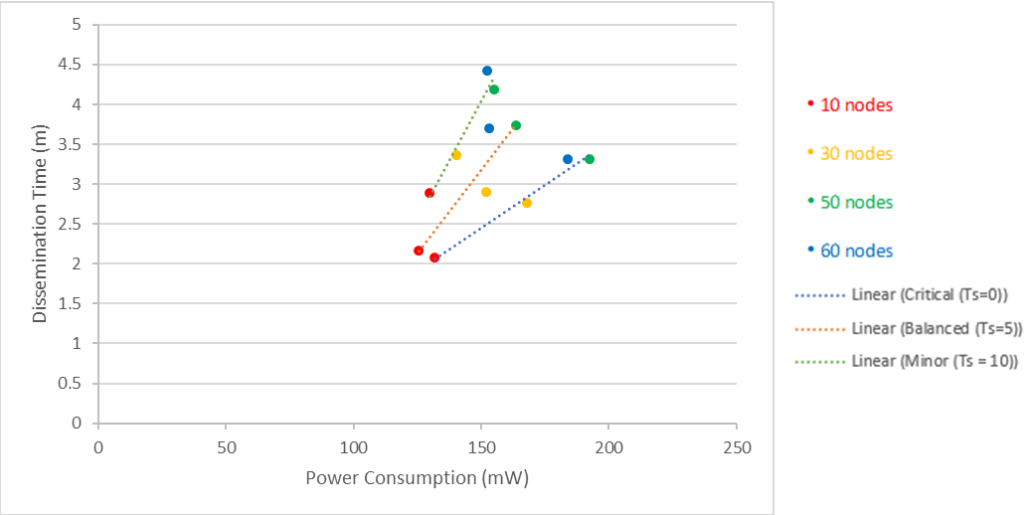

Figure 19 Graph showing trendlines of average early off time percentage

4.2.3 Intermittent Failures

We also tested the reliability of our algorithm with random intermittent failures. By simulating the intermittent failures, we have modified Contiki-NG network stack to make it not accept and incoming packets or send out any packets during the failures.

We simulated every topology with different rate of intermittent failures, 0%, 1%, 3%, and 5%. In 10-node and 20-node topologies, a failure will randomly occur to a node every 5 and 10 seconds, respectively. In 30-node or more, a failure will randomly occur to a node every 15 seconds. For each failure occurring, the node will be not responsive for 15 seconds.

The results in Figure 20 and Figure 21 show that intermittent failure occurring affects the performance of the dissemination but not significantly because a node can ignore unavailable nodes and continue downloading from others.

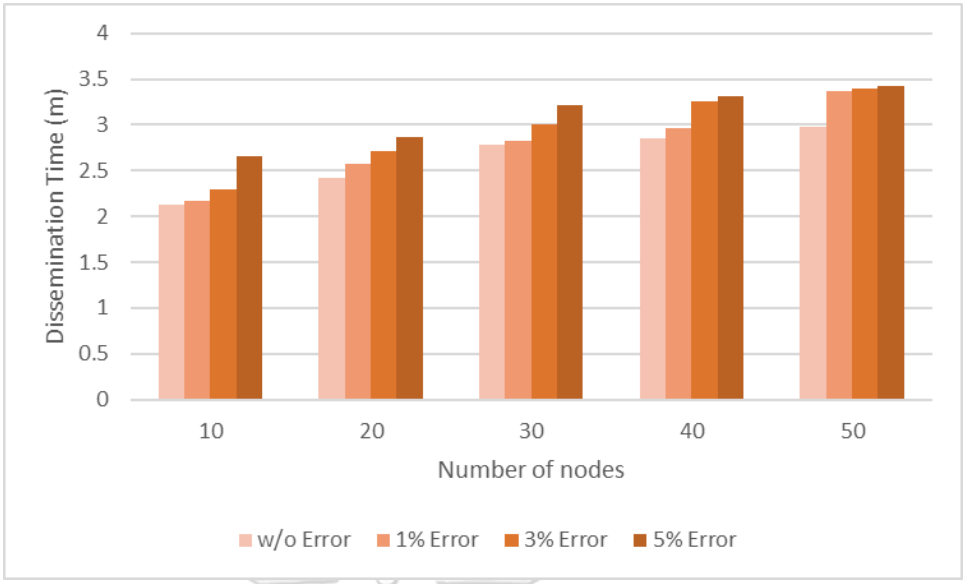

Figure 20 Graph showing comparison of average time used in the disseminations using our algorithm with multiple rates of intermittent faults as number of nodes increasing

Figure 21 Graph showing comparison of average power consumption per node in the disseminations using our algorithm with multiple rates of intermittent faults as number of nodes increasing

4.2.4 Communication Overhead

To analyze the communication overhead of our algorithm, we classify the messages sent from nodes as control messages or data messages. HANDSHAKE, INTEREST, CHOKE, UNCHOKE, and REQUEST messages are control messages used to synchronize the flow of the communications and BLOCK messages are the actual data. The percentages of number of these two are showing in Table 2.

