Micro-Moving Target IPv6 Defense for 6LoWPAN and the Internet of Things

Matthew Gilbert Sherburne

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

Electrical Engineering

Joseph G. Tront, Chair

Scott F. Midkiff

Randolph C. Marchany

April 3, 2015

Blacksburg, Virginia

Keywords: Internet of Things, IPv6 Security, 6LoWPAN, Moving Target Defense

Copyright 2015, Matthew Gilbert Sherburne

Micro-Moving Target IPv6 Defense for 6LoWPAN and the Internet of Things

Matthew Gilbert Sherburne

(ABSTRACT)

The Internet of Things (IoT) is composed of billions of sensors and actuators that have varying tasks aimed at making industry, healthcare, and home life more efficient. These sensors and actuators are mainly low-powered and resource-constrained embedded devices with little room for implementing IP security in addition to their main function. With the fact that more of these devices are using IPv6 addressing, we seek to adapt a moving-target defense measure called Moving Target IPv6 Defense for use with embedded devices in order to add an additional layer of security. This adaptation, which we call Micro-Moving Target IPv6 Defense, operates within IPv6 over Low power Wireless Personal Area Networks (6LoWPAN) which is used in IEEE 802.15.4 wireless networks in order to establish IPv6 communications. The purpose of this defense is to obfuscate the communications between a sensor and a server in order to thwart a potential attacker from performing eavesdropping, denial-of-service, or man-in-the-middle attacks. We present our work in establishing this security mechanism and analyze the required control overhead on the wireless network.

Acknowledgments

I want to thank my committee including Dr. Joseph Tront, Dr. Scott Midkiff, and Prof. Randy Marchany for guiding and mentoring me along this path. To everyone in the IT Security Office and Lab, thank you for your unwavering support, assistance, and wisdom you have lent along the way. I also want to thank Dr. Michael Brownfield for giving me the inspiration to pursue research in the area of wireless sensor networks and also to Dr. Scot Ransbottom for keeping that inspiration lit. To my wonderful family - Emily, Colin, and Ryan for supporting me during a very busy two years.

Contents

1 Introduction **1**

1.1 Purpose and Goals 3

1.2 Structure of Thesis 3

2 Background **5**

2.1 Internet of Things 5

2.2 IPv6 8

2.2.1 IPv6 Address Assignment 10

2.2.1.1 SLAAC 10

2.2.1.2 DHCPv6 11

2.3 6LoWPAN 12

2.4 RPL 14

3 Related Work **15**

3.1 TLS and DTLS . . . 15

3.2 IPsec . . . 16

3.3 Intrusion Detection System . . . 17

3.4 MT6D . . . 17

3.4.1 Motivation to Use MT6D . . . 18

3.4.2 Purpose of MT6D . . . 19

3.4.3 MT6D Address Hashing . . . 19

3.4.4 MT6D Implementation . . . 20

4 Testbed Creation **23**

4.1 Software Selection . . . 23

4.1.1 Contiki OS . . . 24

4.1.2 802.15.4 Packet Sniffer - Foren6 . . . 24

4.2 Hardware Selection . . . 26

4.2.1 Tmote Sky . . . 26

4.2.2 Econotag II . . . 27

4.2.3 Raspberry Pi . . . 27

4.3 Hardware and Software Integration 28

5 Implementing MT6D Functionality in Contiki OS **32**

5.1 ICMPv6 RPL Control Messages 34

5.2 Dynamic Addressing Changes 36

5.3 Establishing NTP 44

5.4 Performing Hashing of New IPv6 Address 47

6 Evaluation of μMT6D **49**

6.1 Methods 49

6.1.1 Address Creation and Route Addition Success Rate 50

6.1.2 Binary File Size 50

6.1.3 Average Current Consumption 51

6.2 Results 53

6.2.1 Evaluation of μMT6D Hashing Process 54

6.2.2 Binary File Size 55

6.2.3 Average Current Consumption 56

7 Conclusion and Future Work **59**

7.1 Conclusion 59

7.2 Future Work 61

Bibliography **62**

List of Figures

2.1 Smart Home Connectivity Diagram 6

2.2 IPv6 Header 9

2.3 IPv6 Address Format 9

2.4 6LoWPAN Architecture 13

3.1 MT6D Operating Modes 21

3.2 Smart Home with MT6D Protection 22

4.1 Foren6 Screenshot 25

4.2 IoT Network Diagram 28

4.3 Physical Testbed 31

5.1 μMT6D Flowchart 33

5.2 Base Configuration RPL Packet Exchange 35

5.3 Address Change Function Flowchart 38

5.4 Route Addition Success Rate 42

5.5 RPL with Address Changing 43

6.1 Average Current Consumption Measurement 52

6.2 Route Addition Success Rate with Hashing 55

6.3 Average Current Consumption 58

List of Tables

6.1 Comparison of Binary File Sizes. 56

6.2 Comparison of Average Current Consumption. 57

Chapter 1

Introduction

The proliferation of the Internet of Things (IoT) has developed a massive increase in IP-based traffic originating not from computers and mobile devices, but from autonomous sensors, actuators, and even home appliances. These devices and appliances include ovens, refrigerators, air conditioners, smoke detectors, baby monitors, thermostats, door locks, light bulbs, and power outlets just to name a few. These devices are also commonly sold with poor security incorporated and default passwords that consumers never change. These factors, combined with the fact that they are constantly connected to the Internet, have led to an increase in malicious activity. Proofpoint, Inc. discovered that between December 23, 2013 and January 6, 2014, more than 100,000 Internet-connected devices such as home-routers, Smart TVs, media centers, and at least one refrigerator were involved in a cyber attack consisting of 750,000 malicious emails sent to Enterprises and individuals around the world [44]. Now that these devices are handling important tasks in the evolving Smart Home such as appliance

control, personal monitoring, and physical security, there is more at stake. Hewlett-Packard and their Fortify unit conducted a study in 2014 which found that, "90 percent of devices collected at least one piece of personal information via the device, the cloud, or its mobile application," and, "70 percent of devices used unencrypted network service." [25] Collection of personal information combined with unsecured communications presents a clear security risk and can allow an attacker to eavesdrop. An even more ominous scenario could play out if an attacker can gain full access to a smart home. They can use sensors and monitors to observe when the residents are away and shutoff the thermostat in the winter to cause the water pipes to break. These devices are not just limited to Internet-connected objects around the home. There are also wearable devices in sectors such as security and safety, medical, wellness, sport and fitness, lifestyle computing, communication, and glamour [3]. These applications include fitness monitoring, emotional response, and responsive learning. This means that this subset of the IoT will have longer and closer access to personal data.

With so many different devices operating in a smart home, a framework that can assess their security vulnerabilities can help develop and standardize solutions. Denning et al. [12] discuss a framework for evaluating security risks of technology used in the home. There are many reasons to fix security risks found in this framework. For example, we want to keep the identities of our devices private so that attackers can not find out their true and sometimes sensitive purpose such as a weight scale. We also want to ensure command authenticity so that attackers can not send a malicious command to the device. In the case of our research, we want to secure the communications between devices in the home that communicate with

servers in the cloud.

1.1 Purpose and Goals

In this thesis, we seek to implement and evaluate a moving target defense called Moving Target IPv6 Defense (MT6D) in order to hide the fact that a conversation is taking place between a sensor in the home and a server in the cloud by changing and obfuscating IPv6 addresses thus implementing end-to-end security. Micro-Moving Target IPv6 Defense (μMT6D) as proposed here is designed to provide this security on low-powered and resource-constrained sensors. We seek several goals in this implementation. The first goal is to implement this defense technique on a common Internet of Things wireless network operating IPv6. Next we want to implement the technique at the sensor level and make no changes at the wireless border router of this network. Last, we want to minimize the code space and additional energy consumption this technique could have on the sensor.

1.2 Structure of Thesis

This thesis is organized by first discussing the background of the IoT in Chapter 2. A survey on related work is provided in Chapter 3. We then explain our choice of hardware and software in establishing a physical IoT testbed in Chapter 4. In Chapter 5 we describe how we implemented μMT6D on our testbed. The results of our evaluation of μMT6D are

reported in Chapter 6. Finally, in Chapter 7 we offer our conclusions and suggestions for future work.

Chapter 2

Background

In this chapter we will discuss the Internet of Things and an example of a smart home. We will then discuss IPv6 communications that take place between a sensor in the home with a server in the cloud. Next we will discuss the open standard, 6LoWPAN, that allows IPv6 communications to extend to wireless sensors. This information will provide the information necessary to understand the IoT standards and protocols in which we will implement MT6D.

2.1 Internet of Things

According to Beecham Research, the IoT can be broken down into nine service sectors: buildings, energy, consumer and home, healthcare and life science, industrial, transportation, retail, security and public safety, and IT and networks [49]. They all contain sensors and actuators performing various tasks. Each service sector seeks to make our lives more

efficient by surrounding ourselves with sensors and actuators all connected via the Internet. Never before have we had so much data reporting on our daily lives that is produced by autonomous sensors. In the future, we will be enjoying efficiencies in this automated lifestyle e.g., returning home from work with a house set to a comfortable temperature and an oven pre-heated for cooking. This is very exciting, but it also presents major security challenges when it comes to protecting the data these sensors produce or the commands sent to actuators and devices. An overview of an example smart home, found within the consumer and home service sector of the IoT is seen in Figure 2.1

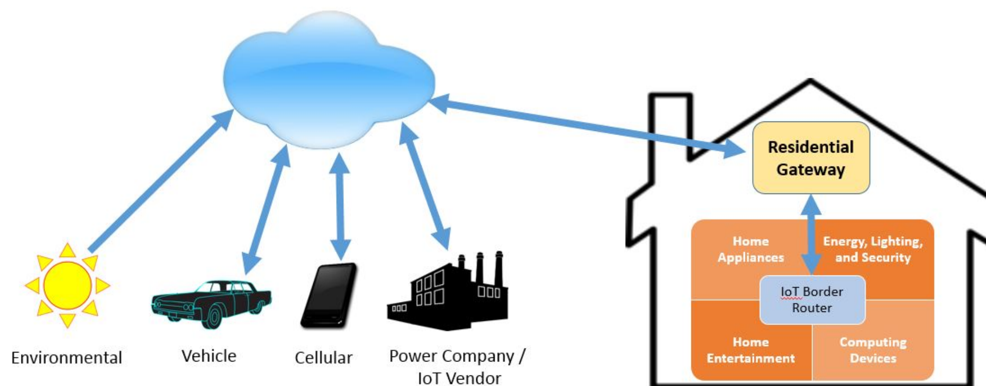

Figure 2.1: Smart Home Connectivity Diagram

In this example of a smart home, an IoT wireless border router connects various sensors and actuators to the Internet represented by the cloud. Service providers, manufacturers, environmental data, vehicles, and mobile devices also interface with the cloud in order to communicate with the smart home. Weather forecasts can give the smart home preparation to adjust to temperature changes. The manufacturer of the sensors and actuators used in the smart home can monitor the maintenance requirements of an appliance or push new updates

to the sensors and actuators themselves. As we have discussed in the last chapter, a large majority of these devices do not implement a satisfactory level of security. The United States Federal Trade Commission, tasked with protecting consumers, has only recently weighed in on security discussions about consumer-purchased IoT devices used within the home. Their staff report, however, does not require but only recommends very generic security best practices to manufacturers [6]. The FTC has very good reason to start this discussion, since the sheer number of devices that make up the IoT will be large. ABI Research, a market forecast research company, estimates that there will be 40.9 billion wireless connected IoT devices in 2020 [48]. Similarly, Cisco's Internet Business Solutions Group (IBSG) is predicting similar figures. They predict that 25 billion devices will be connected to the Internet in 2015 and increasing to 50 billion by 2020 [18]. IBSG also recognizes that the slow transition to IPv6, disparate standards, and developing new energy sources have acted as barriers in slowing IoT development. Billions of IoT devices will require unique IPv6 addresses and new energy sources to keep them powered longer. Companies, such as Texas Instruments are actively engaged in energy harvesting as an alternative energy source for the IoT using various sources such as solar, thermal electric, electromagnetic, and vibration energy [55]. There are many different groups forming to organization and standardize the IoT. The three main groups today in competition are the AllSeen Alliance [54], the Thread Group [22], and the Open Interconnect Consortium [7]. Each group has representation from major technology companies.

