UNIVERSITY OF CALIFORNIA,
IRVINE

Security of Internet of Things Devices and Networks

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Computer Science

by

Zonglin Guo

Dissertation Committee:
Associate Professor Ian G. Harris, Chair
Professor Alex Nicolau
Professor Tony Givargis

2016

DEDICATION

To

My family, Donna and friends

Table of Contents

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to express the deepest appreciation to my advisor, Professor Ian G. Harris, for his advice and guidance through my Ph.D. program at University of California, Irvine. I would like to thank him for his continuous support, patience, motivation, enthusiasm, and immense knowledge in the past five years. He has been always very supportive not only just to my research, but also my professional development. I deeply appreciate his encouragement for me on discovering novel ideas, improving writing skills and discussing with other researchers. I could not have imagined having a better advisor and mentor for my Ph.D. study and research.

I would like to express my special appreciation to the professors in my dissertation committee, Professor Alex Nicolau and Professor Tony Givargis. They have provided insightful guidance and advice to my research work.

In addition, a thank you to Professor Phillip Sheu, who introduced me to Embedded System, and whose enthusiasm for the "semantic hardware" had lasting effect. I thank the help and support from all of my friends and colleagues at UCI and Broadcom, Lih-feng Tsaur, Li Liu, Ke hao, Qi Wang, Brandon Chen, David Liu, Xianbo Chen, Xiaoyu Ma, Jack Chen, Arthur Chen, Monica Cheung, Kurt Cheung, Jack Zhang, Steffen Peter. Besides, I would like to thank Donna Chen, Lianjie, Wesley Jiang, Fangjie, Ting, Yupeng Song and all other friends here who have offered me help, support and joy.

Last but not least, I would like to especially thank my family for giving me continuous love and care. I could not have done it without their supports and encouragement.

CURRICULUM VITAE

Zonglin Guo

EDUCATION

2007 B.S. in Electronic Information Engineering,
Huazhong University of Science and Technology, Wuhan

2009 M.S. in Computer Science, Wuhan University

2016 Ph.D. in Computer Science, University of California, Irvine

FIELD OF STUDY

Intrusion Detection, Internet of Things, Indoor Positioning

PUBLICATIONS

"Control-flow Checking for Intrusion Detection via a Real-Time Debug Interface." Zonglin Guo, Ram Bhakta and Ian G. Harris. Smart Computing Workshops, 2014 International Conference on IEEE.

"An on-demand scatternet formation and multi-hop routing protocol for BLE-based wireless sensor networks." Zonglin Guo, Ian G. Harris, et al. Wireless Communications and Networking Conference (WCNC), 2015 IEEE.

"A Residual Battery-Aware Routing Algorithm Based on DSR for BLE Sensor Networks." Zonglin Guo, Ian G. Harris, et al. Wireless Telecommunications Symposium (WTS), 2016 IEEE.

"An Efficient Approach of Preventing Battery Exhaustion Attack on BLE based Mesh Networks" Zonglin Guo, Ian G. Harris, et al. In submission.

ABSTRACT OF THE DISSERTATION

Security of Internet of Things Devices and Networks

By

Zonglin Guo

Doctor of Philosophy in Computer Science

University of California, Irvine, 2016

Associate Professor Ian G. Harris, Chair

The internet of things (IoT) has been attracting growing attention in recent years. As one potential technology used for IoT sensor network, Bluetooth Low Engery (BLE) is becoming one of the most anticipated solutions to establish IoT networks. Whilst BLE based IoT devices are becoming increasingly popular, security becomes an unavoidable concern not only for the devices by themselves but also for the whole network constructed with BLE devices.

In this thesis, we investigate the security issues on devices and we study the potential security issues for BLE-based IoT sensor networks. In order to explore the security on BLE scatternet, we explore how to form a BLE scatternet, how to implement an appropriate routing algorithm specifically for this network and what kind of potential attack this network may encounter often.

The research endeavor makes four contributions to the security field of intrusion detection on devices and IoT sensor networks. First, we propose a hardware-based

intrusion detection approach called CONtrol-flow VERification SystEm (CONVERSE), which ensures control-flow integrity by verifying the destination of control-flow branches at runtime. Many techniques exist for an attacker to alter control-flow to trigger malicious behavior, such as stack and heap overflows which overwrite a return address or function pointer. By verifying branch target addresses at runtime, security exploits can be detected as illegal control-flow. Secondly, we propose an approach for scatternet formation and multi-hop routing for BLE scatternet. We define procedures for device discovery, communication between piconets and forming multi-hop scatternet. Thirdly, we improve the routing algorithm for BLE network in step 2, and present BLE Scatternet Battery-Aware Routing (BSBR), a power aware routing mechanism based on Dynamic Source Routing (DSR) which can be applied to BLE-based Mobile Ad-Hoc Networks (MANETs). Furthermore, we study the impact of battery exhaustion attacks on BLE-based networks. In order to defend against this type of attack, we propose an intrusion detection and prevention approach requiring the suspicious nodes to switch roles with its connected nodes after a pre-determined time. If the suspicious node is identified as a malicious one, it will be blacklisted to prevent future attacks.

Chapter 1

Introduction

The internet of things (IoT) involves prevalence of objects and entities – known, in this context as things. Manifold definitions of IoT are proposed from either the "things" perspective or the "Internet" perspective, according to their interests and backgrounds. IoT is growing at a very fast pace. Much of the increase in IoT products comes from computing devices and embedded sensor system used in home and building automation, vehicle to vehicle communication, healthcare and wearable computing devices. Admittedly, the fast development makes big changes to people's lifestyle and bring convenience to our daily life. But as with every good thing, there is a downside to it. IoT devices and networks is becoming an increasingly attractive target for attackers. The IoT security has now become an issue of high concern. Security at both the device and network levels is critical to the operation of IoT. And therefore we decided to look into the security on IoT device itself and IoT networks. As one of the major communication technologies on offer to construct ad-hoc sensor networks, Bluetooth Core Specification 4.2 enables BLE devices for the IoT. It quickly becomes a highly adopted and anticipated communication solution for IoT. Since the BLE-based IoT security covers the area of endeavor concerned with safeguarding the devices and the BLE-based sensor networks, we break our research into several steps. Firstly, we investigate the security on individual devices. Then we study the BLE based IoT sensor network and investigate the network based attack.

1.1 Security on Devices

Malware, malicious software which executes on a machine without the user's consent, is the most prominent computer security threat today. The use of malware has enabled dramatic increases in the volume of cyber-theft which is common today. Malware is a threat not only to traditional computers but also to embedded systems which are increasingly computationally complex. The malware threat in the embedded systems domain has grown dramatically in recent years due to the ever-increasing level of network connectivity [4].

In order to defend computer and embedded systems from malware it is necessary to capture all aspects of malware behavior and analyze that behavior to gain a complete understanding of malware operation. Research in the fields of Intrusion Detection and Malware Analysis investigate the characterization of malware behaviors. Intrusion detection describes the automated detection of malicious behavior in real-time while an attack is occurring. Host-based Intrusion Detection Systems (HIDS) techniques evaluate the internal behavior of a host to identify malware attacks. Anomaly-based HIDS detects behavioral differences from a known-correct program execution. Accurate intrusion detection and malware analysis requires the acquisition of fine-granularity behavioral data in a stealthy manner which is invisible to malware. High performance is required for embedded systems where speed and cost are critical. Our approach develops a data extraction and analysis approach which supports satisfies all of these requirements.

We present a hardware-based approach which extracts behavioral data directly from the processor and analyzes that data at runtime [36]. A processor is added to the board-level design whose task is to check the behavior of the main processor. One of the main differences between our approach and other common hardware-based approaches is

the latter ones need to modify the exsiting chip design which lead to remendous cost. However, the hardware-based approach we propose does not have this cost by using the exsiting debug interface which is integrated into modern chips as one of important improved features. There are quite a few examples such as Freescale MPC553x, MPC555x and MPC556x series which contains a real time debug interface named as NEXUS [6]; AM335x series made by TI with implementation of Embedded Trace Macrocell (ETM). In our approach, the detection process or is a standard off-the-shelf processor which extracts behavioral data directly from the main processor to perform analysis concurrently with the function of the main processor. The fine-granularity of the data which is extracted at the hardware level increases the detection accuracy of the intrusion detection process.

1.2 Security on BLE based Networks

1.2.1 BLE-based IoT sensor network formation and routing

Bluetooth Low Energy (BLE) is a short-range wireless transmission technology aiming at low-cost, low-complexity communication. This technology was published originally under the name “Wibree”, and was later merged into the main Bluetooth standard in 2010 as an enhanced feature of Bluetooth when Bluetooth Core Specification version 4.0 [15] was adopted. BLE technology opens entirely new markets for devices featuring low cost and low power wireless connectivity. This technology was engineered to optimize the life of battery operated devices by allowing for short bursts of data transmissions instead of continuous streams which drain battery life and stimulate the invention of battery-operated devices and sensors. As an essential technology BLE

supports the predicted exponential market growth in wearable electronics as shown in Figure 1 from ABI Research. In this figure the Y axis shows market size in millions of units. According to the Harvard Business Review [33], the IoT is expected to connect over 28 billion "things" to the internet by 2020 including automobiles, appliances, industrial equipment, and wearable devices such as smart watches. BLE is an essential technology to support the predicted exponential market growth in wearable electronics. Increasingly, BLE products in application domains such as sports and fitness, medical devices, home automation, animal tagging and so on are emerging in people's daily lives. This class of devices can only communicate with each other at a short range due to power and battery constraints. How to use those devices to form a network and share the information between those devices is an important research topic.