The number of BLOCK messages or actual data is around 40-47% of all packets but the number is decreasing as the number of nodes increasing. While HANDSHAKE, INTEREST, and CHOKE messages are increasing as the number of nodes increasing. However, considering only their sizes, the size of BLOCK messages or actual data is around 86-91% of all packets.

Number of Messages (%)	10 nodes	20 nodes	30 nodes	40 nodes	50 nodes
HANDSHAKE Messages	13.11	16.70	21.56	22.13	23.83
INTEREST and CHOKE Messages	27.55	26.61	26.42	24.67	24.49
REQUEST Messages	12.68	12.13	11.43	11.30	11.13
BLOCK Messages (Data)	46.66	44.55	40.59	41.90	40.54

Table 2 Table showing percentage of Number of Control Messages vs. Number of Data Messages in the disseminations as number of nodes increasing

Size of Messages (%)	10 nodes	20 nodes	30 nodes	40 nodes	50 nodes
HANDSHAKE Messages	3.80	5.01	6.92	6.92	7.64
INTEREST and CHOKE Messages	4.67	4.66	4.51	4.51	4.59
REQUEST Messages	1.23	1.21	1.22	1.18	1.19
BLOCK Messages (Data)	90.30	89.11	87.39	87.39	86.59

Table 3 Table showing percentage of Size of Control Messages vs. Size of Data Messages in the disseminations as number of nodes increasing

4.2.5 Code size

As our algorithm was designed to be a side program intended to retrieve and load the main application so it requires some additional space in flash memory of the microcontroller. Considering multicast is serving the same purpose, Table 3 shows the comparison of the code size that will take up space in the flash memory. Note that each combined with Contiki-NG operating system.

	Root	Sink
Our algorithm	60.15 KiB	60.07 KiB
MPL	56.99 KiB	56.66 KiB
ESMRF	50.95 KiB	50.75 KiB

Table 4 Table showing code size of our algorithm, MPL, and ESMRF

As shown in Table 4, in exchange of higher performance, our algorithm takes more space for around 5.68% and 15.52% than MPL and ESMRF respectively and in addition to this, MPL and ESMRF can also be used to send multicast packets in other purpose other than for software updates.

4.2.6 Implementation Challenges

Since Contiki-NG process model are built on top of lightweight stackless threading library called Protothreads which aims to reduce memory overhead of multithreading design. So, the scheduler switches between threads without storing any states. So, the challenge is to define our own state and store it in efficient way.

Because of inability to use large code size TCP transport protocol as in BitTorrent, we use UDP transport protocol instead. The UDP transport protocol does not handle packet order and packet loss so we had to design our handling mechanism by ourselves.

We used an unsigned 32-bit integer to represent a 32-piece possession bitfield which each bit representing each piece, this makes it easier to do bitmasking in piece selection process. But if a network wants the number of pieces to be more than 32, redesigning in bitfield storing may be required.

5 CONCLUSION

In this research, we have introduced a decentralized energy-speed trade-off configurable peer-to-peer data dissemination algorithm for wireless sensor networks that adopted several features from BitTorrent for being highly scalable and efficient.

Our algorithm can be configured for energy-speed trade-off for 3 priority levels of software updates: Critical (T_s = 0), Balanced (T_s = 5), and Minor (T_s = 10). The algorithm disseminates the Critical Updates fastest while consume around 8-20% more power. But disseminating Minor Updates, the algorithm can achieve highest energy efficiency but spends 15-25% more time than Balanced Updates and Critical Updates.

The evaluation results also show that our algorithm performs better than multicast-based algorithms in term of download speed up to 58.97% and energy efficient up to 79.39%. In order to achieve these results, our proposed algorithm requires additional information regarding to neighbors from the operating system. Furthermore, it requires more memory to operate comparing to those multicast-based algorithms. Nevertheless, these can be considered minor limitations as modern sensors become more powerful.