It is prudent that we engage in developing new methods of security to maintain confiden-

tiality, availability, and integrity. That is easier said than done considering the numerous different standards that operate on the IoT. Our research is focused specifically on the subset of IoT devices that communicate wirelessly using the IEEE 802.15.4 standard [1] and IPv6 over Low power Wireless Personal Area Networks (6LoWPAN) [27] to communicate at Layer 3 using IPv6 addressing. We will discuss the details of IPv6 in our next section.

2.2 IPv6

A basic understanding of Internet Protocol version 6 (IPv6) [11] is needed to understand why MT6D and address hopping can work without causing issues for the Internet. IPv6 was developed in response to the shortage of IPv4 addresses. IPv6, with 128-bits of addressing, has a significantly larger addressing space than IPv4's 32-bits of addressing. To be precise, there are $2^{96} = 7.9 \text{x} 10^{28}$ more IP addresses in IPv6 than in IPv4. In fact, each square nanometer of the Earth's surface, including all the oceans, can represent 667,000 IPv6 addresses. Figure 2.2 shows the structure of an IPv6 header. This header includes the IP version, traffic class, flow label, payload length, next header, hop limit, and the source and destination addresses. In our discussion of 6LoWPAN in the next section we will show how this header is compressed in order to traverse a low-powered wireless network.

Within IP networking we have networks and hosts. Hosts are defined as computing endpoints of the network that have an IPv6 address. IPv6 typically subnets networks to 64 bits, or the first half of the IPv6 address. The last 64 bits are then used for the host address. Figure 2.3

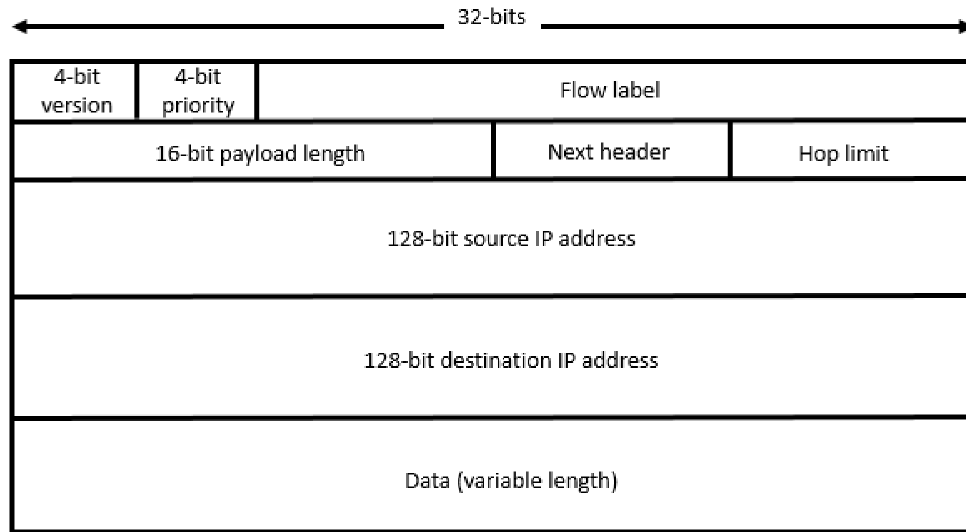

Figure 2.2: IPv6 Header

shows how the IPv6 global link address is formatted. The first 64 bits of the IPv6 address represents the network portion. The last 64 bits represent the host portion. The 64 bits of this host portion is provided by the interface ID (IID). We will now explain how IPv6 addresses are generated and assigned to hosts.

Figure 2.3: IPv6 Address Format

2.2.1 IPv6 Address Assignment

IPv6 uses two different ways to automatically assign addresses on a network: Stateless Address Autoconfiguration (SLAAC) [56] and Dynamic Host Configuration Protocol for IPv6 (DHCPv6) [14]. SLAAC is the most commonly used address assignment method, especially for networks where strict control of addresses is not a concern as long as host addresses are valid and routable. DHCPv6 lets network administrators have control over host address assignment

2.2.1.1 SLAAC

SLAAC enables a simplified method to address assignment that requires no servers. With an IPv6 router configured for use with SLAAC, hosts can form their own IPv6 addresses and advertise them to the router. This self-address creation process is typically performed by using the host's MAC address, used as an interface identifier (IID), to form the link-local host address and further described in RFC 4291 [26]. The host, after forming this address, will verify that it is unique and not a duplicate on the network by sending a Neighbor Solicitation (NS) message to the network. If the host receives no response from the network, then a valid address is formed and the address is assigned to the interface. The host, now with a valid link local address, transmits a Router Solicitation (RS) message to see if a router is present. If a router, configured to run SLAAC, is present on the network, it will respond with a Router Advertisement (RA) message that will contain its network address

(subnet prefix) and lifetime value. Once the host has received this RA message, it now can form its global link address be combining the subnet prefix including its host address with a valid lifetime set in the RA message. The advantage to this address assignment is that it is simple, but the disadvantage is that the host could be configured to use the same interface ID for its host address. If the host is mobile while traversing different IPv6 networks, an attacker could track that host logically and physically by filtering traffic for the same last 64 bits that form the interface ID. Next we will describe the alternate address assignment process, DHCPv6.

2.2.1.2 DHCPv6

DHCPv6 as defined in RFC 3315 [14] allows the network to assign the host an address. The address assignment process for DHCPv6 starts by a host sending a Solicit message to the IANA defined DHCP relays and servers multicast address. A DHCPv6 server that can service the host's request will respond with an Advertise message. In the case that multiple servers respond to the Solicit message, the host will pick one of the servers to which it will send a Request message. The server to which the Request message is sent will then respond with a Reply message. The Reply message will confirm address assignment and other related configuration information. DHCPv6 is now concluded once the host receives the Reply message. The lifetime of the address is specified by the server. The host will send a Renew message to the server to request an extension. The server will send a Reply message with the new lifetime. Normally the host will send a Solicit message every time it boots up.

Even if the host has an address with a valid lifetime, the parameters of the network could have changed. Our next section will discuss how IPv6 is implemented in IEEE 802.15.4 wireless networks.

2.3 6LoWPAN

Wireless sensor networks operating with IEEE 802.15.4 will utilize IPv6 over Low power Wireless Personal Area Networks (6LoWPAN) [27] in order to extend IPv6 addressing to wireless sensors. RFC 4919 [30] and RFC 4944 [36] discuss the issues of assigning IPv6 to wireless sensor network devices. 6LoWPAN is an adaption layer that lies between the data link and network layers. This adaptation layer performs three different functions: packet fragmentation and reassembly, header compression, and data link layer routing for multihop connections. Packet fragmentation and reassembly is required because the Maximum Transmission Unit (MTU) of an IPv6 packet is 1280 bytes and the 802.15.4 frame has a MTU of only 127 bytes. Therefore packet fragmentation will occur and 6LoWPAN will handle the control of these fragments. If the IPv6 packet can fit within the frame MTU, then no fragmentation takes place. 6LoWPAN can also can compress the IPv6 header to further reduce its size and make more room for payload data. In the case of implementing MT6D, we do not want to enable compression of the IPv6 addresses. Figure 2.4 shows the architecture of 6LoWPAN. As this network is an ad-hoc mesh network, there needs to be a routing protocol present. A node, such as a wireless sensor, participating in the 6LoWPAN can act either as

a router, host, or both depending on the mobility of the nodes. Communications that route from host nodes to the border router are said to travel upstream. Communications traveling from the border router to the host node moves downstream. Solid lines between router nodes and host nodes represent the paths communications take. The dashed line between the two router nodes represents an alternate route that can change to a primary route if another router node is no longer in communication with the border router. Routing protocols take care of establishing primary routes and alternate routes so that routing loops do not occur. Our next section will discuss the underlying ad-hoc mesh network routing protocol used in 6LoWPAN called RPL.

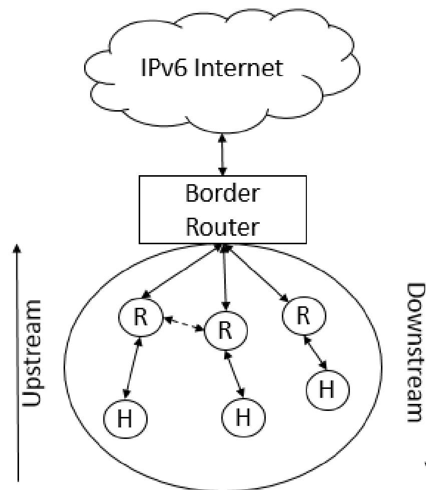

Figure 2.4: 6LoWPAN Architecture

2.4 RPL

IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL) is defined in RFC 6550 [57]. It was developed by the IETF Routing over Low-power and Lossy Networks (ROLL) working group. It is a hybrid of the Ad-Hoc On-Demand Distance Vector Routing (AODV) [41] and Optimized Link State Routing (OLSR) [5] protocols producing a proactive distance-vector routing protocol. RPL is necessary in order for wireless sensors to learn about their neighbors and border router. We will discuss more about RPL and the control messages it uses as it pertains to our research in Chapter 5.

We have discussed the background information of the IoT and how it will continue to increase in size and complexity. IPv6 and its addressing size will be able to assign unique, globally routable IP addresses to all the IoT devices. We have covered the basic standards and protocols that exist for a predominant portion of the IoT whose sensors use IEEE 802.15.4 for low-powered wireless communications. Finally we discussed how 6LoWPAN and RPL are used to extend IPv6 communications, addressing, and routing to these sensors. In our next chapter, we will discuss several techniques of securing 6LoWPAN and provide their advantages and disadvantages. We will then discuss the operation of MT6D in detail.

Chapter 3

Related Work

There is a vast amount of research into securing the embedded Internet at various layers of the Open Systems Interconnection (OSI) model and with different techniques. We will discuss several security schemes that are designed for use with 6LoWPAN. Finally, we will describe in detail how MT6D works since it forms the basis of our research.

3.1 TLS and DTLS

Perelman [40] implements a limited version of Transport Layer Security (TLS) [13] and Datagram Transport Layer Security (DTLS) [47] for use with an AVR Raven running Contiki OS. TLS is a security scheme that encrypts Transport Layer TCP datagrams. DTLS is a security scheme, based off of TLS, that encrypts Transport Layer UDP datagrams. Wireless sensors, depending on the application and function, will use either TCP or UDP commu-

nications. The challenge with implementing TLS or DTLS is the memory resources they consume on a resource-constrained sensors. Perelman concluded that this implementation is viable, but only using a cipher suite lightweight enough to run on a resource-constrained sensor. This research brings continued emphasis on securing the IoT through encryption. The overall shortcoming here is that TLS or DTLS, in any form, still allows the observation of the conversation because the IP addresses are static at each end of the communications. Next we will describe a general purpose security standard built-in to IPv6.

3.2 IPsec

Raza et al. [45] propose adapting IPsec [29], an authentication and encryption protocol, already included in IPv6 for use with 6LoWPAN. IPsec communications enables end-to-end confidentiality, integrity, and authentication at the Network Layer and is independent on what Transport Layer method is used. Due to the compression that 6LoWPAN can conduct on IPv6 communications, these researchers adapted IPsec within 6LoWPAN so that it can function with other hosts and networks also enabled with IPsec on the Internet. The shortcoming to this implementation is that an attacker can still observe the conversation due to the fact that the IPv6 source and destination addresses are visible in the basic setting of IPsec. Even IPsec Encapsulated Security Protocol (ESP) [28] in tunnel mode will still produce a static IPv6 tunnel address. If an attacker gains access to the trusted, internal network, they can observe which host is communicating. Next we will describe another

security technique designed to detect malicious behavior on the network.

3.3 Intrusion Detection System

Le et al. [31] assessed that encrypting traffic alone does not secure the wireless sensor network from external or internal attack. Their approach is to introduce an intrusion detection system (IDS) as another layer of defense in order to monitor malicious activity internal and external to the wireless network. Having such a security mechanism employed does provide an additional layer of defense that can alert if a potential attack is taking place. The shortcoming here is that an IDS in combination with encryption e.g., TLS, DTLS, or IPsec, still does not protect the network from passive eavesdropping. The static IPv6 addresses used at each end of communications allows for an attack to be specifically targeted at those addresses that an IDS may or may not be able to detect. Next we will explain MT6D and how it is designed to hide the fact a conversation is even taking place to solve this potential vulnerability.