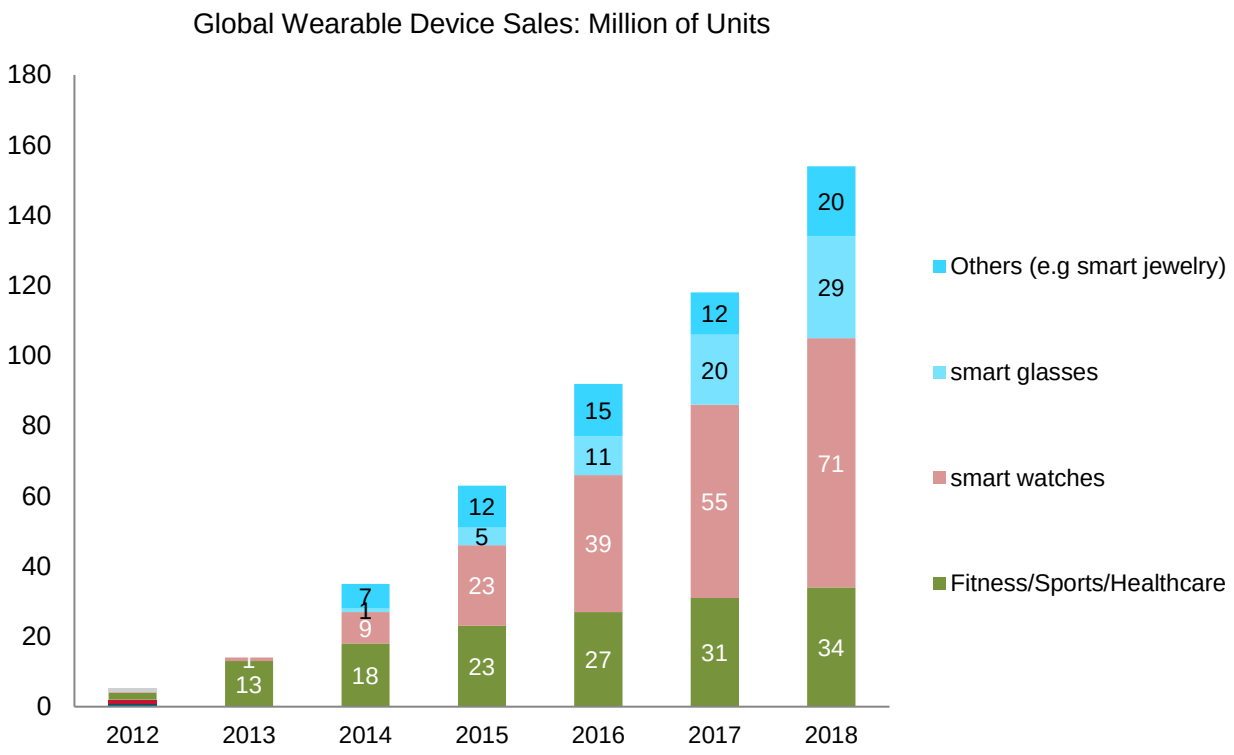


Figure 1.1 Global Wearable Devices Sales and Charts

BLE was originally introduced into the main Bluetooth standard in 2010 with the adoption of the Bluetooth Core Specification version 4.0. It allows Bluetooth devices to include new low-power profiles and extend the battery lifetimes. BLE as defined in the version 4.0 specification also places a number of limitations for interconnection between BLE devices. Specifically, each node can take only one role (master or slave) at a time, communication between two nodes can only occur within a single piconet, and a node cannot communicate with another one beyond its radio range. This makes it impossible to form a scatternet with BLE version 4.0 devices. The potential of scatternet formation and multi-hop routing has only become possible with the introduction of BLE version 4.1 [35] in December 2013. Key features included in Bluetooth Specification version 4.1 are the ability for a single node to be part of multiple piconets, and the ability for a node to act dual mode, as both a piconet master and slave. Thus BLE devices can take multiple roles at the same time which enables communication between piconets. Later, In Dec 2014 the newest version, v4.2 [31], was introduced. Version 4.2 provides flexible IPv6 connectivity and enables IoT devices with support for IPv6 over 6LoWPAN [32] or Bluetooth Smart Gateways. In this way, forming a scatternet and performing multi-hop routing becomes possible. Also as one potential technology being used for IoT, Bluetooth Low Engery (BLE) becomes one of the most popular solutions to establish BLE-based IoT networks.

Our work addresses this problem of forming a IoT sensor networks using a set of BLE devices, presenting an algorithm to form a scatternet and perform multi-hop routing to form a BLE-based MANET. Our work exploits the features of version 4.1 of the Bluetooth Specification which allows nodes to act as both master and slave, and to be members of multiple piconets. The routing protocol we propose is an on-demand approach [37]. It does

not require each node to have an overall knowledge of the whole network. Our approach allows for the dynamic inclusion of new nodes in the network at any time, supporting node mobility. We define the contents of BLE data packet including by-passing path information. Each master stores information about neighboring slaves, and each slave stores information about only the master whose piconet it is contained in. This is an improvement over other approaches which require a node to maintain information about all neighbors. Each time a new node joins an existing network as a master (slave), each neighboring slave (master) updates its knowledge of the network to include the new node. From that time onward, the multi-hop routing algorithm can take advantage of the new node in order to find the shortest communication path between two nodes.

Furthermore, in order to design an efficient battery aware routing algorithm for BLE MANETs, we must take into consideration specific features of the BLE protocol. In particular, four challenges present themselves when considering routing for BLE MANETS: 1) Communication range is reduced in BLE MANETs due to power constraints. 2) The amount of data transmitted between BLE devices is generally small. 3) BLE devices are mainly button cell battery-operated. When a routing algorithm attempts to select a path, it must consider the available battery capacity of each node in addition to the path length. 4) Large routing tables are inappropriate for BLE devices due to their limited storage space. To address these challenges, we extend an improved BLE based routing algorithm, Dynamic Source Routing (DSR), by applying a novel power aware mechanism. The proposed algorithm aims to provide efficient power utilization and achieve longer lifetimes for the scatternet [38].

1.2.2 Security Threats

Figure 1.2 shows the potential security challenges to IoT. This data from Janco Asscoiate shows the security threats is the most concerned issue to IoT. And therefore, we decide to study the DoS attacks to BLE based IoT, especially the battery exhaustion attack.

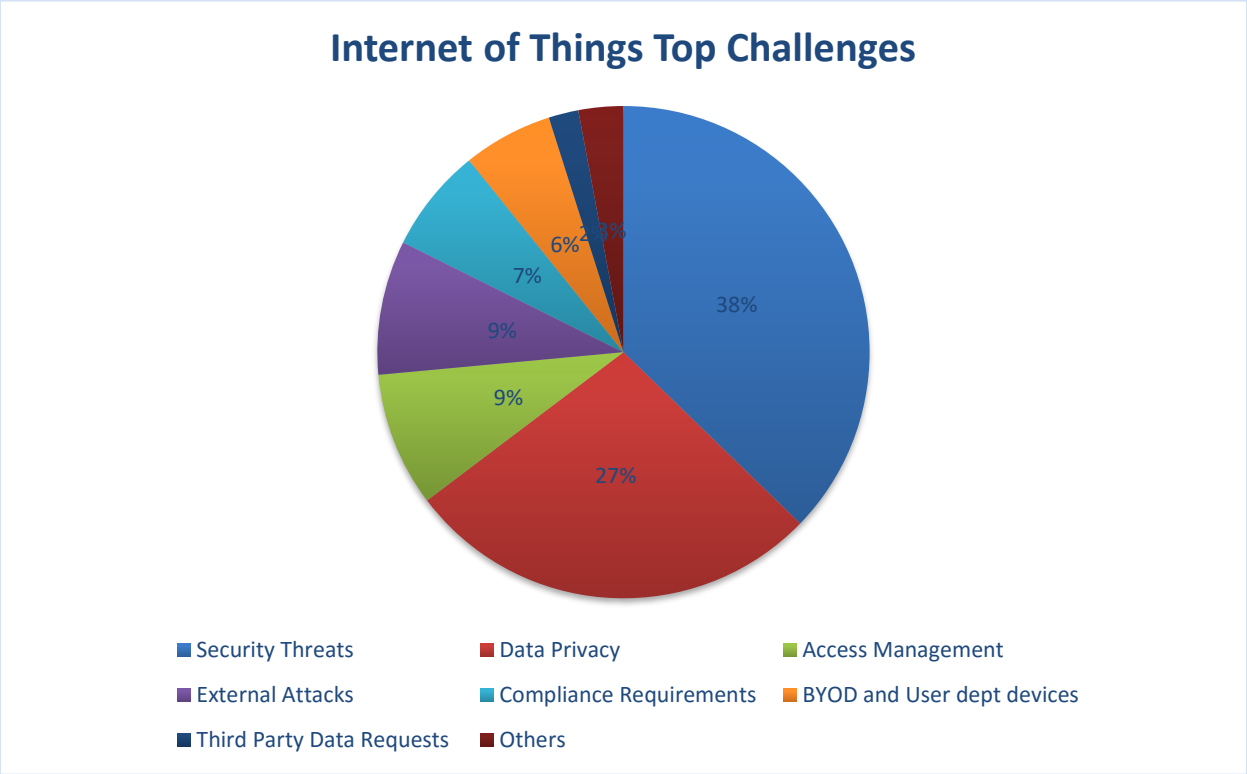


Figure 1.2 IoT Security

With the introduction of BLE based mesh networks, the security becomes an inevitable issue to pay attention to. There are different present threats such as denial of service (DoS), eavesdropping, sniffing and Malware injection. Among those threats, DoS is a common attach to target network. It is an attack that intentionally consumes device resources so as to ultimately prevent the victim to rightfully use them. There are several different ways to launch DoS attack to target network, and draining the battery is one typical of them, especially for the network powered by batteries. We focus on the analysis and prevention of DoS attack with battery exhaustion.

Bluetooth sensor networks are subject to a unique form of denial of service attack which is known as battery exhaustion attack [34]. This type of attack is launched with the intention of rapidly draining the battery of the target. This "battery exhaustion" attack prevents the devices from entering normal low power idle and disables the users from having continuous access to resources.

1.3 Dissertation Outline

This thesis is focused on firstly investigating the device based attack. And we propose the CONVERSE system using the real-time hardware debug interface of the processor to extract branch target addresses at runtime with no performance overhead and no area overhead on-chip. Secondly, we implement a BLE-based sensor network in order to investigate the network based attack. We propose an approach for scatternet formation and routing for the network. Then we propose another new routing algorithm BSBR to improve the efficiency. Finally, we study the battery draining attack, as one typical type of DoS attack to the BLE sensor network. In order to detect and prevent this attack, we propose an approach enforcing the authentication and verification on the devices which request service from the target networks.

The dissertation is organized as follows. In chapter 2, we present the related work of security on IoT devices and BLE based sensor network. Chapter 3 is dedicated to present our research of intrusion detection of devices based attack. Chapter 4 presents our research of formation, routing and security of BLE based network. Chapter 6 concludes the thesis and discuss some possible future research directions.

Chapter 2

Related Work

In this chapter, we will survey the related work of our research on security of IoT devices and BLE based networks.

2.1 Security on Devices

Hardware-based HIDS approaches vary in the granularity of the information which is extracted and analyzed. Researchers have exploited the exsiting hardware components such as Hardware Performance Counters (HPCs) which is a part of the processor's performance monitoring unit (PMU) [12], [11]. These approaches use HPCs to enforce the CFI by counting the events. However, this approach still causes performance overhead. In [12], the NumChecker needs the support of Virtual Machine running on operating system which is presumed to be trusted.

Some research groups have explored the possibility of adding logic to the design of the main processor in order to more extract fine-grained behavioral information [13], [3]. Research presented in [13], [9] target control-flow integrity by verifying the destinations of sequences of branch/jump instructions. Research presented in [3] examines destinations of individual branch/jumps and also examines the executed instructions.

Among these mechanisms, hardware-assisted CFI approaches are also introduced. In[8], the Last Branch Recording (LBR) history table of recent Intel processors is used by kBouncer. In this approach, LBR entries is validated according to a coarse-grained CFI

policy and the number of instructions executed between two indirect branches is counted by kBouncer. In [11], CFIMon uses performance counter and the Branch Trace Store(BTS) mechanism to check CFI which requires an offline training phase. A similar approach is also proposed in[14]. Recent research presented in [7] demonstrates the weakness of proposed software-based approaches which ignore some fraction of control branches in order to reduce performance impact.

Different from existing coarse-grained CFI solutions which suffer from performance overhead or a trade-off between performance and security, this proposed hardware-assisted framework uses a per-function label and a state model to enforce a fine-grained CFI.

2.2 Security on BLE Sensor Networks

2.2.1 Formation and Routing of BLE Sensor Networks

Protocols for scatternet formation and routing have been proposed by several authors, but so far there is no research discussing multi-hop routing protocol on a BLE network since Bluetooth Specification version 4.1 is new.