Although we have designed this algorithm to be portable that can operate on other protocols but as it operates by exchanging large number of messages, it is still inconclusive that the performance of this algorithm would still be good if it worked on networks with higher latency such as LoRA networks or ZigBee networks.

In future works, we will focus on adapting our algorithm for using in Unattended Wireless Sensor Networks (UWSN). A UWSN is a wireless sensor network that monitors an extreme environment such as volcano or battlefield [17]. The sensed data cannot be transmitted immediately, instead it will have to be stored for being retrieved later. Because of the necessity of storing data in extreme conditions, one of the important factors is the survivability of the stored data. Thus, our algorithm will enable nodes in an UWSN to back up their sensed data by efficiently distribute the data to their neighbors in order to extend survivability of the data.

REFERENCES

1. Chlipala, A., J. Hui, and G. Tolle, *Deluge: Data Dissemination for Network Reprogramming at Scale.* 2004.
2. Kulkarni, S.S. and W. Limin, *MNP: Multihop Network Reprogramming Service for Sensor Networks*, in *25th IEEE International Conference on Distributed Computing Systems (ICDCS'05)*. 2005. p. 7-16.
3. Kushalnagar, N., G. Montenegro, and C. Schumacher, *IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals.* 2007.
4. Lavric, A. and V. Popa, *Internet of Things and LoRa™ Low-Power Wide-Area Networks: A survey*, in *2017 International Symposium on Signals, Circuits and Systems (ISSCS)*. 2017. p. 1-5.
5. BitTorrent Foundation *BitTorrent (BTT) White Paper*. 2019.
6. Winter, T., et al., *RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks.* 2012.
7. Levis, P., et al., *The Trickle Algorithm.* 2011.
8. *IEEE Standard for Low-Rate Wireless Networks*. IEEE Std 802.15.4-2015 (Revision of IEEE Std 802.15.4-2011). p. 1-709.
9. Hui, J. and R. Kelsey, *Multicast Protocol for Low-Power and Lossy Networks (MPL).* 2016.
10. Clausen, T., A.C. de Verdiere, and J. Yi, *Performance analysis of Trickle as a flooding mechanism*, in *2013 15th IEEE International Conference on Communication Technology*. 2013. p. 565-572.
11. Oikonomou, G., I. Phillips, and T. Tryfonas, *IPv6 Multicast Forwarding in RPL-Based Wireless Sensor Networks.* Wireless Personal Communications, 2013. **73**(3): p. 1089-1116.
12. Abdel Fadeel, K.Q. and K. El Sayed, *ESMRF: Enhanced Stateless Multicast RPL Forwarding For IPv6-based Low-Power and Lossy Networks*, in *Proceedings of the 2015 Workshop on IoT challenges in Mobile and Industrial Systems - IoT-Sys '15*. 2015. p. 19-24.
13. McGoldrick, C., et al., *TinyTorrents - Integrating Peer-to-Peer and Wireless Sensor Networks*, in *2009 Sixth International Conference on Wireless On-Demand Network Systems and Services*. 2009. p. 119-126.

14. AndreasF. *Thread Tutorial: Practical guide for device upgrade OTA*. 2019; Available from: https://devzone.nordicsemi.com/nordic/short-range-guides/b/mesh-networks/posts/thread-tutorial-practical-guide-for-device-upgrade-ota.

15. Eriksson, J., et al., *Scaling RPL to Dense and Large Networks with Constrained Memory*, in *Proceedings of the 2018 International Conference on Embedded Wireless Systems and Networks*. 2018, Junction Publishing: Madrid, Spain. p. 126–134.

16. Dong, W., et al., *Dynamic linking and loading in networked embedded systems*, in *2009 IEEE 6th International Conference on Mobile Adhoc and Sensor Systems*. 2009. p. 554-562.

17. Choi, H.-B., Y.-B. Ko, and K.-W. Lim, *Energy-Aware Distribution of Data Fragments in Unattended Wireless Sensor Networks*, in *2018 Third International Conference on Security of Smart Cities, Industrial Control System and Communications (SSIC)*. 2018. p. 1-8.

VITA

NAME	Natchanon Nuntanirund
DATE OF BIRTH	10 January 1994
PLACE OF BIRTH	Phetchabun
INSTITUTIONS ATTENDED	Chulalongkorn University