3.4 MT6D

Dunlop et al. [17], developed an IPv6 defense scheme that provides security through obscurity by rapidly changing IPv6 addresses of two end points participating of a conversation, similar to frequency hopping in radio communications. The idea of extending MT6D into em-

bedded devices, more specifically the Smart Grid, came with Groat et al. [21]. Although they proposed this extension, they did not evaluate the efficacy of implementing MT6D on such resource-constrained devices. This paper provided the motivation for us to develop a version of MT6D that can be adapted for use on common, low-powered and resource-constrained devices. We will now fully explain how MT6D works.

3.4.1 Motivation to Use MT6D

As discussed in the previous chapter, there are two methods of address assignment in IPv6: SLAAC and DHCPv6. These methods potentially cause privacy and security issues based on how they build the host address. If these addresses are formed using the same unique identifier process across any network, then it is possible to track the same last, and unique, 64 bits of the IPv6 address if that host moves between networks. This process is true for the case of embedded devices using 6LoWPAN since they normally implement SLAAC. The SLAAC method will normally generate its interface identifier using the globally unique MAC address. As discussed before, this forms the link local address and also the global link address. A similar issue arises in DHCPv6, but the privacy and security concerns stem from the use of the same DHCP unique identifier (DUID) the host connects with across any network. This vulnerability requires an attacker to have internal access to the network in order to observe this DUID. Although DUIDs are a concern, SLAAC-generated unique host addresses do not require internal network access to exploit tracking of hosts across the network.

MT6D was designed to address the issue of static IPv6 addresses and unique host addresses communicating across the network by taking advantage of SLAAC. MT6D works by changing the source and destination host's IPv6 addresses at predetermined time intervals in order to hide the fact a conversation is taking place. It does this by calculating the addresses based on a cryptographic hashing algorithm making it computationally difficult for an attacker to know which addresses the two hosts in communication will change to next.

3.4.2 Purpose of MT6D

MT6D's purpose is to hide the fact that two hosts are communicating over an untrusted network by not disguising their true identities. An attacker attempting to follow a conversation based on observing the IPv6 source and destination addresses will have a difficult time trying to establish the conversation in the first place. This difficulty is due to the fact that the addresses constantly rotate with time. In addition, and besides obfuscating IPv6 address, MT6D can also tunnel traffic encrypted or unencrypted.

3.4.3 MT6D Address Hashing

MT6D implements a cryptographic algorithm in order to hash and generate the next source and destination address a host. MT6D generates the new addresses by hashing the following inputs: the initial IID of the source or destination host, the session key, and timestamp. Using a hashing algorithm such as SHA256, MT6D will concatenate the IID, session key,

and timestamp in order to calculate a new IID. A host will perform this hash for both its next source address and destination address. Likewise, the other host in communication will also perform this same hashing calculation. The session key discussed is a symmetric, pre-shared key communicated out-of-band. As long as each host hashes with the same initial IID, symmetric session key, and timestamp, they can successfully compute a matching set of source and destination addresses at both ends of the communication. Due to slight inaccuracies of network time used for the timestamp, each host has a sliding window of addresses bounded to its interface. A later revision of MT6D also includes changing source and destination ports to further obfuscate communications.

3.4.4 MT6D Implementation

MT6D can be implemented in one of two ways: on the wireless sensor itself (host mode) or on a device that borders the untrusted network (gateway mode). These two operating modes are shown in Figure 3.1. MT6D is designed to look at all packets and will either handle MT6D communications or let non-MT6D traffic pass. Implementing MT6D on the sensor itself means that the sensor must use its own resources to compute the addresses. A gateway mode implementation allows a separate device to handle the computation instead of a sensor or sensors within the trusted network the gateway borders. The gateway can also be a central management point for all MT6D links terminated to that network and allows easier management of session keys. Gateway mode, in its current form, has some inherent drawbacks such as representing a single point of failure and also containing all the session

keys for MT6D links terminated on that gateway. In host mode, each sensor will change its own address and maintain its own MT6D parameters.

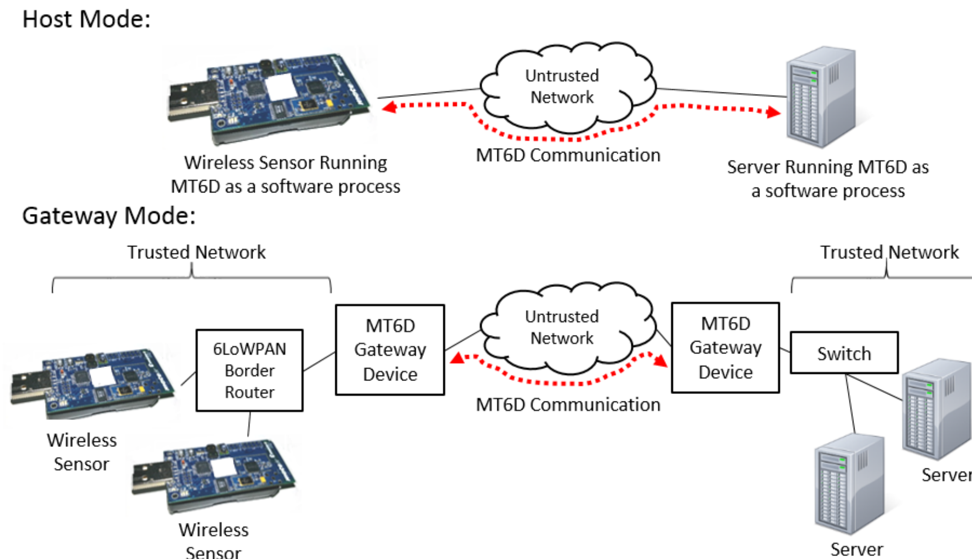

Figure 3.1: MT6D Operating Modes

We have decided to implement the host mode of MT6D operation for use in 6LoWPAN. A gateway mode implementation could easily be established for 6LoWPAN based off the findings of the creators of MT6D. We wanted to investigate if MT6D could be implemented directly on resource-constrained devices for end-to-end security. Figure 3.2 shows MT6D host mode as visualized with our smart home example. In this case, a MT6D secured link exists between a home appliance and its manufacturer's appliance maintenance server. The appliance, which can be thought of as a wireless sensor and actuator, and the server are both changing IPv6 addresses on their communications interface.

In summary, there are security methods that exist in 6LoWPAN such as TLS and DTLS,

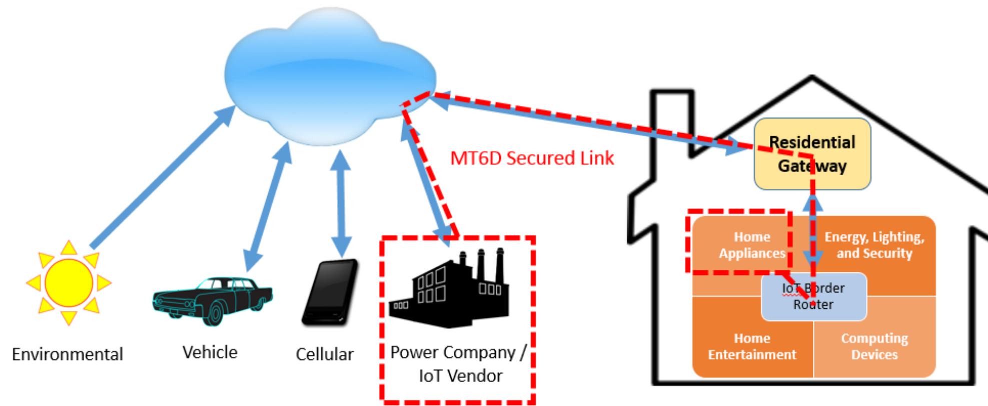

Figure 3.2: Smart Home with MT6D Protection

compressed IPsec, and intrusion detection systems that can be employed. They do have merit when it comes to a defense-in-depth approach to security of a network, but they still leave open the ability to identify a conversation taking place between a sensor and a server. We have discussed, in detail, how another method to security exists called MT6D that is designed to hide the fact a conversation is even taking place between two endpoints. In the next chapter, we will discuss the establishment of a 6LoWPAN testbed in order to implement MT6D.

Chapter 4

Testbed Creation

Building a physical, Internet-connected testbed involved researching the latest hardware and software environments designed for IoT development. The most critical element of the selection process is that both hardware and software need to be open-source. It also needs to best replicate what is or will be used in the smart home and communicate using IPv6. In this chapter we will discuss the software and hardware we used to replicate a smart home and implement MT6D onto a wireless sensor.

4.1 Software Selection

The software to establish our testbed includes the main operating system for the wireless sensor and border router. We also needed to select software to use for a wireless packet sniffer in order to observe the 802.15.4 traffic. This section explains our selection of the

Contiki operating system and the Foren6 802.15.4 packet sniffer.

4.1.1 Contiki OS

Contiki OS, developed by Adam Dunkels [16], is an operating system for low-powered and resource-constrained devices. It includes a full network stack called μIP that allows embedded devices to communicate with the rest of the Internet using IPv6 addressing. Contiki OS has development support with a fully built virtual machine called Instant Contiki. This VM includes all the compilers and toolchains necessary to compile code for several common embedded devices. The operating system is written in C. It has a socket-like API that applications can use called protosockets. The operating system utilizes a lightweight thread model called protothreads. There are other open-source operating systems such as TinyOS [32] and RIOT [2], but Contiki OS offered a simple development environment including the fact that several products on the IoT market use this operating system such as the LiFX Smart Bulb [4].

4.1.2 802.15.4 Packet Sniffer - Foren6

A wireless network sniffer is required to observe and record the communication exchange between the wireless sensor and the border router. The third Tmote Sky we used runs a 802.15.4 sniffer [9]. When this sniffer is combined with a program called Foren6 [10], we are able to passively capture 6LoWPAN traffic and render the network state in a graphical

user interface. The Center of Excellence in Information Technology and Communication (CETIC) developed Foren6 and the 802.15.4 sniffer. The sniffer program is designed to read in all 802.15.4 frames that it captures and immediately forwards them to Foren6. Foren6 can capture the data and save the observation period as a pcap file for later analysis. Its greatest strength is showing the network visually; representing nodes as circles that contain the last two hexadecimal digits of a node's IPv6 address. When Foren6 observes a packet from a node indicating it is on the network, it will show it as a circle on the screen. If that node forms a route to the border router, it will then connect the circles representing the node and the border router with a line. Foren6 is run on a 24” Apple iMac with 4 GB RAM and a 2.66 GHz Core 2 Duo Processor operating Ubuntu 14.04. The Tmote Sky running the sniffer program connects to the USB port on the back of the iMac.

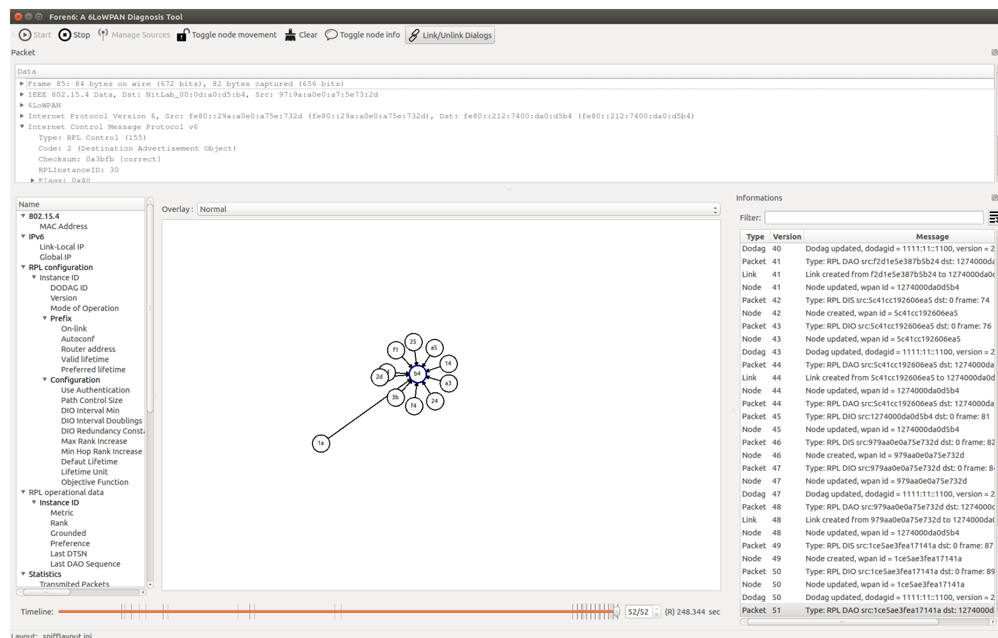

Figure 4.1: Foren6 Screenshot

A sample screenshot is seen in Figure 4.1. The top window of Foren6 is used to display the frame and packet header details including addresses and flags. The bottom-right of the program shows a real-time display of all packets heard. The bottom-center window is where the nodes and border router are graphically displayed. To the left is a more generic breakdown of the packets.