1) Scatternet Formation

Among those scatternet formation algorithms for mobile ad-hoc devices, there are several scatternet formation algorithms for Bluetooth (not BLE) devices such as BluesStars[20], Bluemesh[21], Bluetrees[22] and SHAPER[23]. Previous work on Bluetooth scatternet formation is not applicable to BLE version 4.1 due to the enhancement and changes on the features. The earlier Bluetooth specifications limit the number of slaves for each master to be up to 7. In the new specification there is no limit on maximum number of

slaves to a piconet master. Additionally, the procedures for forming a Bluetooth piconet are totally different from forming a BLE piconet. Bluetooth defines four device modes (active, parking, sniff, and hold) which need to be dealt with in the formation procedure but are not applicable for BLE scatternet. In the formation of BLE scatternet, the nodes take the roles of initiator, scanner or advertiser based on the need.

2) Scatternet Routing

Numerous MANET routing algorithms have been presented in previous work [19]. MANET routing algorithms can be classified into two categories: Table-driven (proactive) and On-demand (reactive). In a proactive algorithm, each node needs to have complete knowledge of the network which in turn requires the node to have enough memory space to store the routing information. Conversely, in a reactive algorithm each node does not need to maintain a routing table. Instead, it discovers routes for each transaction based on need. Each node performs two functions: route discovery and route maintenance. These two types of algorithms have advantages and disadvantages based on the application. If the topology of the network does not change often, the table-driven routing protocol can provide available routes immediately. Otherwise, on-demand algorithms achieve better performance since they do not have the overhead associated with frequently updating a routing table. The performance varies based on how often the topology of the network changes.

Table-driven routing protocols require each node in the network to maintain one or more tables storing routing-related information, and these tables are updated periodically by propagation when the network topology changes. The main advantage of these protocols is that they provide the available routes from source to destination immediately.

The weakness is that it incurs slow reaction on restructuring, substantial signaling traffic and power consumption, especially when the topology of wireless network changes frequently. There are already quite a few researches done in this field: Destination sequenced Distance-Vector Routing Protocol in [27] based on Bellman-Ford routing algorithm preventing loop-formation, Wireless Routing Protocol in [28] and Zone-based Hierarchical Link State Routing Protocol in [29]. These algorithms differ in the number of necessary routing-related tables and the method by which changes in the topology are distributed through the network.

On-demand routing protocols establish routes only when required by the source nodes. The main advantage of these protocols is they save power and bandwidth by eliminating the periodic table-update packet required in the table-driven protocols. These protocols do not need to update and maintain the tables. The drawback of these protocols is they incur high time latency in establishing the route from source to destination. Some of the researches are as follows: Ad hoc On-demand Distance Vector Routing in [30], Dynamic Source Routing Protocol (DSR) in [17] and Better Approach to MANET (BATMAN) in [18]. BATMAN is a proactive routing protocol for multi-hop ad-hoc mesh networks. It does not calculate all routes. Instead, it maintains routing information to the nearest neighbor. As a result, the complete network topology is not visible to any single node. Routing decisions are distributed between all nodes. Dynamic Source Routing (DSR) is a reactive routing protocol. It is based on source routing whereby all routing information is maintained at mobile node instead of relying on a routing table at each intermediate device. It discovers the route on-demand by flooding a packet containing the source node address, destination

node address, request id, and path throughout the network and subsequently receiving reply packets with path information.

Algorithms like the two summarized above are typical yet are not well suited for BLE-based MANETs since they do not consider specific features of BLE devices and BLE based mesh network such as piconet synchronization, communication scheduling, and role switching.

3) Unique Aspects of BLE

Existing routing algorithms including those we have discussed above, cannot be applied to BLE-based MANETs directly since those algorithms do not take the specific features of BLE devices and BLE-based mesh networks into consideration.

1) Piconet Synchronization. BLE-based mesh network is composed by multiple different piconets, and each piconet has only one master. Its structure is significantly different from the other MANETs which do not address piconet synchronization issues within the scatternet.

2) Communication Scheduling. In intercommunication between BLE devices, the master takes the control of the clock and the communication is event driven. Devices communications within the same piconet use different channels with different event for communication based on need. At the route discovery phase, the appropriate value of advertisement interval must be assigned based on the required frequency of sending data packets.

3) Experiments on Real Hardware. Almost all results in related previous work are generated using simulation rather than real hardware. By generating results for our

approach using real hardware we are forced to deal with all of the complexities which arise in the physical world and may be obscured through simulation.

We propose an approach for implementing the routing protocol of a real system. Since a key feature of BLE networks is to periodically exchange small amounts of data with low power consumption, the battery power consumption should be a first-order consideration in the routing algorithm in order to ensure a long lifetime for the network. Hence, we propose an algorithm which addresses some of the concerns specific to BLE networks. Our algorithm recognizes that, due to energy constraints, network lifetime is a key parameter of performance. The energy failure of a node affects its ability to forward data packets and hence affects the overall network lifetime.

2.2.2 Battery Exhaustion Attack

The notion of battery draining attack which is also termed as sleep deprivation torture attack is firstly tackled by Stajeno and Anderson [34]. This emerging class of attacks represent malicious situations whereby the battery lifespan of the victim has been discharged, and therefore the user is deprived access to information [45]. Martin further presents three variations of these battery exhaustion attacks [39]:

1) Service Request Power Attack: The attackers send certain services requests repeatedly to target devices in order to deplete their batteries.

2) Benign Power Attack: This attack forces target devices to repeatedly execute energy-guzzling tasks to accelerate the process of battery depletion such as requesting the victim device to run a hidden java script.

3) Malignant Power Attack: This attack creates or modifies an executable to not only drain the battery, but also perform harmful behaviors like trojan horses to the target systems.

In recent years, a lot of researches have been carried out to prevent the battery exhaustion DoS attacks on mobile devices including Bluetooth devices.

Nash introduced a multiple linear regression model to analyze the power consumption of each process [41]. In his power estimation model, it takes system performance parameters into consideration such as CPU load, disk read/write, network bytes operation or so. The correlation coefficient is produced with those parameters and is used to model power consumption of the target system. In this way battery exhaustion attack can be detected when power consumption of a process exceeds its estimate amount of usage.

Jacoby presented the Battery-based Intrusion Detection System (B-BID) aiming at solve the battery exhaustion attacks [42]. As the first power monitoring anomaly-based intrusion detection system, it consists of three modules: Host Intrusion Detection Engine, Scan Port Intrusion Engine and Host Analysis Signature Trace Engine.

Racic exploited the vulnerabilities of the multimedia messaging service and demonstrated battery exhaustion attacks by keeping the victim cellular phone in a busy state even without the awareness of the phone user and network administrator [43]. Without the protection against a battery-sensing IDS, this type of attack will drain more device power.

Buennemeyer demonstrates a Battery-Sensing Intrusion Protection System using a Dynamic Threshold Calculation algorithm which iteratively takes known device processes,

system states and backlighting into consideration [44]. As a hybrid of Anomaly detection system and tradition IDS, it provides security administrators to detect anomalous Wi-Fi and Bluetooth attack activity. When power consumption changes in mobile hosts, the security administer will be alerted and look into this change.

These researches mainly focus on analyzing the power consumption using either additional hardware system or algorithms which produce performance overhead and delay. Difference with these existing approaches, our method adopts the newest feature of Bluetooth specification enabling the BLE devices to switch roles during the communication.

Chapter 3

Security on Devices

In this chapter, we study the security issue for IoT devices. And we focus on control flow checking for intrusion detection.

3.1 Intrusion Detection

Different methods and software/hardware approaches.

Then we use Hardware/ Why

Why we choose the control flow integrity- Real -time and no performance overhead

3.2 Threat Model

There are several different attacks aiming to change the branch destination address. Stack-based buffer overflow attack is a typical example. Many exploits depend on the existence of stack-based buffer overflows which allow control-flow to be redirected by overwriting a function return address and other stack data. As a common way to exploit a buffer overflow vulnerability, return-oriented programming attacks [10] redirect control-flow to re-sequence code present on the target machine to perform malicious behaviors. As a common form of return-oriented attack, Return-to-libc attack does not need an executable stack. Instead it redirects the vulnerable program to jump to some existing functions in the libc library such as system(), and places arguments on the stack in order to control the execution of the library function which in turn ruins the control-flow integrity.

The common thread relating these malware attacks is that the normal control-flow must be altered in some way. Our approach defends against these attacks by detecting abnormal changes in the control-flow which do not match the correct behavior of the executing code.

Our control-flow integrity approach, as is true for all previous CFI approaches, cannot fully capture the impact of hardware interrupts on the control-flow. Unlike software interrupts, hardware interrupts are often triggered by unpredictable events, and can alter the control-flow at any point during functional execution. A change in control-flow to the start of an interrupt service routine, or a change from the end of an interrupt service routine, is always possible, and therefore difficult to distinguish from malware behavior. In spite of this weakness, control-flow integrity has been found to provide excellent protection against malware because the attacker is strictly limited to interrupt behaviors which are difficult to exploit in practice.

3.3 CONVERSE System Architecture

The CONVERSE system and its interaction with the system under test (SUT) at runtime are shown in Figure 3.1. We assume that the SUT has a real-time debug interface which is capable of transmitting the target addresses of each branch at runtime. The CONVERSE system receives the branch target addresses and verifies their correctness in real-time, while the SUT continues to execute. CONVERSE is implemented on a microcontroller unit (MCU) and its runtime operation involves two main components, the Branch Checking Code which verifies the correctness of each branch, and the Branch Destination Table which contains the legal destinations for each branch.

The system under test which we use to evaluate the CONVERSE system is implemented on the Freescale MPC5604P target board which contains the Freescale e200z0 processor. This processor supports the IEEE-ISTO 5001-2003 standard Nexus debugging interface [6] which the CONVERSE system also supports. The CONVERSE system is built using the 8bit MCU with 64kb on-chip memory.

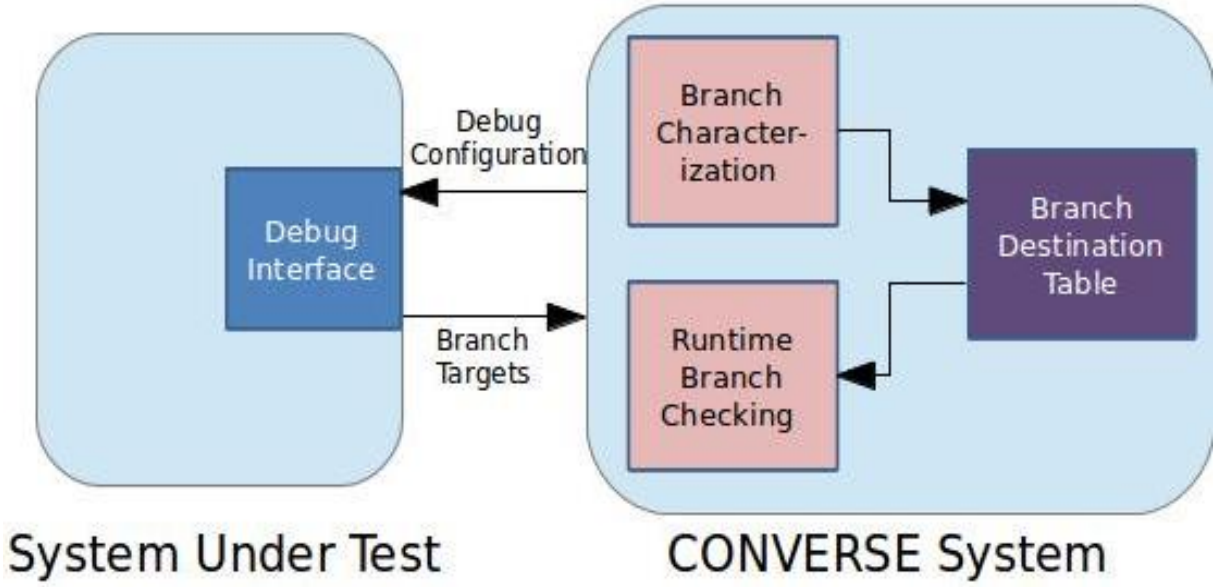


Figure 3.1 Interaction between CONVERSE and SUT

A. Runtime Checking

CONVERSE is in this mode after the system is deployed while the system is in functional operation. When the system under test is booted, CONVERSE communicates with its debug interface in order to configure it to transmit branch source/destination pairs. During functional operation each branch source/destination received by CONVERSE is the branch target is compared to the legal branch targets for the branch instruction, as contained in the Branch Destination Table. If a mismatch is found, then an attack is detected.