4.2 Hardware Selection

The hardware for our testbed had to be both open-source and representative of a low-powered and resource-constrained wireless sensor. For this research, we define this as a sensor powered by a microcontroller with less than 256kB of RAM that can be battery powered. The hardware must communicate using IEEE 802.15.4 on the 2.4 GHz ISM band. The reason for constraining hardware selection to devices that communicate on the 2.4 GHz ISM band is that this band is approved for use globally. This fact also means that future sensors will also communicate on this band due to reduced cost of producing a sensor with a single frequency range for consumers around the world.

4.2.1 Tmote Sky

The Tmote Sky [42] is a very popular wireless sensor network test platform. It also comes integrated with several sensors such as temperature, pressure, and light. Contiki OS fully supports this platform with several example programs such as *sky-websense.c* which is a

web server that displays the temperature and light sensor data via a web page. This device represents a typical low-powered and resource-constrained wireless sensor. It is powered by an 8MHz, 16-bit Texas Instruments MSP430 microcontroller with 10kB of RAM and 48kB of flash memory. It includes a USB port for ease of connection and rapid programming and reprogramming. The Tmote Sky is used for the first phase of μMT6D implementation by first developing the code necessary to conduct address changes.

4.2.2 Econotag II

The Econotag II [46] is a more capable platform powered by a Freescale MC13224v ARM7 microcontroller with 32-bit processor operating at 24MHz and has 96kB of RAM. The Econotag II also includes a USB port for rapid programming and reprogramming. This platform was used in the development of WigWag [24] which is a planned home automation system. This wireless sensor is more representative of what is currently being used in tech-startup production. The Econotag II is used in the later phase of μMT6D development because of the additional memory it offered that is necessary to include the code that allows for hashing of IP addresses.

4.2.3 Raspberry Pi

We also want the border router to be similar in form factor and processing capabilities of a home Wi-Fi router. In order to establish a connection to the Internet, the 6LoWPAN

wireless network must bridge to a platform that can route these packets. A Raspberry Pi model B [19] is a small computer that best represents what a home border router is like both in size and computing power. It offers 512MB of RAM, a 100Mbps Ethernet Port, and a 700-MHz ARM11 Broadcom BCM2835 processor. This Raspberry Pi runs the operating system Debian Wheezy version 7.8.

4.3 Hardware and Software Integration

The physical testbed needs to best replicate a wireless sensor in a smart home communicating with a server in the cloud. Figure 4.2 shows a network diagram of what this looks like. From right to left we see the wireless sensor communicates with the 6LoWPAN Border Router across the Internet until it reaches the first-hop router of the destination server in the cloud. The 6LoWPAN Border Router and wireless sensor would both be located inside the smart home within line-of-sight range of each other.

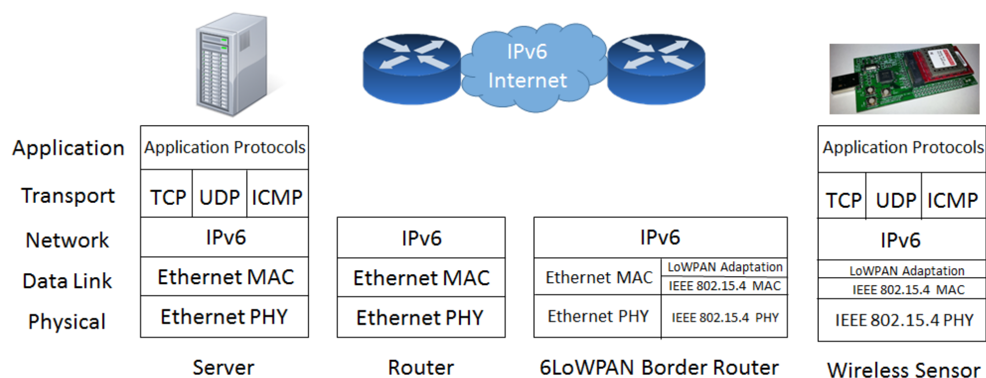

Figure 4.2: IoT Network Diagram

In the development of this testbed and implementation of μMT6D, it is assumed that only one wireless sensor is on the wireless network and communicating with the border router. We also assume no threat to the RF channel from attack and only look at the work needed to change IP addresses while having those addresses added to the routing table of the border router. Although the MT6D protocol could be implemented at the border router (gateway mode), we sought to establish a true end-to-end security solution by running a condensed form of this protocol on the wireless sensor itself (host mode). The establishment of this testbed can be found in Sherburne et al. [53].

Tmote Sky wireless sensors were the first sensors we could obtain in building the testbed. Using Contiki OS version 2.7, we loaded one Tmote Sky with a program called *RPL-Border-Router*. This program allows the Tmote Sky to act as the DAG root node in the 6LoWPAN network. With the Tmote Sky connected to the USB port of the Raspberry Pi, a program called *connect-router* is executed and establishes a tunnel interface, tun0, using the program, tunslip6. The Raspberry Pi's eth0 interface is connected to the Virginia Tech's network that provides both IPv4 and native IPv6 connectivity. We need a routable IPv6 subnet to allow global communication on the 6LoWPAN wireless network. Virginia Tech does not provide routable IPv6 subnets for such cases. In order to have the border router route an IPv6 subnet, we utilized Hurricane Electric's tunnelbroker.net service that assigns /48 IPv6 subnets. An interface labeled *he-ipv6* is created on the Raspbery Pi that terminates a 6in4 tunnel. A 6in4 tunnel [39], establishes a dedicated fixed endpoint for the tunnel and maintains a reliable and easier to troubleshoot link. From here we assign a /64 subnet within the /48 subnet

to the tun0 interface created when we execute the *connect-router* program. When *connect-router* is executed with the network address, the border router will broadcast this prefix to its 802.15.4 wireless network and allow wireless sensors to establish their global-link IPv6 addresses.

In order to test the Internet connectivity of the testbed all the way to the wireless sensor, we loaded another Tmote Sky with Contiki OS example program *Sky-Websense*. This program allowed us to establish TCP communications over IPv6 to the Tmote Sky from across the Internet and access the web page hosted by the web server running on the Tmote Sky. It also allowed us to observe that the Tmote Sky formed its host address based off of its MAC address. This host address is known as the Interface Identifier (IID). In this case, no matter which network the Tmote Sky connects to, it will maintain the same IID and thus could be tracked across the Internet by correlating the last 64 bits of the IPv6 address. This was one major motivation for the authors of MT6D to design a system in which the IID constantly changes.

Figure 4.3 shows the physical testbed used in development of μMT6D. The Tmote Sky wireless sensor on the right communicates with the border router comprised of another Tmote Sky connected to a Raspberry Pi. The Foren6 sniffer is located between the wireless sensor and the border router.

We have now described a testbed that, from the open-source perspective, represents an Internet-connected smart home wireless sensor network. We chose an open source operating system, hardware, and communications standards in order to have maximum flexibility in

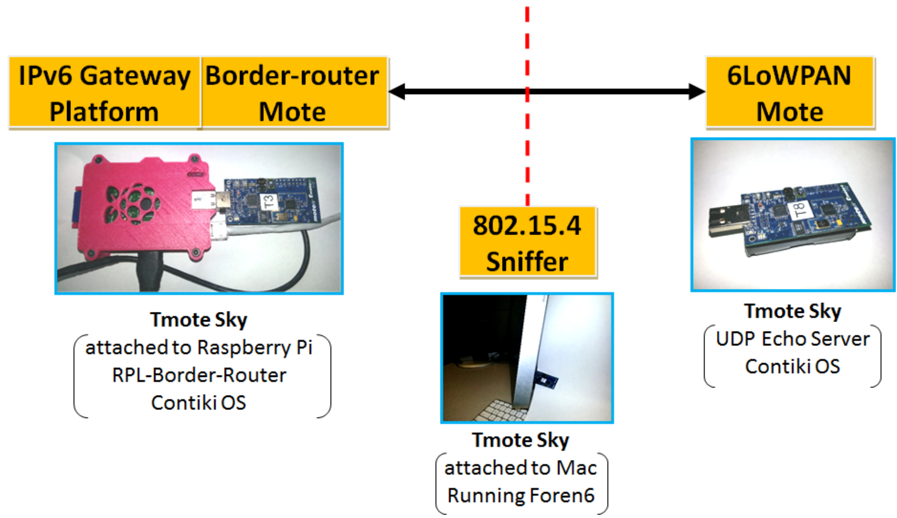

Figure 4.3: Physical Testbed

development and implementation of μMT6D. In the next chapter we will present our work in implementing μMT6D onto a wireless sensor.

Chapter 5

Implementing MT6D Functionality in Contiki OS

In this chapter we will explain how to implement the MT6D protocol, in a reduced form, on our testbed that we have just described in the previous chapter. We first will describe a flowchart that displays the overall functionality of μMT6D before describing each step in further detail. Figure 5.1 shows the steps necessary to carry out MT6D functionality on a wireless sensor. We begin with the boot up sequence of the wireless sensor followed by the initialization and transmission of its Layer 2 and Layer 3 addresses. Next, a process that tests whether the wireless sensor has found a border router will then start NTP synchronization if a border router is found or else the process will wait until the wireless sensor does link with a border router. Another process will test whether valid NTP time is synchronized before the hashing of addresses begins. If no valid time exists on the sensor, the process will hold

until another NTP synchronization takes place. With the successful synchronization of NTP time, a new source and destination IPv6 address pair is calculated with the *mt6d_hash()* function. Next, the IID of the new source address is passed to the *set_global_address()* function. This function builds the new Layer 3 and Layer 2 addresses of the wireless sensor and then advertises them to the border router. An event timer, called an etimer, is set for a specific address change interval. If the timer expires, then a new source and destination address pair are recalculated. If the timer has not yet expired, the sensor will check for any outbound or inbound traffic.

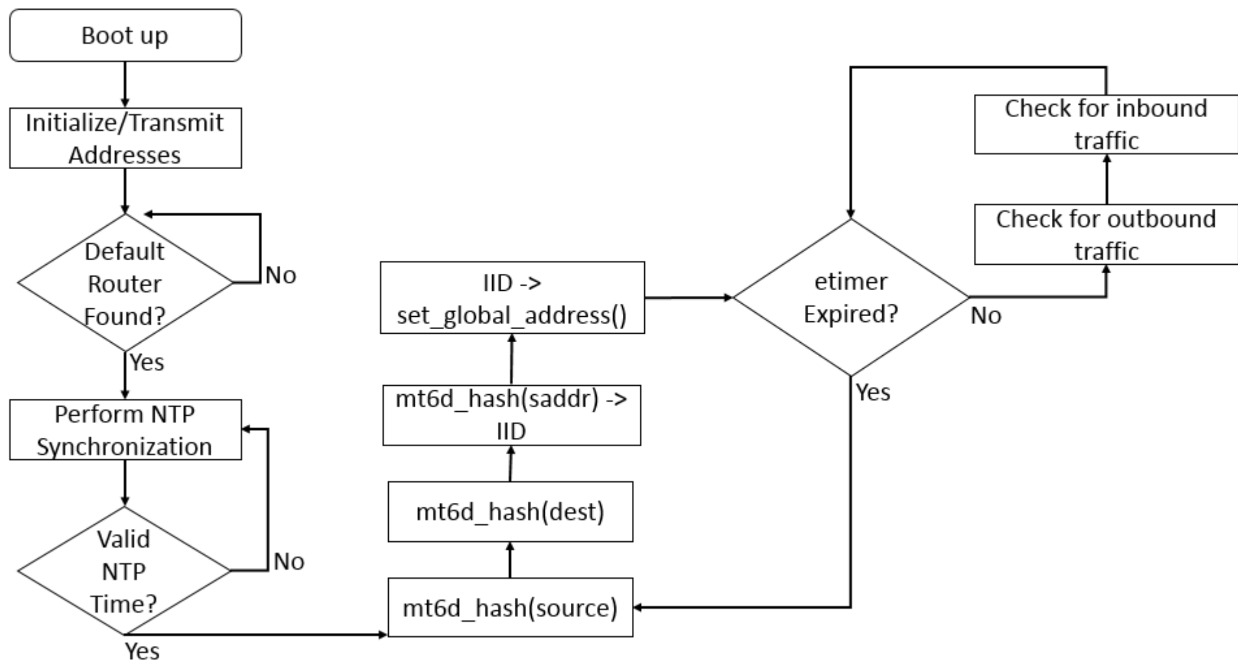

Figure 5.1: μMT6D Flowchart

Now that we have described the high-level functionality of μMT6D, we will next describe the neighbor and routing control messages used in 6LoWPAN and RPL. Since we are working

with a wireless ad-hoc mesh network, we must use the existing RPL routing protocol in order to perform address advertisements of the new address bound on the radio interface of the wireless sensor to the border router. We will discuss this control message exchange process in our next section.

5.1 ICMPv6 RPL Control Messages

RPL control messages specified in RFC 6550 [57] are a new ICMPv6 message. The base ICMPv6 messages are specified in RFC 4443 [8]. There are three main control messages: DODAG Information Object (DIO), DODAG Information Solicitation (DIS), and Destination Advertisement Object (DAO). These are important and relevant to the implementation of MT6D in the wireless sensor because they update address information to the wireless network and the border router. We will now better explain the purpose of each of these messages. A DIO control message, much like a neighbor advertisement, contains information that allows other nodes on the wireless network to discover and learn the configuration parameters of an RPL Instance, select a DODAG parent set, and maintain the DODAG. These messages are constantly being sent from RPL nodes at controlled time intervals. Within Contiki OS, *rpl-conf.h* defines a variable *RPL_DIO_INTERVAL* to default 12. The minimum interval is defined as 2^n milliseconds. With our default set to n = 12, a DIO packet will be sent every 4.096 seconds. With this default set for every node, this means a DIO will be sent approximately every 4.096 seconds while there is activity from other nodes on the network. This will

be a factor to consider when trying to send address updates from one node and the border router sending a DIO packet at this interval. The second control message, DIS, is much like a neighbor solicitation packet and is used to solicit a DIO (neighbor advertisement) from another node. A DIS is also analogous to a Router Solicitation in IPv6 Neighbor Discovery. Any node in a wireless sensor network can act as a host or a router and is why this type of solicitation is used. The final control message, DAO, is used to conduct an IPv6 address advertisement to the DAG root, better known as our border router. This message is required to store the wireless sensor's global link address in the border router's route table. The DIO and DAO pair of messages are important in maintaining and establishing a global route. The packet diagram in Figure 5.2 shows the baseline operation of ICMPv6 RPL control messages in order to establish neighbors and routes in 6LoWPAN.

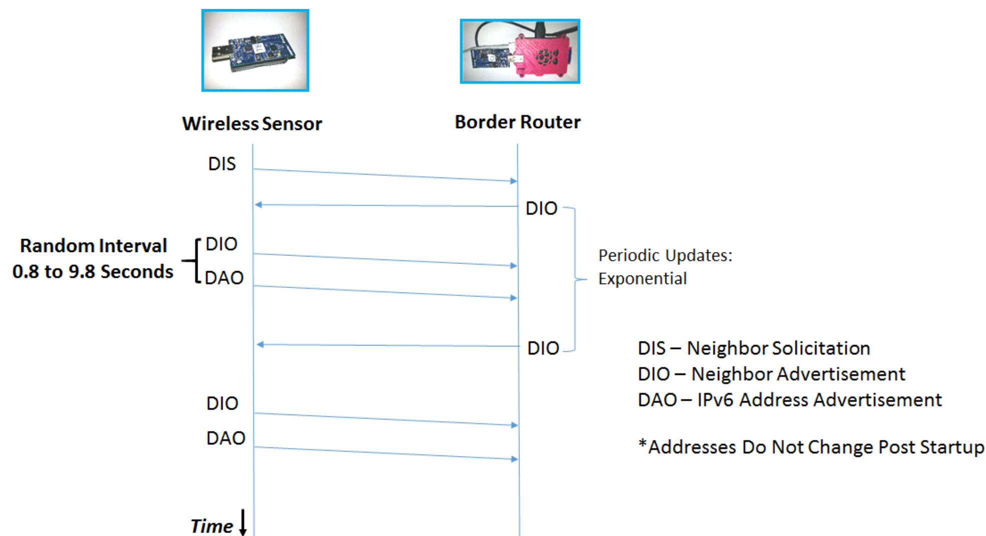

Figure 5.2: Base Configuration RPL Packet Exchange

The wireless sensor boots up and first sends a DIS neighbor solicitation packet in order to

request neighbor information from other nodes on the wireless network. The border router responds with the DIO neighbor advertisement packet that contains the prefix address of the IPv6 network and the fact it is a border router. The wireless sensor then sends back a DIO neighbor advertisement packet and DAO IPv6 address advertisement packet that provide the border router with the wireless sensor link-local and global-link addresses. Note that the DIO neighbor advertisement destined for multicast so other nodes that may be present on the network may also add the wireless sensor to their neighbor table. The DAO IPv6 address advertisement packet is sent unicast to the border router directly.

We have now explained the necessary control messages that handle neighbor and routing updates to the 6LoWPAN. These are important to understand because they are used to update the border router with the new address information on the wireless sensor performing μMT6D. Next we will discuss how we implemented dynamic address changes and explain why it is a necessary component of implementing MT6D within 6LoWPAN.

5.2 Dynamic Addressing Changes

In order to implement MT6D, we must establish a function that can set a new address and bind it to the wireless sensor's radio interface. That address must also be advertised to the border router so that incoming packets destined to the new address will have a route established. Since 6LoWPAN, by default, does not query the wireless network for who has an inbound packet destination address that is not in the routing table, communication can

only continue if the border router has the updated address already in its routing table. It was our decision not to make modifications on the border router with this implementation as we were investigating only implementation of MT6D on the wireless sensor. The research conducted by Preiss et al. [43] identified the critical files, methods, functions, and global variables required to establish dynamic address changes within Contiki OS. Wireless sensors, by default, only set their MAC, link-local, and global-link addresses once upon boot up and discovery of their DAG root. There is no need to change their address again. μMT6D, on the other hand, requires that the sensor be able to establish a new address and advertise that address to the border router.

Figure 5.3 shows the basic flowchart of the *set-global-address.c* file that is executed when we call *set_global_address()*. We first start by passing, as a parameter, the new 64-bit IID. From here, we perform the EUI-64 conversion followed by removing the old addresses from the array and next adding the new link-local and global-link addresses to that array. We then set the Layer 2 addresses and send the RPL messages needed to advertise our new addresses to the border router. This is the general flow of *set-global-address.c*. We will explain in more detail how each of these steps are carried out.

We will now describe what these wireless sensors do by default upon boot up in order to establish an address on the network. The code in *contiki-sky-main.c* starts by performing a conversion to a global array labeled *ds2411_id*. This conversion changes the 802.15.4 MAC address to a compatible EUI-64 bit address [26]. This variable, for the Tmote Sky, forms the basis to which all addresses are created. This global array contains a unique value, once

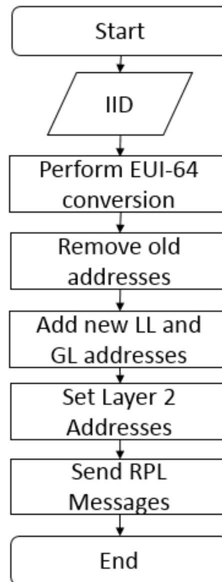

Figure 5.3: Address Change Function Flowchart

initialized, that derives from the Tmote Sky hardware specifications including family, type, and *node_id*. The Tmote Sky uses *ds2411_id* as its MAC address since it is unique to an individual node and does not change. After the *ds2411_id* EUI-64 conversion is performed, a local method, *set_rime_addr()*, performs a memcopy of *ds2411_id* and sets the Rime address by using *rimeaddr_set_node_addr(&addr)*. The Rime stack is a lightweight network stack that can be used instead of the full IPv6 stack. This method, *set_rime_addr()*, is important as it is the first method that relies on *ds2411_id*. We can begin building our own dynamic address by inserting our address into this method instead of it reading from *ds2411_id*.

The next method explained, *cc2420_init()*, is specific to the 802.15.4 radio used on the Tmote Sky and is used to initialize it. The method sets the radio's PAN (Personal Area Network) address to the Rime address. The CC2420 radio acknowledges all inbound packets'

destination addresses by performing a cross-check of its PAN address with the destination of the packet. Next, a memcopy of the *ds2411_id* into the global variable, *uip_lladdr.addr*, is performed. The *uip_lladdr.addr* variable contains the IPv6 link local address. A global array called *uip_ds6_if* stores the global and link local addresses. Once the Rime and PAN addresses are initialized, *uip_lladdr.addr*, the link local address, is added to this *uip_ds6_if* array. This address is used as the source address of the first DIS, or neighbor solicitation, control message that is sent upon initialization of the Tmote Sky. In response to this DIS, neighboring nodes will send back a DIO which contain the information about the wireless network. The DIO from a border router will contain the IPv6 network prefix address. If the node receives a DIO from the border router, it then adds this prefix to a global array of prefixes. Then the node uses the received prefix to create a full global address by combining the prefix with its link local address. This newly formed global link address is then added to the same global array, *uip_ds6_if*, that just recently stored the link local address. With a global address formed, the Tmote Sky will now send a DAO, or IPv6 address advertisement, control message back to the border router to indicate that this is the address it wants as a global address in the border router routing table.

Using the above initialization information, we were able to replicate this process in order to set new addresses at specified intervals. Working in the memory constrained environment of the Tmote Sky, we first produced new addresses by simply iterating an address to increase the value of its last octet by 0x01. This was a simple way to change addresses and ensure the code was working properly. We also tested that specified address change intervals, one

through ten seconds increasing in one second intervals, were being executed. These address change intervals were selected during initial testing which showed a 100% global address route addition in the border router at 10 second interval changes. We then evaluated system performance by increasing the rate of address changes to see how quickly we can send address updates on the wireless network.

We set, in a method, the IID, or last 64 bits of the IPv6 address via eight, 8-bit unsigned integers with the network address prefix hard coded for use in forming the global address. This was used in initial testing until a more suitable alternative can be found to pull the prefix from the DIO packet of the border router. We then convert the IID to make the address Ethernet compliant. This is done by bitwise XOR bits 40-47 with 0xfe and changes the 7th bit to 1 to be Ethernet compliant. We now remove the previous link local and global link addresses from the global array mentioned previously that stores the addresses for the node. We changed the global array that stores the addresses for the node and mark the addresses as not used; mirroring the behavior of *uip_ds6_addr_rm* in order to reduce code. These array positions will now be overwritten upon a following add into the array. If we had not done this, then adding a new set of addresses would have failed rather than overwrite.

Now we add the new address into the *uip_net_if* list and set the global variables mentioned previously. We create a *uip_ipaddr_t* struct for the local and global addresses and initialize them using the *uip_ip6addr_u8()* method. This method takes, as a parameter, a 128-bit number broken into sixteen, 8-bit numbers. We then use *uip_ds6_addr_add()*, that takes as parameters: the IP address, lifetime, and address type. We set the lifetime to 0 to make

it infinite. The lifetime could be set to a specified interval, but because we are changing addresses that will cause an old address in the routing table of the border router to be overwritten, it was not necessary at the time to find an optimal interval. The address type is set to ADDR_PREFERRED since this is the same type used in the initialization of the first address. Other address types include ADDR_MANUAL and ADDR_TENTATIVE. Next, we perform the bookkeeping of global variables in which we set the Rime address, CC2420 PAN address, and global *uip_lladdr.addr*. This allows any address that will be used throughout the messaging and stack protocols to be updated to the new address. A 64-bit Rime address struct rimeaddr_t named *addr* is created and its values are set by accessing the address as an 8-bit array. This is performed by accessing the rimeaddr as *addr.u8[i]* where i is the index, from 0-7, to change the Rime address to the new address. We set the Rime address using *rimeaddr_set_node_addr()*. Then we set the CC2420 PAN address by copying the Rime address and using *cc2420_set_pan_addr*. Our last bookkeeping of global variables consists of setting up *uip_lladdr.addr* by using memcopy to copy the newly set Rime address into this structure.