B. Branch Characterization

Branch characterization is performed prior to system deployment in order to populate the Branch Destination Table. Each entry in the Branch Destination Table contains a Source Address, the address of the branch instruction, and a Destination Address, the address of the branch destination. Entries are not repeated in the table, so if the branch source/destination pair is already contained in the Branch Destination Table, it is ignored.

A branch instruction may be either a direct branch, whose address is hard-coded, or an indirect branch. Each direct branch will have a single entry in the tables since each direct branch can have only one destination address. An indirect branch destination is determined by the contents of a register which can change across different executions of the code. For this reason, a single indirect branch may have several entries in the table which all represent legal branch targets.

Previous work in software-based control-flow integrity have applied one of two methods to perform branch characterization. Some approaches perform static analysis of the executable to determine all branch destinations. Other approaches perform dynamic analysis by executing the code with a test sequence and recording the branch destinations at runtime [1]. This approach can capture indirect branches but control flow modeling may be incomplete if the test sequence is insufficient. In CONVERSE we use both static and dynamic analysis approaches.

1) Static Branch Characterization: Static analysis of the executable of the system under test is performed to determine the destinations of all direct branches. The pseudocode for the static analysis algorithm is shown in Figure 3.2. The branch destination table BDT is initialized to an empty table on line 1. The while loop starting at line 3 of the algorithm scans each instruction starting with the first instruction in the text section of the

executable file. The address of the instruction currently being examined is sourceAddr and it is initialized to the beginning of the .text section on line 2 of the algorithm. The instruction instr is fetched from the current address and is checked to determine if it is a direct branch on line 5 of the algorithm. If the instruction is a direct branch, then the destination address is extracted from the instruction on line 6 and the source/destination pair is inserted into the branch destination table on line 7.

```
1. BDT = EmptyBDT;
2. sourceAddr = 0;
3. while (sourceAddr < END_TEXT_SEGMENT){
4.     instr = FetchInstruction (sourceAddr);
5.     if (IsDirectBranch (instr)){
6.         destAddr - GetDestination ( instr) ;
7.         InsertIntoBDT (sourceAddr, destAddr);
8.     }
9.     Increcment (sourceAddr);
10.}
```

Figure 3.2 Static Analysis Algorithm

2) Dynamic Branch Characterization: Dynamic analysis used to determine the destination addresses of all indirect branches whose destinations can only be determined at runtime. The system under test is used in functional mode prior to deployment and is stimulated with test input data. It is assumed that characterization is performed in a secure environment so that correct system execution is guaranteed. After the initialization of

Nexus Development Interface (NDI), branch source/destination pairs are generated and received by the CONVERSE system and are used to populate the Branch Destination Table.

3.4 NEXUS 5001 Debugging Interface Standard

The Nexus 5001 Forum standard, developed by the IEEE Industry Standards and Technology Organization, is an attempt to expand and standardize the diverse approaches to on-chip debugging [6]. The feature of the Nexus debug interface which is used by our tool is its support for Program Trace, providing information about every non-sequential control-flow change. We configure the Nexus debug interface to transmit a **Branch Trace Message (BTM)** when a branch instruction is executed. The fields of interest are the Branch History field which contains the source address of the branch, and the Full Target Address which contains the destination address of a branch. CONVERSE receives these messages and parses them to trace the control-flow of the system under test.

The Nexus interface contains a message buffer used to hold trace messages before they are read by an external debugger such as CONVERSE. If messages are not read quickly enough it is possible for the buffer to become full, causing branch trace messages to be dropped. When one or more branch trace messages is dropped, a *BTM overflow* message is generated to notify the CONVERSE system. Dropping BTM messages degrades security because the branches which are dropped cannot be verified for correctness. For this reason, we seek to characterize the branch drop rate so that the CONVERSE system can be designed to reduce or eliminate the problem.

Branches will be dropped if the rate at which BTMs are entering the message buffer is greater than the rate at which BTMs are being extracted from the buffer. The rate at

which BTMs are extracted is limited by the speed of the message buffer, the width of the message buffer interface, and the maximum rate at which CONVERSE can process each BTM.

To formulate the branch, drop rate, we define the following variables,

- R_b is the rate at which branches are executed by the system under test. This is determined by the functional code and the performance of the processor on the system under test.
- B is the bitwidth of the debug interface. The Nexus interface uses a minimum bitwidth of 4 bits which are used to extract data from the message queue. We refer to the B as the size of the debug word.
- R_e is the maximum rate at which debug words can be extracted from the message queue. This is determined by the speed of the processor in the system under test.
- R_p is the average rate at which CONVERSE can process the data extracted from the message queue. CONVERSE reads bits until an entire BTM has been extracted before processing the data, so R_p is computed as an average. The value of R_p depends on the processor used in CONVERSE and on the algorithm which converse uses to verify the branch destinations.

We formulate the maximum rate at which debug data can be extracted and processed, R_f, as $R_f = min(R_e * B, R_p)$. The min function is used because the rate is bounded by both the rate of data extraction and the rate of data processing. The branch drop rate is expressed in Equation 3.1.

$$R_{drop} = \begin{cases} 0, & if\ Rf \geq Rb \\ Rb - Rf, & otherwise \end{cases} \quad (3.1)$$

3.5 Evaluation

We evaluate the effectiveness of our proposed system by performing two sets of experiments. We first evaluate the ability to detect exploited vulnerabilities in software by developing a radio frequency (RF) transmission/reception test bench system and inserting a vulnerability into that system. Then the RF system is operated in conjunction with CONVERSE, and we perform a cyber-attack against the RF system to see that CONVERSE detects the attack.

```
1. void halSpiReadBurstReg(INT8U addr, INT8U *rxbuffer,
                           INT8U packetLength) {
2.      INT8U i, temp;
3.      temp = addr — READ_BURST;
4.      CSN = 0;
5.      while (MISO)
6.         SpiTxRxByte(temp);
7.      for (i=0; i¡packetLength; i++)
8.         rxBuffer[i] = SpiTxRxByte(0);
9.      CSN = 1;
10. }
```

Figure 3.3 Vulnerable Receiver Function *halSpiReadBurstReg( )*

The second set of experiments evaluate the detection latency of CONVERSE and the branch drop rate which can occur when the branch rate is high. A set of sorting algorithms are executed with different data on the system under test and several branch destinations

are changed manually. The use of a set of relatively small programs allows us to have more direct control over the branch rate and explore the variation in the branch drop rate.

A. RF Test bench

We have designed and implemented an RF test bench. The Texas Instruments CC1101 RF module is used as the wireless transmit/receive terminal. To implement the transmitter, we use an 8-bit MCU, and operate the wireless module at 433MHz. The transmitter contains two buttons A and B which both cause a data packet to be transmitted to a buffer in the receiver. Pressing button A transmits a data packet which is smaller than the receiver's buffer, so the data is properly processed. Pressing button B transmits a packet which overflows the receiver's buffer, overwriting the return address on the receiver's stack, and altering the control-flow of the receiver.

1) Buffer Overflow Vulnerability: The stack-based buffer overflow vulnerability in the receiver can be seen in the code shown in Figures 3.3. The function *halSpiReadBurstReg(),* defined in Figure 3, is called to receive an incoming RF packet. The third argument to the *halSpiReadBurstReg()* function is *packetLength* which is computed on as the size of the incoming data packet.

The code in Figure 3.3 defines the *halSpiReadBurstReg()* function which is called when a new RF data packet is received. The purpose of this function is to move the incoming message into the buffer named rxBuffer. The data is read using the common SPI protocol, and the CSN and MISO pins are defined as part of that protocol. The number of bytes copied into the rxBuffer is determined by the *packetLength* argument passed to the function. In our RF test benchmark code, the *packetLength* argument is mistakenly

computed based on the size of the incoming message, not the size of rxBuffer. If the size of the incoming message is larger than the rxBuffer, a buffer overflow will occur.

B. RF Experimental Results

In order to evaluate the ability of CONVERSE to detect attacks, we use the transmitter to launch the cyber-attack against the RF test bench. Upon power-up of the receiver and transmitter, both components configure themselves and the receiver waits to receive RF data packets. The receiving buffer is 33 bytes long. Testing is performed by pressing the A and B buttons on the transmitter in order to send normal and overflow packets, respectively. Pressing the A button transmits 32 bytes, while pressing the B button transmits 64 bytes, overflowing the buffer and changing the return address to alter control-flow.

The detection and performance results are shown in Table I. We transmitted a series of 32 byte packets which do not overflow a stack buffer, and 64 byte packets which do overflow the stack buffer, changing the return address on the stack. Each row in the table shows the results of sending a packet of the size contained in the first column. Latency varies slightly between executions due to buffering delays in the debug interface, so we repeated each experiment, transmitting 5 short packets and 5 overflow packets, and Table 3.1 reports the average delay.

After transmitting each packet, the RF system returns from the *halSpiReadBurstReg()* function by branching to the return address located on the stack. The *Overflow* column indicates whether or not a stack buffer overflow does occur, and the table shows that this was the case for the 64 byte packets, but not the 32 byte packet. The *Detection* column indicates whether or not the CONVERSE system detected a buffer

overflow. Table 3.1 shows that CONVERSE correctly detected an overflow only when an overflow occurred, for both 32 byte and 64 byte packets. The *Det. Latency (µs)* column shows the time between when the branch occurred and when the CONVERSE system correctly validates the branch. Note that this latency value does not impact the functional performance of the system because the validation process is concurrent with the functional execution.

Table 3.1 RF Detection and Performance Results

Packet size	Overflow (Y/N)	Detection (Y/N)	Det. Latency (µs)
32 bytes	N	N	303
64 bytes	Y	Y	592

C. Branch Rate Experiments

Our second set of experiments are used to evaluate the impact of variation on the branch rate on the branch drop rate. We use CONVERSE to detect control-flow changes in our implementations of three sorting algorithms, insertsort, shellsort, and quicksort. The algorithms are executed on the Freescale MPC5604 microcontroller which contains a Nexus 5001 debug interface. Each algorithm is executed with different datasets of two different sizes, 256 bytes and 4096 bytes, in order to explore some range in the rate of branch execution. Five different data sets of each size are used and we report the average detection latency across all 5 executions.