Our final task is to send the DIS, DIO, and DAO control messages now that our new address is set. DIS and DIO control messages are sent to the destination multicast address of ff02::1a. The DIO control message will add the link local address to the neighbor table of the border router. The DAO, which is sent directly to the address of the border router, will add the global link address to routing table of the border router. We use *dio_output()* and *dao_output()* in order to send the DIO and DAO control messages. Initially we called

dio_output() only and let the DAO control message be sent via a built-in timer that calls *dao_output()*, but that timer was designed to be randomized between 0.8 and 9.8 seconds based on our analysis. This means that the next address changes before the DAO can even be sent if we have an address change interval faster than ten seconds. You can see the impact this timer has on the successful addition of addresses per address change interval in Figure 5.4.

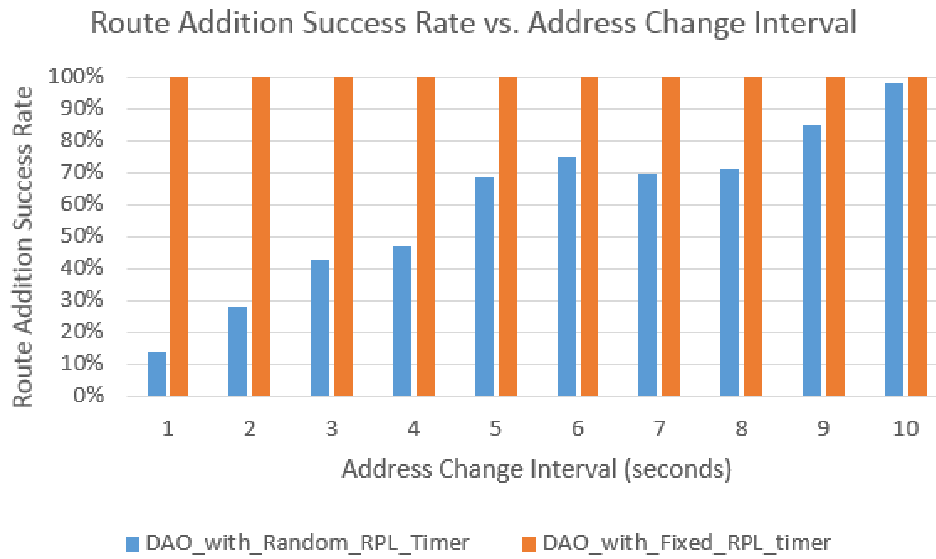

Figure 5.4: Route Addition Success Rate

It was necessary to disable the DAO delay timer call found in *rpl-timers.c* and manually call *dao_output()* directly after *dio_output()*. The purpose of the random delay timer is to decrease contention on the RF channel since it is assumed there are multiple wireless sensors on the network. This is the one change we made to RPL from the standpoint of the wireless sensor. In order to implement MT6D functionality, we had to send address change control

messages immediately with no randomized delay.

The packet exchange that occurs based on this new dynamic address change method is shown in Figure 5.5.

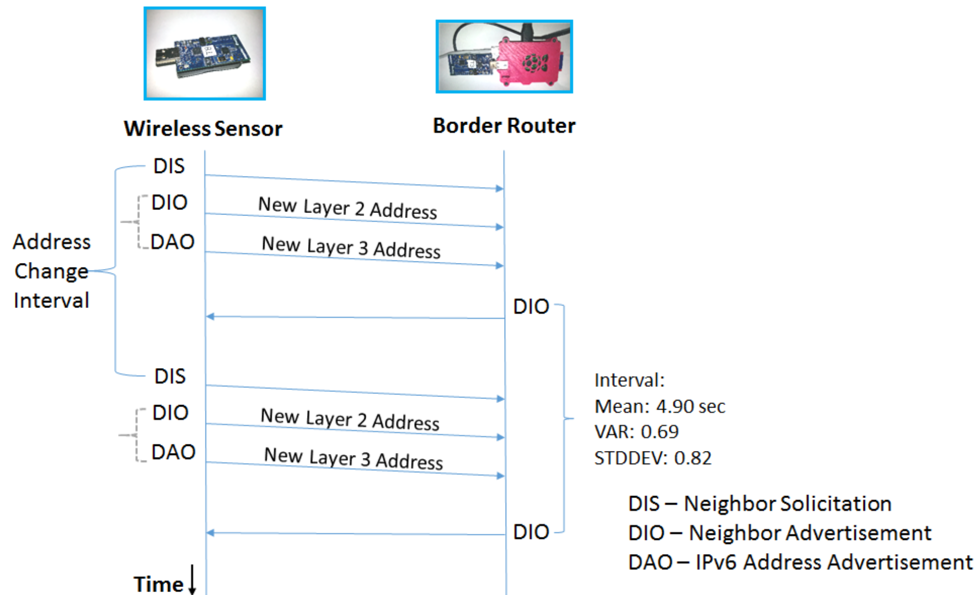

Figure 5.5: RPL with Address Changing

This exchange now shows the wireless sensor changing addresses based on the address change interval with DIO and DAO control messages sent in pairs. Initially, when we called both functions, one right after the other, we had unsuccessful route additions. We found that the DAO packet was attempting to send before the DIO was finished transmitting. A 0.3 second delay was established in order to provide enough delay between the completion of the DIO packet being sent and the start of transmission of the DAO packet. This delay was selected because it was the minimum delay needed for us to achieve an approximate 99% to 100% route addition success rate.

Upon completion of implementing the dynamic address change code, *set-global-address.c*, we reached the near limit of the memory on the Tmote Sky. Another wireless sensor platform was needed with more memory in order to add the address hashing algorithm libraries and requisite code. For this reason, research was moved to the Econotag II platform. Due to the network stack of Contiki OS and the way addresses are changed and set in our code, *set-global-address.c*, only one minor change was made to the code in order to use with the Econotag II. We commented out *cc2420_set_pan_addr* since the Econotag II does not have this radio. The same tests as performed with address changes on the Tmote Sky were performed on the Econotag II. Having achieved the same successful results, we continued work on implementing the use of hashing algorithms to obtain the new IPv6 address just as in MT6D. Next we will discuss establishing the timestamp that MT6D requires for its address hashing function through the use of the Network Timing Protocol.

5.3 Establishing NTP

The timestamp parameter in the *mt6d_hash* function in *addr_gen.c* that is used in the calculation of the next IPv6 address is the Network Timing Protocol time [34], which uses Unix time. Unix time was chosen by the creators of MT6D because it is a value that can change on both ends that is accurately synced across the Internet on both hosts running MT6D. This means that hosts on each end of the Internet can calculate matching addresses and necessary in order for communication to be successful.

Consideration was made on whether NTP should be used in the implementation of μMT6D on an embedded device. There needed to be a way for a value to constantly change that is processed accurately on both ends of the communication. The only logical choice is to implement a version of a NTP client that can run on embedded devices and within the framework of Contiki OS. After researching for NTP clients that were designed to be run on Contiki OS, two candidates were found and evaluated. The first NTP client is by Josef Lusticky [33]. His implementation is specifically designed for the AVR Raven embedded platform with precision timing. Due to this fact, the code was written in such a way that it was only compatible for AVR devices and the specific timers their architecture utilizes. There was not a practical way to port this code for use with either the Tmote Sky or the Econotag II.

The second NTP Client, by Anuj Sehgal [51], was developed for use on any generic device that Contiki OS supports. It utilizes the Simple Network Timing Protocol (SNTP) [35]. Although not precise on the order of nanoseconds, it does provide a means to establish NTP on the sensor. This client has the option to request NTP time either from the border router that the sensor discovers or a set address to a NTP server. The advantage of selecting the border router that the sensor discovers is that the time is more accurate due to lower delays than if requesting time from a distant server. Requesting time from a distant server also allows mobility, but its address could potentially be blocked by the local network access control list.

For implementation of SNTP into μMT6D, the tun0 IPv6 address of the Raspberry Pi is used

because *rpl-border-router.c* did not include code in order to establish the sensor connected to the Raspberry Pi as a NTP server. The code in the NTP client running on the wireless sensor is set to find the wireless 802.15.4 address of the border router and use that as the NTP server address. This obviously presented an issue and is why the IPv6 address of the tun0 interface is used. As stated before, the interface, tun0, is created when connect-router is run on the Raspberry Pi which establishes a Serial Line Internet Protocol (SLIP) connection [50] with tunslip6 between the Raspberry Pi and the Tmote Sky through the USB port, in this case, /dev/ttyUSB0. The Raspberry Pi, to be a NTP server, has to have its NTP daemon running in order to pull NTP time from a higher stratum server and to serve this time to requests from the NTP client running on the wireless sensor. It is important to note that the appropriate security should be in place on the Raspberry Pi in order to ensure that NTP port 123 is not accessible from the outside Internet. Upon boot up of the wireless sensor, Contiki OS sets its initial addresses and begins to establish connectivity with the border router. A control measure is added to the SNTP client code that delays the client from sending an NTP time synchronization UDP packet until the wireless sensor has a default route established that is indicated by *uip_ds6_defrt_choose()* being true. This test allows time for the global link address of the wireless sensor to be added into the routing table of the border router. With this route added, the NTP server can now successfully send the NTP time packet to the wireless sensor. Another necessary control measure added is to perform a test of whether valid NTP time is now being kept on the wireless sensor before starting the address change process. This is implemented using a simple *if* statement that

tests whether *getCurrTime()*, the function that returns the Unix time value, is not equal to 0. Once the wireless sensor synchronizes its NTP time with the server, it will no longer have a value of 0 and then proceed to change addresses as often as the interval is set. This NTP client will also periodically re-synchronize its time with the NTP server. With our wireless sensor now able to maintain time using NTP, we will now explain in the next section how we implemented the MT6D hashing function using the SHA256 hashing algorithm in order to calculate the next source and destination address information.

5.4 Performing Hashing of New IPv6 Address

In order to perform hashing of the new IP address on the wireless sensor, some work was conducted in porting the MT6D file, *addr_gen.c* [37], to work correctly with Contiki OS. The original file assumed functionality on a full Linux OS that could perform its buffer memory initialization and allocation. Contiki OS' memory management is contained in a file called *mmem.c*. Here the memory size is defined, default to 4096B, and includes the initialization, allocation, and memory free functions. These are used to establish a memory heap in *addr_gen.c* in order to process the hashing of the address and return a new address. Many different hashing functions can be used with MT6D, so long as they are the same ones used on each end of the communications. We decided to use SHA256 by using a version of SHA2 written in C [20]. This decision was based on the fact that Anuj Sehgal also developed TLS and DTLS clients [52] for use with Contiki OS and they also utilized this version of

SHA2. Additionally, this same hashing algorithm is used with MT6D.

Since the wireless sensor's address change function requires a 64-bit value - the Interface Identifier (IID), the last 64-bits of the returned hashed IID is copied into the IID array. This value is then passed to the *set_global_address* function in order to update the link local and global link addresses and send them to the border router to be added to the neighbor and routing tables.

In this chapter we have discussed how we used the ICMPv6 RPL control messages utilized in 6LoWPAN in order to send address updates to the router. We covered how we implemented dynamic address changes and the functions and methods necessary to build new addresses both at Layer 2 and Layer 3. Then we showed how the border router was successfully storing these new addresses from the wireless sensor in its neighbor and routing tables. We then discussed how we were able to integrate a NTP client that allows us to maintain a synchronized network time for use as our timestamp parameter for the MT6D hashing function. Finally, we discussed how we integrated the SHA256 hashing algorithm and memory management so that the MT6D hashing function can successfully calculate new source and destination address information. In our next chapter, we will evaluate the performance of μMT6D in order to assess the viability of implementing MT6D on a wireless sensor.

Chapter 6

Evaluation of μMT6D

In this chapter we will evaluate μMT6D on its performance based on three parameters: address creation and route addition success rate, binary file size, and average current consumption. The results of these parameters will allow us to draw a conclusion on the viability of using MT6D on a wireless sensor. We will explain the methodology of the testing for each of these parameters followed by the results of the tests.

6.1 Methods

We will discuss in this section the methods used to measure address creation and route addition success rate, binary file size, and average current consumption and why we chose these methods. These methods were developed as the baseline test measures to assess performance of μMT6D. They will provide a satisfactory baseline to which we can draw conclusions from

the data results.

6.1.1 Address Creation and Route Addition Success Rate

It is important to assess that the wireless sensor is able to compute the SHA256 hash of the next source and destination addresses. It is equally important that the wireless sensor can also advertise the source address to the border router so that the border router can properly route inbound packets destined for that sensor. The experiment performed in [43] was performed again, but this time with addresses being calculated with the SHA256 hashing algorithm function, *mt6d_hash* instead of a simple 0x01 increment of the last octet of the IID. In order to test the full capabilities of the Econotag II, the hashing function was also calculating the destination address and destination port addresses for the same time interval as the source. This test is designed to show that the wireless sensor can handle calculating both source and destination address information and advertising its address to the border router. In addition to this test, we want to also make measure the amount of code size that μMT6D adds.

6.1.2 Binary File Size

We want to assess the size of μMT6D in terms of binary file size. This, at a minimum, shows the amount of RAM, in this case code space, that is required on the Econotag II. We can then compute the percentage of the loaded binary file dedicated to μMT6D. The Econotag

II has a maximum RAM size of 96kB which it uses for code execution. This low memory size is why it is relevant to conduct this measurement. Next we will discuss our testing methodology for measuring the average current consumption.

6.1.3 Average Current Consumption

As discussed earlier, most devices on the Internet of Things are powered by batteries. It is important to assess what additional current is consumed by implementing μMT6D. There are several methods in which to collect this data. Contiki OS has a software-based calculation program called Powertrace [15]. The program calculates the time a device spends using its CPU, operating in a sleep state, sending data, or receiving data. It can then approximate the average current consumption based on how long the device spends in each of these states. This program is unfortunately not compatible with the Econotag II as of this research. We proceeded to measure the average current consumption of a device running μMT6D and compare it to a device in baseline configuration.

The baseline configuration, called the control configuration for this experiment, is the wireless sensor not running μMT6D. The baseline program is from the Contiki OS example called *udp-echo-server.c*. This is the same code base used for implementation of μMT6D. In order to sample a sufficient time period of data, we will record the average current consumption over a time period of five minutes. The average current consumption of the control configuration sensor will be recorded along with the device running μMT6D operating at address change

intervals of one through ten seconds in one second increments. The purpose of this evaluation is to observe the average current consumption of μMT6D running at different address change intervals to then draw a conclusion on whether there is an optimal configuration of μMT6D with respect to average current consumption.

We utilized the shunt resistor method to record the average current consumption of the Econotag II. With this method, the Econotag II, shunt resistor, and power source are wired as seen in Figure 6.1.

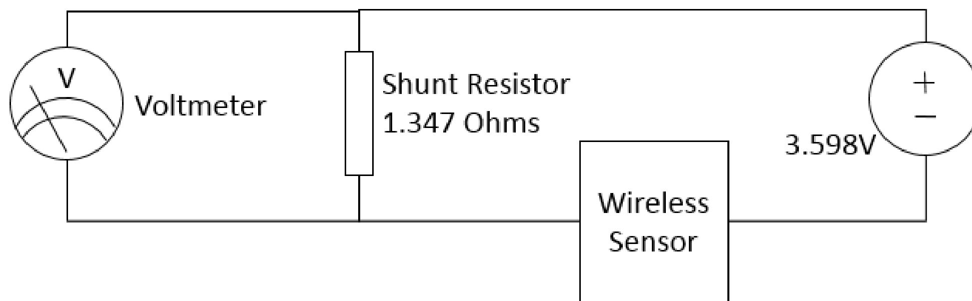

Figure 6.1: Average Current Consumption Measurement

We then measure and sample the voltage drop across the shunt resistor. In order to find the current from the sampled voltage at time, t, we use Equation 6.1.

$$I = V/R_{shunt_resistor} \tag{6.1}$$

To measure this voltage drop, we used a Mooshimeter data acquisition multimeter with 0.5% accuracy. It has a 24-bit ADC and can process 256 samples per second. A 1 Ohm resistor is used as the shunt resistor. A Hewlett Packard 3478A Multimeter was used to measure

the resistance of the shunt resistance and accurate to approximately 0.1%. The resistor was measured at 1.347 Ohms. To power the Econotag II, we have to use a power source separate from the USB port of the computer that programs the Econotag II with the binary file. For our testing, we used a Hewlett Packard E3631A Triple Output DC Power Supply. The output voltage was set to 3.60 volts and measured at 3.598 volts in order to replicate the typical battery voltage that would be used to power the Econotag II. One such example of a 3.6 volt battery is the LAA 3.6 volt Lithium-Thionyl Chloride (Li-SOCl2) battery. These batteries have a typical capacity of 2600mAh. This fact will be used in the results to show the approximate lifetime of a battery when used to power a device. We will use Equation 6.2 in order to show approximately how many hours and minutes a battery can last when executing the control versus μMT6D.

$$Lifetime[H] = Battery_Capacity[mAh]/Current_Draw[mA] \tag{6.2}$$

We will only account for the effects of the address hashing process and RPL control overhead on the average current consumption. No additional data communications between the Econotag II and the border router are performed. The radio transmit power is set to 0 dBm.

6.2 Results

In this section, we report our results using the above methods. These results will provide the data necessary to conclude whether the implementation of MT6D host mode onto a low-powered and resource constrained wireless sensor running Contiki OS in a 6LoWPAN

is viable. This data includes the ability to calculate both source and destination address information and for the wireless sensor to advertise the new source address to the border router. It also includes data on the size of the control configuration binary file in comparison to the μMT6D binary file. Last we provide the results of the average current consumption of the control configuration and μMT6D.

6.2.1 Evaluation of μMT6D Hashing Process

We now evaluate the performance of the μMT6D hashing and address advertisement process. We present the success rate of the border router storing the wireless sensor address advertisements in its routing table. With this address in the routing table, an inbound MT6D packet can be further routed to the wireless sensor. Figure 6.2 shows the results of this test with address change intervals ranging from one to ten seconds.

One second address change intervals averaged a 95% route addition success rate for one or two hashing processes. The reason for this rate is the packet collisions when the border router's DIO neighbor advertisement packets are sent. This fact is also true for the other address change intervals. There were test runs at one second address changes that caused the sensor to halt. This was later attributed to the numerous printf's being sent to the screen used in earlier troubleshooting of the code. Those printf's were removed for all but the ones that indicated what IPv6 address the sensor was currently bound. This then allowed the sensor to function properly. It is important to note that 100% of the IPv6 link-local neighbor

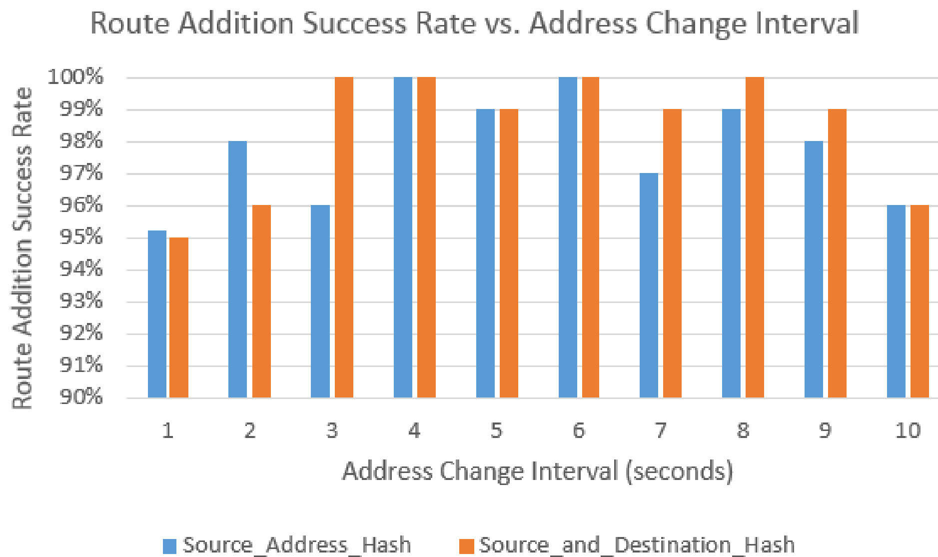

Figure 6.2: Route Addition Success Rate with Hashing

addresses are being added at all address change intervals with the border router. In order for the sensor to initialize addresses with the border router to conduct NTP synchronization before the start of the μMT6D process, *rpl-timers.c* was configured so that DIS, DIO, and DAO packets were sent. As the *set_global_address()* function sent DIO and DAO packets, *rpl-timers.c* was also sending DIS and DIO packets for that new address. These extra packets contributed to the occasional loss in DAO packets that led to a 95% route addition success rate.

6.2.2 Binary File Size

The file sizes are listed in the table below. μMT6D adds 2948 bytes to the base code. This leaves 37584 bytes available for additional code or memory usage.

Table 6.1: Comparison of Binary File Sizes.

	File Size [B]
Control Configuration	55468
μMT6D	58416

6.2.3 Average Current Consumption

The below table shows the average current consumption over the course of five minutes. The average is taken across 1950 current measurements for each test. The margin of error is $\pm$ 0.6% A or, in the case of the control, $\pm$ 98 μA. The battery lifetime is calculated from Equation 6.2.

Figure 6.3 shows a graphical representation of the average current consumption. There is a clear increase in current consumption between the control at 16.274mA and μMT6D at its maximum address change interval of one second, 16.346mA. This difference, however is not as large as we first expected. With an increase in packet transmission to update the address change to the border router, it was first thought that the difference would be on the order of mA and not μA. The explanation comes from the fact that the power required to receive and to transmit, at least at 0 dBm, is nearly the same. Instead of the radio mostly in receive mode as with the control code, we are now replacing those time periods with transmission. The extra current consumed then comes from processing the hashing of new addresses with

Table 6.2: Comparison of Average Current Consumption.

	Current [mA]	Battery Lifetime
Control	16.274	159H 46M
μMT6D 1 second	16.346	159H 4M
μMT6D 2 seconds	16.333	159H 11M
μMT6D 3 seconds	16.311	159H 24M
μMT6D 4 seconds	16.308	159H 26M
μMT6D 5 seconds	16.292	159H 35M
μMT6D 6 seconds	16.294	159H 34M
μMT6D 7 seconds	16.297	159H 32M
μMT6D 8 seconds	16.292	159H 35M
μMT6D 9 seconds	16.301	159H 30M
μMT6D 10 seconds	16.298	159H 31M

SHA256. The difference of μA is within our error of margin, but we can still conclude that the addition of μMT6D does not greatly increase the average current consumption.

With our results now presented and described, we can form a conclusion as to the efficacy of implementing a reduced form of MT6D onto a wireless sensor from the perspective of control overhead. This is important in understanding how effective this security scheme performs under operation on a low-powered and resource-constrained sensor. As these sensors are

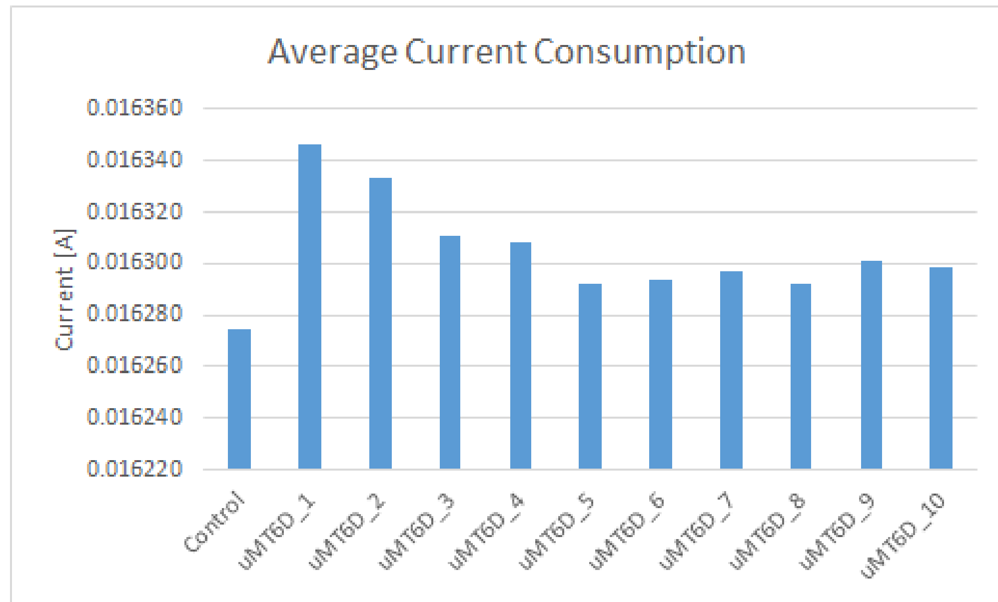

Figure 6.3: Average Current Consumption

battery powered with a small amount of memory and communicate over a low-powered and lossy network, we have to keep in mind whether the security scheme we implement will, in the end, cause a greater impact on resources than the attacks this security scheme was designed to thwart. In our last chapter we will conclude our findings and offer suggestions for future work.