Performance results of the CONVERSE system with the sorting algorithms are shown in Table 3.2. Column 1 shows the name of the sorting algorithm used. Column 2 shows the size of the data set applied. The *BranchRate(b/sec)* column is the rate of branch

execution in branches/second. The *Det. Latency (µs)* column shows the time between when the branch occurred and when the CONVERSE system correctly validates the branch, on average. It is important to note that the validation time does not impact the performance of the system under test because validation is performed by CONVERSE in parallel with functional execution.

Table 3.2 Sort Detection and Performance Results

Sort	Data Size	Branch Rate (b/sec)	Det. Latency (µs)
insertsort	256	87117	0.98
	4096	93354	0.95
shellsort	256	65253	2.02
	4096	89011	1.02
quicksort	256	73888	1.05
	4096	97476	1.45

Figure 3.4 shows the relationship between the branch rate and the branch drop rate by plotting each of the six results presented in Table 3.2. In Figure 3.4, the X-axis is the Branch Rate in 10^3 branches per second. and the Y-axis is the Branch Drop Rate in 10^3 branch trace messages per second. The goal of the experiment is to see if the actual drop rate matches the drop rate predicted by Equation 3.1, which is drawn as a line in the Figure. Notice that the actual drop rates are relatively close to the rates predicted by Equation 3.1.

In order to lower and even eliminate the miss rate, we would need to ensure the fetch rate is at least equivalent to the branch rate. Using a processor with higher performance in the CONVERSE system would achieve that goal.

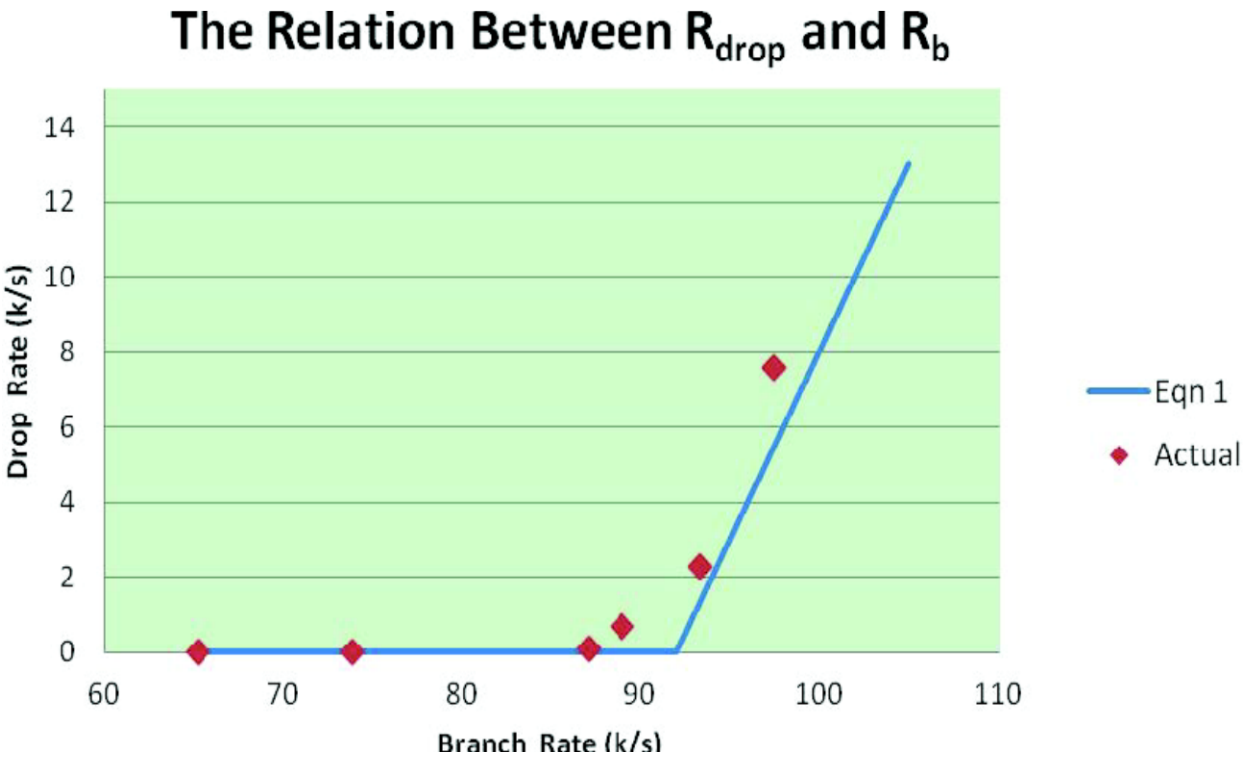


Figure 3.4 Branch Drop Rate Results

D. Comparison with Other Approaches

The comparison with other relevant approaches are shown in Table 3.3. Those two approaches are also compatible with COTS design techniques without any changes to the hardware either, but they produce performance overhead. In NumChecker[11], The lowest performance overhead is 2.8% when NumChecker is specified to be invoked every 5 seconds. But it goes up to at least 4% when there is just file copy with just 1MB. In CFIMon [12], the overhead is about 9% just for benchmarks of apache-put. Compared with this, our approach is more efficient and generic by using a real-time existing debug interface.

Table 3.3 Comparison with Other Approaches

Approaches	OS Support (Y/N)	Real-time (Y/N)	Overhead (%)
NumChecker	Y	N	2.8
CFIMon	Y	N	6.1
Converse	N	Y	0

3.6 Conclusions

In this chapter, we propose a system to detect the intrusion attack to IoT devices. This CONVERSE system provides an effective approach to defend against control-flow-based attacks, and we have validated its detection ability with an RF test bench and several smaller examples. Our approach is hardware-based which does not incur functional performance overhead. Due to the use of an existing real-time debug interface, the detection is in real time with a short validation latency. As experimental results indicate, the detection of attacks which modify control-flow is achieved by our approach. By using an existing debug interface, our approach does not require changing the design of the system under test, and is therefore compatible with COTS design techniques.

Chapter 4

Security on BLE Based Sensor Network

In this chapter, we study the formation of BLE scatternet. In order to enable the data sharing in this BLE based IoT network, we also study the routing protocol particularly for BLE based sensor network. Then we look into the security issue of BLE sensor network, investigate the potential intrusion detection, especially the battery draining attacks.

4.1 BLE Protocol and IoT

Bluetooth (BT) technology has enjoyed popularity and prosperity in the last 10 years for audio and stereo communications. With the maturity of the technology and its strong presence in the marketplace, Bluetooth industry has been working on expanding the applications of the technology from audio communications and stereo streaming to short-range wireless communication markets, such as the IoT and M2M communications. And more and more corporations are betting Bluetooth for IoT applications.

For Bluetooth to be suitable for IoT and M2M applications, it needs to reduce power consumption in order to be adopted in battery powered devices for a longer period of time. To achieve that, BLE is firstly introduced in Bluetooth Specification 4.0 by SIG. Later, it is improved in version 4.1 and further improved in version 4.2. Additionally, expended work on BLE is also done by Bluetooth stakeholders. The Internet Engineering Task Force (IETF)

is making a great effort on standardization of facilitating Bluetooth in exchanging IPv6 over 6LoWPAN.

1) Bluetooth and BLE

Even if both technologies have a common name in “Bluetooth,” it is important to understand that the addition of BLE technology is not the same thing as when new versions of the Bluetooth Specification have been released in the past. BLE, as a whole new game, does not replace Classic Bluetooth technology.

When considering the difference between BT and BLE, it is important to refer to power consumption. Bluetooth was originally designed for continuous, streaming data applications over a short range, while BLE was developed as a way to not exchange large amounts of data as Bluetooth for applications which need to run on battery power for a long time at a cheaper cost. Although that may sound like something negative, it’s indeed extremely positive when talking about M2M communication. With BLE’s extremely lower power consumption, applications can run on a small battery for years. It is actually vital for applications that only need to exchange small amounts of data periodically. BLE create low-duty cycle transmissions or a very short transmission burst between long periods. BLE achieves low power consumption by remaining in sleep mode most of the time and only wakes up when a connection is initiated. BLE power consumption is kept low because the actual connection times are of only a few ms, unlike Bluetooth which takes about 100ms.

Many features of Bluetooth are inherited in BLE, including adaptive frequency hopping (AFH) as well as part of the logical link control and adaptation protocol (L2CAP) interface. In addition, BLE also implements the same link security with simple pairing modes, secure authentication, and encryption.

The table below shows a high level comparison between classic Bluetooth and BLE technology.

Table 4.1 Comparisons between Bluetooth and BLE

Technical Specification	Classic Bluetooth	BLE
Radio Frequency	2.4 GHz	2.4GHz
Modulation	GFSK (modulation index 0.35), π/4 DQPSK, 8DPSK	GFSK (modulation index 0.5)
Distance	100m	100m
Symbol Rate	1~3 Mbps	1Mbps
Channels	79	40 (3 advertising and 37 data)
Bandwidth	1MHz	2MHz
Application Throughput	0.7~2.1 Mbps	305Kbps
Security	64bit/ 128 bit AES	128 bit AES
Error Detection	8 bit CRC (header), 16bit CRC, 2/3 FEC (payload), ACKs	24bit CRC, ACKs
Latency	~100ms	<6ms
Power Consumption	1(reference value)	0.01~0.05 (case dependent)
Primary Use Cases	Wireless headsets File transfers Wireless keyboards/ printers wireless speakers stereo audio and streaming video	Health monitors Sports and fitness devices Industrial monitoring sensors Geography-based, targeted promotions like iBeacon Automotive; Automation

BLE operates in the same band as classic Bluetooth but adopts different FHSS scheme. As BLE devices are incompatible with Bluetooth devices, they will not interoperate with them. Dual mode devices having protocol stack of Bluetooth and BLE need to be developed in order to have interoperability between Bluetooth and BLE devices. A dual-mode device has an integrated circuit that includes both a standard Bluetooth radio and a BLE radio. Dual mode chips have been developed by vendors such as Broadcom, CSR, EM Microelectronics, Nordic semiconductor, Texas Instruments etc. In addition, not immediately apparent from the Bluetooth specifications is the fact that the relaxed requirements for BLE allow vendors to do optimizations which are difficult even impossible to make with classic Bluetooth, lowering sleep and active currents and shortening switching times. These optimizations enable single-mode chips to produce lower power, to be simpler and have lower cost than dual-mode chips.

2) BLE Versions

BLE or Bluetooth Smart is included as a sub protocol in Bluetooth with versions 4.0, 4.1 and 4.2. Since BLE was originally introduced in Bluetooth specification 4.0 in 2010, it has been upgraded twice so far in Bluetooth 4.1 and Bluetooth 4.2. In version 4.0, BLE is specified as a lower power protocol allowing smart devices to remain connected for longer period of time without draining the battery. In this version, BLE is developed for applications which need to only transmit small amounts of data at intervals. The energy saving effect is achieved by the shifting of functions from the host to the controller. The controller handles the background communications from itself and leaves the host longer in sleep mode. In addition, a faster connection reduces the power requirements. Then version 4.1 took BLE to another level. This version eliminates the coexistence issue

between BLE and 4G LTE by coordinating their radios automatically. Without overlap, both can perform at their maximum potential. In addition, this version also improved data transfer. Most of importance is this version lays the ground work for BLE to be a popular solution for IoT network. This version allows BLE slave nodes to participate in multiple piconets and provides smart connectivity. In version 4.2, the privacy and data packet length is improved. BLE is improved to consume lower power and is enabled to access internet

3) BLE for IoT

Besides BLE, there are other protocols being used for IoT, such as ANT+, Zigbee, NFC and WiFi. Here, we compare the key features of these protocols.