Chapter 7

Conclusion and Future Work

We have provided the background to the Internet of Things and potential security risks involved with the communications between sensors and servers. We described a specific subset of the IoT that use IEEE 802.15.4 and 6LoWPAN to communicate. We presented a testbed that implements this network and the work conducted to implement MT6D onto a wireless sensor. We reported our results and are now ready to conclude the viability of using MT6D within 6LoWPAN and the Internet of Things.

7.1 Conclusion

The MT6D protocol can be implemented in reduced form for use with embedded devices that run Contiki OS. Our results show that wireless sensors with similar specifications as the Econotag II can successfully compute the next source and destination address and port

information. The results also show that a wireless sensor's new address can be added to the border router's routing table with 95% reliability. μMT6D adds less than 1 mA of energy consumption to a wireless sensor. μMT6D adds only a 5.3% increase to the binary file size. This proposed version of μMT6D currently requires excessive control overhead in order to ensure the border router maintains a route to the wireless sensor so that inbound packets can be routed successfully to the sensor. This overhead is due to our design decision to only manipulate code and protocols on the sensor without needing to make changes at the border router. The excessive overhead has a potentially negative impact for the communications of other sensors that are on the same wireless network that need to communicate. Our implementation not only changes a sensor's global link address, but also the IPv6 link local and MAC addressing. This means that a sensor can completely change its identity at Layer 3 and Layer 2. An observer to the layer 2 traffic will not be able to follow a conversation taking place. Had we implemented MT6D only at the border router, then we can still see the conversation take place with access to layer 2. The last aspect of this implementation we would like to discuss is key management. μMT6D in its current version uses the same symmetric key system as MT6D. This will not scale to the intended size of the IoT. Next we will discuss suggestions for further work.

7.2 Future Work

Future work should consist of analyzing the impact μMT6D has on the successful transmission of deterministic and non-deterministic sensor data under varying address change intervals. It is also important to assess the impact μMT6D will have on RPL when multiple wireless sensors are present on the same RF channel. The RPL standard, by default, does not require the border router to handle inbound packets with unknown destination addresses. This means that when a server has traffic for a wireless sensor addressed with a global link address not in the border router's routing table, the router drops the packet. In order to reduce control overhead within the wireless network, an assessment should be conducted in whether a border router can handle inbound MT6D traffic with unknown destination addresses by using a multicast Neighbor Solicitation to query the wireless network if anyone is bound to that address. In order to scale μMT6D to the size of the described IoT with respect to key management, future work should also consider implementing the work of Morrell et al. [38] in order to enable MT6D communication between a large number of sensors with a respective server.

Bibliography

[1] I. S. 802.15.4-2006. Wireless medium access control (mac) and physical layer (phy) specifications for low-rate wireless personal area networks (lr-wpans), October 2006.

[2] E. Baccelli, O. Hahm, M. Gunes, M. Wahlisch, and T. C. Schmidt. Riot os: Towards an os for the internet of things. In *Computer Communications Workshops (INFOCOM WKSHPS), 2013 IEEE Conference on*, pages 79–80. IEEE, 2013.

[3] I. Bojanova. Defining the internet of things. http://www.computer.org/web/sensing-iot/content?g=53926943&type=article&urlTitle=defining-the-internet-of-things&lf1=8501255974c686016060658c39045314, 2015.

[4] P. Bosua. Lifx: The light bulb reinvented. http://www,kickstarter,com/projects/limemouse/lifxthelight-bulb-reinvented, 2012.

[5] T. Clausen and P. Jacquet. Optimized Link State Routing Protocol (OLSR). RFC 3626 (Experimental), Oct. 2003.

[6] U. F. T. Commission. Internet of things: Privacy and security in a connected world. https://www.ftc.gov/system/files/documents/reports/federal-trade-commission-staff-report-november-2013-workshop-entitled-internet-thing 150127iotrpt.pdf, January 2015.

[7] O. I. Consortium. Open interconnect consortium. http://openinterconnect.org/, 2015.

[8] A. Conta, S. Deering, and M. Gupta. Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification. RFC 4443 (Draft Standard), Mar. 2006. Updated by RFC 4884.

[9] S. Dawans. Sniffer 15.4. https://github.com/cetic/contiki/tree/sniffer/examples/sniffer, 2013.

[10] S. Dawans and L. Deru. Troubleshooting with foren6. https://github.com/cetic/foren6, 2011.

[11] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460 (Draft Standard), Dec. 1998. Updated by RFCs 5095, 5722, 5871, 6437, 6564, 6935, 6946.

[12] T. Denning, T. Kohno, and H. M. Levy. Computer security and the modern home. *Commun. ACM*, 56(1):94–103, Jan. 2013.

[13] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), Aug. 2008. Updated by RFCs 5746, 5878, 6176.

[14] R. Droms, J. Bound, B. Volz, T. Lemon, C. Perkins, and M. Carney. Dynamic Host Configuration Protocol for IPv6 (DHCPv6). RFC 3315 (Proposed Standard), July 2003. Updated by RFCs 4361, 5494, 6221, 6422, 6644.

[15] A. Dunkels, J. Eriksson, N. Finne, and N. Tsiftes. Powertrace: Network-level power profiling for low-power wireless networks. 2011.

[16] A. Dunkels, B. Gronvall, and T. Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *Local Computer Networks, 2004. 29th Annual IEEE International Conference on*, pages 455–462, Nov 2004.

[17] M. Dunlop, S. Groat, W. Urbanski, R. Marchany, and J. Tront. Mt6d: A moving target ipv6 defense. In *MILITARY COMMUNICATIONS CONFERENCE, 2011 - MILCOM 2011*, pages 1321–1326, Nov 2011.

[18] D. Evans. The internet of things: How the next evolution of the internet is changing everything. `https://www.cisco.com/web/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf`, April 2011.

[19] R. P. Foundation. Model b. `http://www.raspberrypi.org/products/model-b/`.

[20] O. Gay. A fast software implementation in c of the fips 180-2 hash algorithms sha-224, sha-256, sha-384 and sha-512. `http://www.ouah.org/ogay/sha2/`, 2007.

[21] S. Groat, M. Dunlop, W. Urbanksi, R. Marchany, and J. Tront. Using an ipv6 moving target defense to protect the smart grid. In *Innovative Smart Grid Technologies (ISGT), 2012 IEEE PES*, pages 1–7, Jan 2012.

[22] T. Group. Thread. `http://www.threadgroup.org/`, 2015.

[23] O. Hardman. Optimizing a network layer moving target defense by translating software from python to c. Unpublished Master's Thesis, Virginia Polytechnic Institute and State University, 2015.

[24] E. Hemphill. Wigwag: Scan it. control it. rule it. share it. `https://www.kickstarter.com/projects/wigwag/wigwag-scan-it-control-it-rule-it-share-it`, 2013.

[25] Hewlett-Packard and L. Development Company. Internet of things research study. `http://www8.hp.com/h20195/V2/GetPDF.aspx/4AA5-4759ENW.pdf`, 2014.

[26] R. Hinden and S. Deering. IP Version 6 Addressing Architecture. RFC 4291 (Draft Standard), Feb. 2006. Updated by RFCs 5952, 6052.

[27] J. Hui and P. Thubert. Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks. RFC 6282 (Proposed Standard), Sept. 2011.

[28] S. Kent. IP Encapsulating Security Payload (ESP). RFC 4303 (Proposed Standard), Dec. 2005.

[29] S. Kent and K. Seo. Security Architecture for the Internet Protocol. RFC 4301 (Proposed Standard), Dec. 2005. Updated by RFC 6040.

[30] N. Kushalnagar, G. Montenegro, and C. Schumacher. IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals. RFC 4919 (Informational), Aug. 2007.

[31] A. Le, J. Loo, A. Lasebae, M. Aiash, and Y. Luo. 6lowpan: A study on qos security threats and countermeasures using intrusion detection system approach. *Int. J. Commun. Syst.*, 25(9):1189–1212, Sept. 2012.

[32] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler. Tinyos: An operating system for sensor networks. In W. Weber, J. Rabaey, and E. Aarts, editors, *Ambient Intelligence*, pages 115–148. Springer Berlin Heidelberg, 2005.

[33] J. Lusticky. Contiki ntp client. Bachelor's thesis, BRNO University of Technology, 2012.

[34] D. Mills. Network Time Protocol (NTP). RFC 958, Sept. 1985. Obsoleted by RFCs 1059, 1119, 1305.

[35] D. Mills. Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6 and OSI. RFC 4330 (Informational), Jan. 2006. Obsoleted by RFC 5905.

[36] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler. Transmission of IPv6 Packets over IEEE 802.15.4 Networks. RFC 4944 (Proposed Standard), Sept. 2007. Updated by RFCs 6282, 6775.

[37] R. Moore. addr_gen.c. October 2012.

[38] C. Morrell, J. Ransbottom, R. Marchany, and J. Tront. Scaling ipv6 address bindings in support of a moving target defense. In *Internet Technology and Secured Transactions (ICITST), 2014 9th International Conference for*, pages 440–445, Dec 2014.

[39] E. Nordmark and R. Gilligan. Basic Transition Mechanisms for IPv6 Hosts and Routers. RFC 4213 (Proposed Standard), Oct. 2005.

[40] V. Perelman. Security in ipv6-enabled wireless sensor networks: An implementation of tls/dtls for the contiki operating system. Master's thesis, Jacobs University, 2012.

[41] C. Perkins, E. Belding-Royer, and S. Das. Ad hoc On-Demand Distance Vector (AODV) Routing. RFC 3561 (Experimental), July 2003.

[42] J. Polastre, R. Szewczyk, and D. Culler. Telos: enabling ultra-low power wireless research. In *Information Processing in Sensor Networks, 2005. IPSN 2005. Fourth International Symposium on*, pages 364–369, April 2005.

[43] T. Preiss, M. Sherburne, R. Marchany, and J. Tront. Implementing dynamic address changes in contikios. In *Information Society (i-Society), 2014 International Conference on*, pages 222–227, Nov 2014.

[44] I. Proofpoint. Proofpoint uncovers internet of things (iot) cyberattack. `http://investors.proofpoint.com/releasedetail.cfm?ReleaseID=819799`, January 2014.

[45] S. Raza, S. Duquennoy, T. Chung, D. Yazar, T. Voigt, and U. Roedig. Securing communication in 6lowpan with compressed ipsec. In *Distributed Computing in Sensor Systems and Workshops (DCOSS), 2011 International Conference on*, pages 1–8, June 2011.

[46] Redwire. Econotag ii. `http://redwire.myshopify.com/products/econotag-ii`, 2014.

[47] E. Rescorla and N. Modadugu. Datagram Transport Layer Security Version 1.2. RFC 6347 (Proposed Standard), Jan. 2012.

[48] A. Research. The internet of things will drive wireless connected devices to 40.9 billion in 2020. `https://www.abiresearch.com/press/the-internet-of-things-will-drive-wireless-connect/`, 2014.

[49] B. Research. M2m/iot sector map. `http://www.beechamresearch.com/article.aspx?id=4`, 2015.

[50] J. Romkey. Nonstandard for transmission of IP datagrams over serial lines: SLIP. RFC 1055 (INTERNET STANDARD), June 1988.

[51] A. Sehgal. Contiki-ntp-syslog. `https://github.com/sehgalanuj/contiki-ntp-syslog`, December 2012.

[52] A. Sehgal. Contiki-tls-dtls. `https://github.com/sehgalanuj/contiki-tls-dtls`, December 2012.

[53] M. Sherburne, R. Marchany, and J. Tront. Implementing moving target ipv6 defense to secure 6lowpan in the internet of things and smart grid. In *Proceedings of the 9th Annual Cyber and Information Security Research Conference*, CISR '14, pages 37–40, New York, NY, USA, 2014. ACM.

[54] I. The AllSeen Alliance. The open source iot to advance the internet of everything. `https://allseenalliance.org/`, 2015.

[55] P. Thibodeau. Texas instruments builds an alternative energy for the internet of things. `http://www.computerworld.com/article/2861863/texas-instruments-builds-an-alternative-energy-for-the-internet-of-things.html`, 2014.

[56] S. Thomson, T. Narten, and T. Jinmei. IPv6 Stateless Address Autoconfiguration. RFC 4862 (Draft Standard), Sept. 2007.

[57] T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, J. Vasseur, and R. Alexander. RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks. RFC 6550 (Proposed Standard), Mar. 2012.