Table 4.3 Comparisons of Common IoT Protocols

Technology	BLE	ANT+	Zigbee	NFC	WiFi	RF4CE
Engery per bit	153nJ	710nJ	186knJ	5.25nJ	Reader side	~186knJ
Coin battery life (120bit/s)	191 days	52.64days	Too high	Too high	Too high	Too high
Distance	100m	30m	100m	5cm	150m	100m
Throughput	305Kbps	20Kbps	100Kbps	424Kbps	6Mbps	100Kbps
Cost $ (<10k unit)	1.95	>3.33+MCU	3.20	1+MCU	3+MCU	2.75
PCB Size (mm^2)	20	125	306	100	60	305

4.1.1 Background

In this section, the basic concepts and operations of BLE devices are described in detail.

1) BLE Stack

The BLE stack implements all the mandatory and optional features of Low Energy Single Mode compliant to Bluetooth Core Specification. The BLE Stack implements a layered architecture of the BLE protocol stack as shown in Figure 4.1.

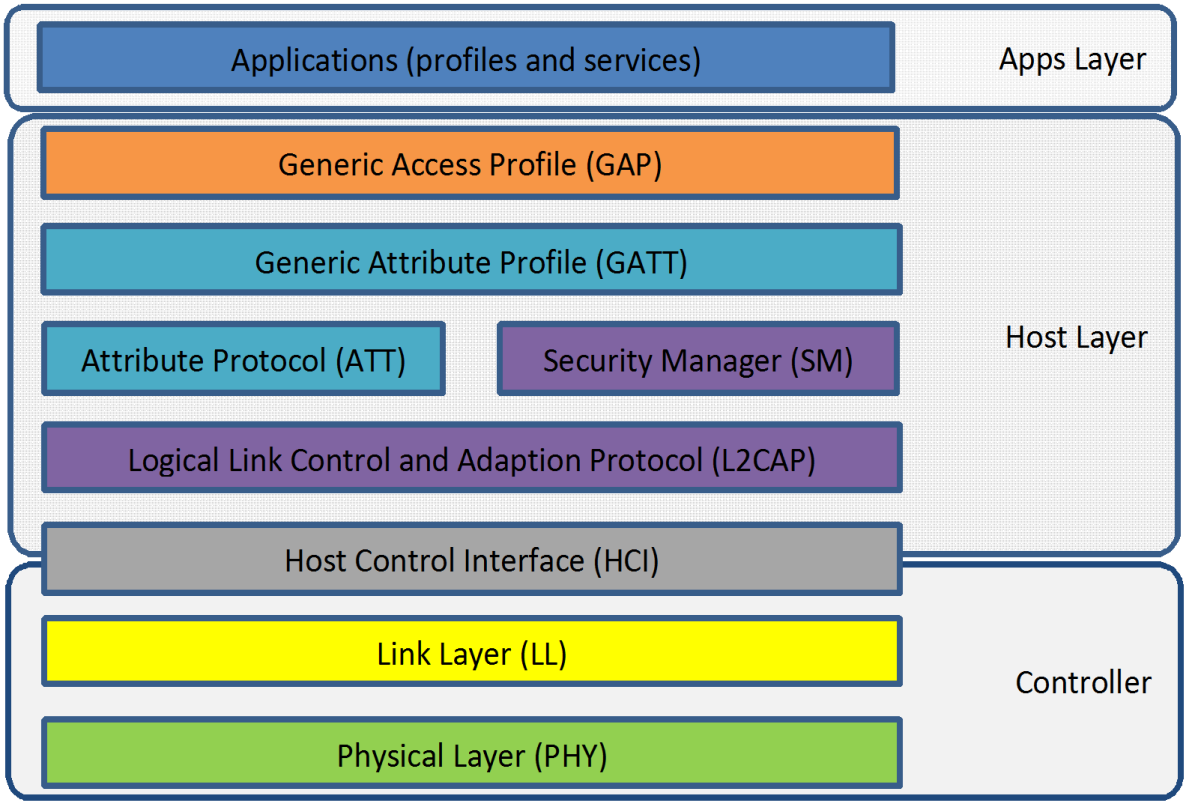


Figure 4.1 BLE Protocol Stack

Physical Layer (PHY) This layer is responsible for transmitting and receiving data over Radio Frequency channels. BLE uses the 2.4GHz ISM band. It has 3 advertising channels used for device discovery, connection establishment and broadcast transmission.

And it has another 37 data channels used for bidirectional communications. The channel diagram is shown in Figure 4.2 [46].

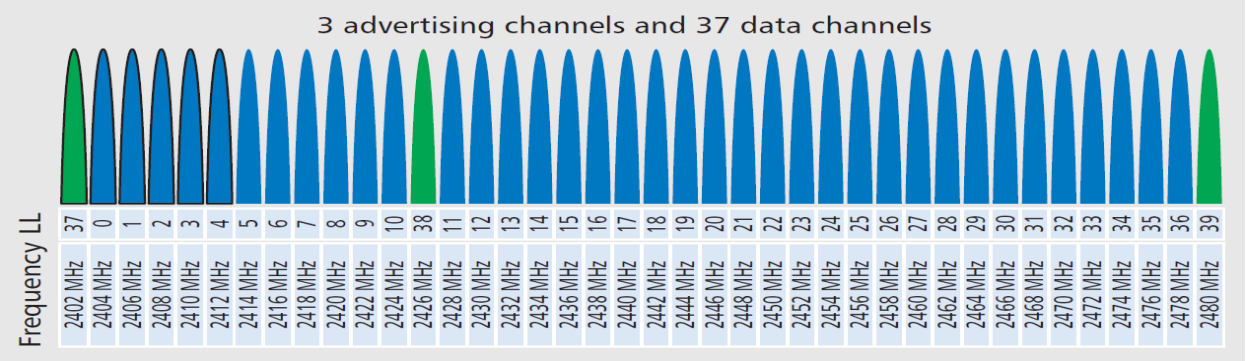


Figure 4.2 BLE Channel Diagram

Link Layer (LL) The operation of this layer can be described in terms of a state machine with 5 states as shown in Figure 4.3. The LL in the Standby State does not transmit or receive any packets. This state can be entered from any other state. In Advertising State, a device is referred as an advertiser, which will be transmitting advertising channel packets and possibly listening to and responding to responses triggered by these advertising channel packets. In Scanning State, a device is known as a scanner, which will be listening for advertising channel packets from advertiser. A device in the Initiating State is known as an initiator, which will initiate a connection with another device if it receives advertising channel packets from a specific device. When a device is in Connection State, it can have one of two roles: Master role or Slave role. After the connection is established between two devices, the initiator corresponds to role for Master while the advertiser corresponds to role for Slave. Also the Link Layer has specified only one packet format used for both advertising channel packets and data channel packets. This format is shown in Figure 4.4. The payload has the length between 2 to 257 octets which used to be 2 to 39 in previous versions.

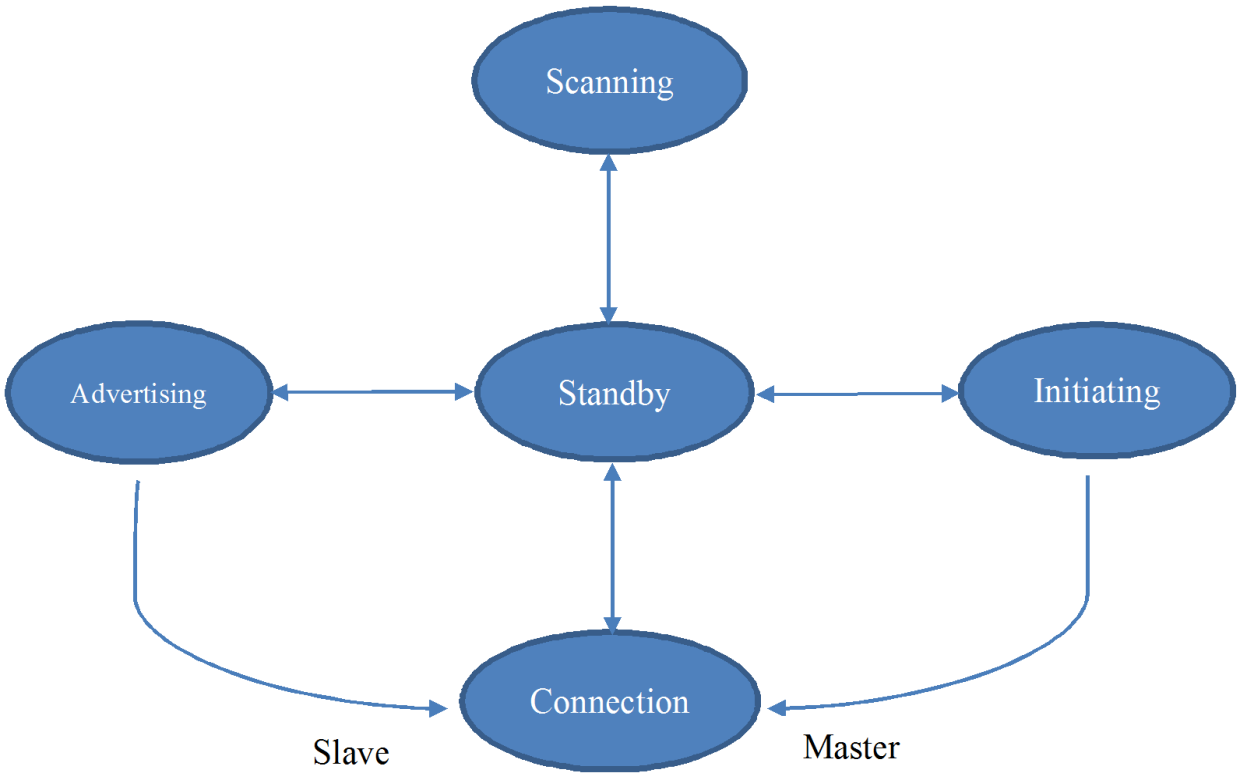


Figure 4.3 BLE LL States

Logical Link Control and Adaption Protocol (L2CAP) This layer supports multiplexing for higher level protocols, SMP, ATTA AND L2CAP LE signaling. The packet fragmentation and reassembly is done in L2CAP.

LSB MSB

Preamble	Access Address	PDU	CRC
(1 octet)	(4 octets)	(2 to 257 octets)	(3 octets)

Figure 4.4 LL Packet Format

Security Manager (SM) This layer defines the procedures and behavior of device pairing，authentication, encryption between the devices and key distribution. It is designed to minimize resource requirements for slave devices by shifting workload to more powerful master devices.

Attribute Protocol (ATT) The Attribute Protocol layer defines a Client/Server architecture above the BLE logical transport channel. This layer is optimized for small packet sizes, and allows an attribute server to expose attributes and their associated values to an attribute client.

Generic Attribute Profile (GATT) This layer establishes common operations and defines a generic service a framework for the data transported and stored by ATT. GATT and ATT are mandatory to implement in LE since it is used for discovering services. GATT defines two roles: Server and Client. GATT also specifies the format of data contained on the GATT server. Attributes, as transported by ATT are formatted as Services and Characteristics. GATT profiles specifies the structure in which profile data is exchanged. And they can minimize the size of data exchange between devices and hence reduce power consumption.

Generic Access Profile (GAP) GAP defines four specific roles: Broadcaster, Observer, Peripheral, and Central. The Broadcaster role is optimized for transmitter only applications while the Observer role is optimized for receiver only applications and receives advertisements from a broadcaster. Neither broadcaster nor observer support connections. The Peripheral role is optimized for devices that support a single connection and are less complex than central devices. And the Central role supports multiple simultaneous connections and is the initiator for the connections with a number of devices in the peripheral role. Since Bluetooth 4.1 was published, a device in the peripheral role is enabled to connect to multiple centrals instead of to be connected to a single central as before. This change provides the feasibility of forming a scatternet using BLE protocol.

Host Control Interface (HC) This interface separates the controller from host. The HCI layer implements a command, event and data interface to allow link layer access from upper layers such as L2CAP, SMP and GAP.

2) Basic terms

In BLE-based sensor networks, there are several terms need to be introduced. Some terms share the same definitions as in Bluetooth-based sensor networks, such as master, slave, piconet and scatternet. Some terms are only for BLE networks, for instance, initiator, advertiser, scanner and so forth.

A group of BLE devices within the same piconet occupy a shared physical channel and can act in one of two: one of the devices is the piconet master and the remaining devices become slaves. Piconets are formed in decentralized ad-hoc manner without intervention or assistance of any devices outside of the piconet. The master device defines the piconet physical channel characteristics using its Bluetooth Clock and Bluetooth Device Address. All of the slaves in a single piconet are connected to the piconet master. Due to the innovative improvement in Specification 4.1, BLE scatternet formation is enabled which means that communication between different piconets is possible. The scatternet is formed by two or more piconets where there are devices participating in multiple piconets and using time division multiplexing (TDM) to interleave their activity on each piconet physical channel. During the procedure of piconet formation, the advertising devices broadcast advertisement on the advertising channel and the scanning devices listen on the advertising channel. The listening devices can send connection requests packets to the advertiser and becomes the initiator for the connection. And once the initiator starts to initiate the connection with the advertiser, then the piconet is being formed where the

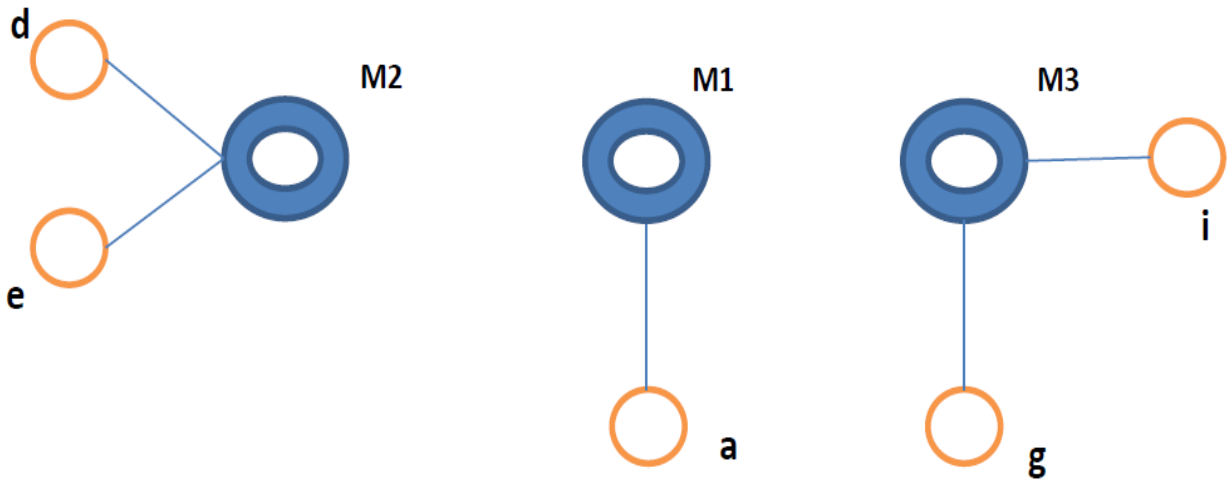
initiator becomes the piconet master and the advertiser becomes slaves. The master initiates the beginning of each connection event as well as can end each connection event at any time. One significant difference between Bluetooth specification 4.0 and specification 4.1 is that the former one does not allow dual mode for BLE devices and only allows a node to participate in one piconet which makes scatternet formation and multi-hop routing be impossible.

3) Piconet vs Scatternet

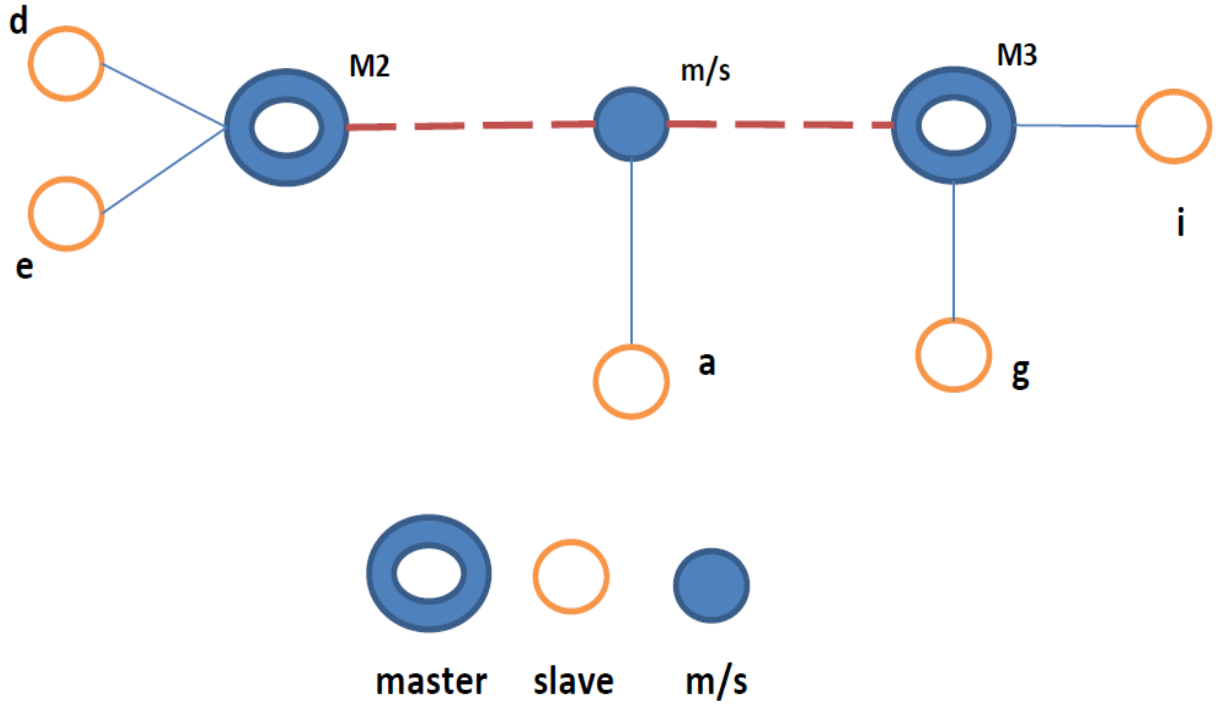
BLE devices can form a piconet and a scatternet. A piconet is the type of connection that is formed between two or more Bluetooth/BLE-enabled devices. In each piconet, there is only one device taking the role of master and all of the remaining devices are connected to it assuming a role of slave for synchronization. The physical distance between nodes in a single piconet is limited by the broadcast power of each device because all slaves in a piconet must be within direct communication range of the master.

A scatternet consists of a number of interconnected piconets which can be formed when a member, either the master or a slave, of one piconet participates as a slave in another piconet. A device participating in both piconets can act as a relay and forward data from one piconet to another one using TDM. Due to the improvement of the new version of Bluetooth specification 4.1, it is possible to form a large scatternet with numerous piconets and expand the size of BLE scatternet beyond BLE's limited communication range. Figure 4.5a and 4.5b show example BLE network topologies which demonstrate the difference between version 4.0 and version 4.1.

In the network shown in Figure 4.5a, device M2 takes the master role in a piconet where device d and e are slaves. Each slave in M2's piconet communicates with the master



a) BLE Topology of Specification 4.0



b) BLE topology of Specification 4.1

Figure 4.5 BLE Topology

on a separate physical channel. Devices M1 and a belong to another piconet with M1 as the master. Device M3, i and g belong to another piconet with M3 as the master. According to specification 4.0, a BLE device is not enabled to work in dual mode. Communication

between piconets is impossible and piconets cannot form a scatternet. After the improvement in specification 4.1 allowing BLE nodes to work in dual mode, a BLE device can act as a relay which can forward the data from one piconet to another piconet, forming a scatternet. As shown in Figure 4.5(b), these three separate piconets now can connect to each other to form a scatternet with the help the relay.

4) Establishing a Connection

In a BLE network, the communication between BLE nodes is event-driven. An event is time unit which is used to subdivide access to a physical channel. Data is transmitted in packets which are synchronized with these events. There are two events: advertising and connection events. At the beginning of each advertising event, the advertisers send an advertising packet on the advertising physical channel. Based on the type of advertising packet, the scanners, nodes listening for advertisements, may make a request to the advertisers even without the intention to connect to advertisers. If the devices listening for connectable advertising packets need to form a connection to advertiser, they are referred to as initiators. When initiators make a connection request to an advertiser and the advertiser receives and accepts this request, the advertising event will be ended and connection event will start. Once the connection is established, the initiator becomes the piconet master and the advertiser becomes the slave. The master and slave alternate sending data packets on the same physical channel achieved by active frequency hopping at the beginning of each connection event.

4.2 Formation of BLE Sensor Networks

In this section, we describe our new scatternet formation protocol generating a mesh network which is 1) dynamic, i.e., devices can participate in the network at an arbitrary time; 2) distributed, i.e., centralized decision-making is not required; and 3) supports the multi-hop scenarios. Our approach aims to achieve the following goals by reducing the path length: 1) reduce the delay; 2) reduce the number of piconets; 3) reduce the average number of master/slave bridges. In this work we assume that the initial scatternet will be formed at the beginning, although nodes could be added at any time. When a node needs to send data to another one, the routing protocol will be used.

1) Scatternet Formation

The scatternet formation procedure involves with two parts: piconet formation and getting local network information.

```
NodeDiscovery(){
1.S = Ø;
2.M = Ø;
3.startTimer(T);
4.while( T< Tmax ){
5.      neighbor = Advertise();
6.      if (neighbor  ≠ Ø)
7.           M = M ∪ neighbor;
8.}
9.startTimer(T);
10.       while( T< Tmax ){
11.            neighbor = Scan();
12.     if (neighbor  ≠ Ø)
13.          S = S ∪ neighbor;
14.       }
```

Figure 4.6 NodeDiscovery Algorithm

In our approach, each node maintains a list. Each master node keeps a list of its connected slaves in the same piconet, and each slave keeps a list of its connected masters in the slave's participating piconets. A node which acts as both a master and a slave will have

both slave and master lists. At the arrival of a new node, the new node will take turns to broadcast advertisements and scanning to listen for advertisements. If the new node receives a connection response when it is advertising, it will send a response back to the initiator in order to establish a connection and the new node is assigned to be a slave. If the new node receives an advertisement packet while it is scanning on the advertisement channel, it will initiate the connection and take the role of master. If the device enters the scan mode by calling the *Scan()* function, the device keeps listens for broadcasts of data from advertising. The scanning device can send a request to the advertiser to get additional user data, and send scan connection request to establish a connection. If it enters advertising mode by calling the *Advertise()* function, the device will broadcast advertising packets on one or more of the three advertising channels at a fixed interval. In return, the advertising device may receive scan requests or connection requests from listening devices. Through this advertising and scanning procedure, BLE devices can discover nearby devices and be discovered by devices in a given area. Newly arriving devices can participate in piconets to form a new piconet or a scatternet. Once a new node joins the existing network, then its connected master/slave node will update its slave/master list.

The pseudo-code for node discovery is shown in Figure 4.6. NodeDiscovery is a function which is executed by each node when it wants to enter the network by communicating with its neighbors. Although each neighbor can act as both master and slave, a node can only act in one role with respect to a single piconet. The NodeDiscovery function establishes connections between local nodes within direct communication range.

In the algorithm for NodeDiscovery shown in Figure 4.6, S is a list of slaves neighboring node the source node, and M is a list of masters neighboring the source node. T is a timer

used to limit the duration of the node discovery process. The algorithm begins by initializing the S and M sets. The algorithm continues with an Advertisement Phase (lines 4-7) during which it performs BLE advertisement, and a Scan Phase (lines 10-13) during which it performs BLE scanning. A timer T is used to limit the duration of each phase. Any neighbors which respond during advertisement are masters so they are added to M (line 7), and neighbors which respond during scanning are slaves so they are added to S (line 13). Table 4.4 shows an example master list and slave list for the example scatternet in Figure 4.3.

Table 4.4 Neighbor Lists

Nodes	Roles
s	slave
e	slave
a	slave relay
r1	m/s relay

a) M2's Slave List

Nodes	Roles
M2	master in piconet M2
M3	master in piconet M3

b) R1's Master List

4.3 Routing of BLE Sensor Network

In this section, we present our routing protocols enable the data sharing and forwarding in BLE networks.

4.3.1 An On-demand Multi-hop Routing Protocol

After the formation of scatternet, the master list of each slave and the slave list of each master has been determined and can be used for routing. The routing procedure consists of the following steps. First, the source node sends out a route request data packet to its piconet's master, including the destination node's information. The piconet master will look into its slave list. If the destination is not in its slave list, the master will initiate a breadth-first search by forwarding the route request to any slaves in its piconet which participate in another piconet. After receiving possible routes from its slaves, the shortest path is chosen. Each node has a route cache containing the most recently used routes. If the desired route is not in the route cache, then the route discovery process is used. Since each node's route cache is initially empty, the route discovery process is used to generate all routes.

The RouteDiscovery function is executed on a node when it needs to find a route to a destination node d. Invoking RouteDiscovery on a node can recursively invoke RouteDiscovery on its neighbors. In order to distinguish invocations on different nodes we prefix each invocation with the name of the node on which it is invoked. So the notation n.RouteDiscovery() indicates that the RouteDiscovery function is invoked on node n.

The task of RouteDiscovery is to return a route from the source node to the destination node. We define a route to be an ordered list of nodes where each adjacent pair of nodes is part of the same piconet. The RouteDiscovery function takes two arguments, a

route r from the source node to the node on which RouteDiscovery is invoked, and the destination node d.

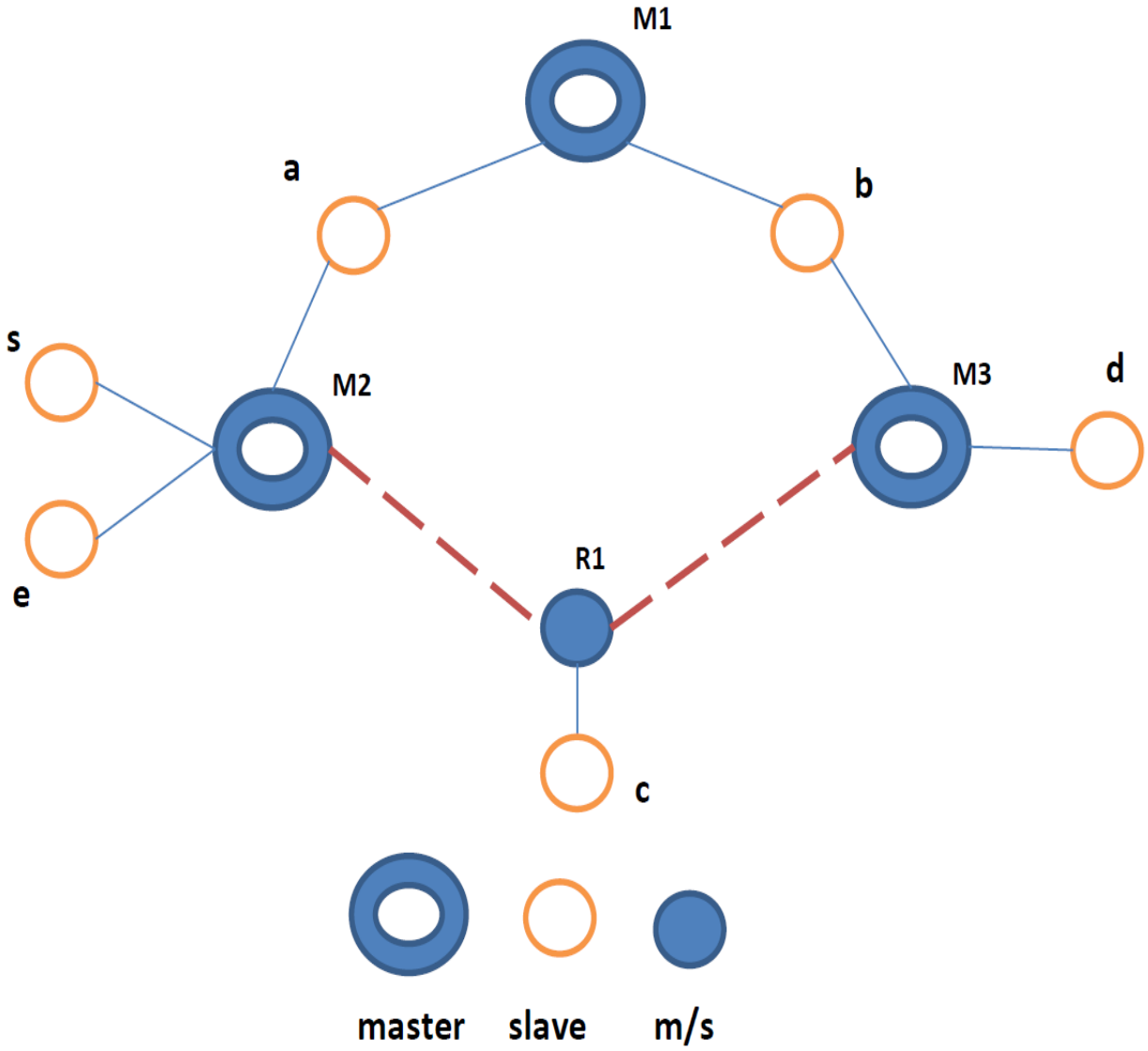


Figure 4.7 Routing Procedure

The RouteDiscovery algorithm shown in Figure 4.8 begins by initializing R, the set of routes found so far (line 1). The route cache is then checked to see if the route is the node's route cache (line 2). If the route is not found in the cache, then search loops are detected by checking if the node n is contained in the route so far (line 3). If node n is contained in

route r, then a loops is being explored so Ø is returned (line 4). Next, the destination is compared to all neighbors, either slave or master depending on the role of the node (line 5). If the destination is a neighbor then node n and the destination node d are appended to the route r and returned (line 6). If the destination is not a neighbor then the route request is forwarded first to neighboring master nodes (lines 7-8), and next to neighboring slaves which are masters of other piconets (lines 9-11). All paths found by forwarding the route requests to neighbors are placed in a list R and the path with the fewest hops is returned as the route (line 14).

```
n.RouteDiscovery( route r, destination d){
1.        R = Ø;
2.        if(checkRoute_Cache(r,d) is false){
3.          if(n∈r)
4.            return(Ø);
5.          if((d∈S) ∨(d∈M))
6.            return(r+n+d);
7.          foreach m∈M
8.            R = R∪s.RouteDiscovery(r+n,d);
9.          foreach s∈S
10.  if(s.isMaster())
11.    R = R∪s.RouteDiscovery(r+n,d);
12.       addRoute_Cache(shortestPath(R));
13.       }
14.     return(Route_Cache(r,d));
15.     }
```

Figure 4.8 RouteDiscovery Algorithm

As an example of the RouteDiscovery algorithm, we will describe the routing process which occurs in the network shown in Figure 4.7 when node s needs a route to node d. Since s is a slave it will forward the route request to its master M2. The request is forwarded recursively until node d is reached via two paths, one through node M1 and the

other through node R1. When s receives both routes, it compares those different routes and stores the shortest one in its route cache.

4.3.2 An Energy-aware On-demand Routing Approach (BSBP Algorithm)

The BSBR protocol is a reactive routing protocol based on DSR. Under the BLE protocol, after the formation of a scatternet each master node will keep a list of the slaves within its piconet and each slave node will maintain a list of masters for the piconets of which it is a member. In addition to this, our algorithm requires that each master node keep information on the residual battery power of each of its slave nodes.

When a source node needs to send packets to a destination node by calling the function *sendmsg( ),* it will first check its masters' route cache and choose the best route if one or more routes exist. If no route currently exists, route discovery is required to establish a route. Further, if a source node needs to send messages to a destination node, it must first check whether there exists route in its route table. Otherwise, it must also discover the route by calling the *RouteDiscovery( )* routine.

In the *RouteDiscovery( )* procedure the source node will broadcast a *RouteReques*t packet over the scatternet. A node which receives the *RouteRequest* will first check whether it is the intended destination. If it is not, then it is denoted bypass node. Bypass nodes must verify that their residual battery power meets the threshold specified in the *RouteRequest* before adding its ID to the "bypass node list" and forwarding the packet to adjacent nodes. If a bypass node does not meet the power threshold, it will mark itself as a failure node and add its residual power value to the *RouteRequest* packet. Infinite loops are avoided by requiring that bypass nodes check the bypass node list for their own IDs. If the ID of a bypass node is already present in the list it indicates the presence of a loop. In this

instance, the broadcast packet is discarded. When the destination node receives the *RouteRequest* packet it will extract the bypass node list, which contains the final route, and add it to the *RouteResponse* packet. The *RouteResponse* packet will be transmitted along the bypass nodes in reverse order until the source node is reached.

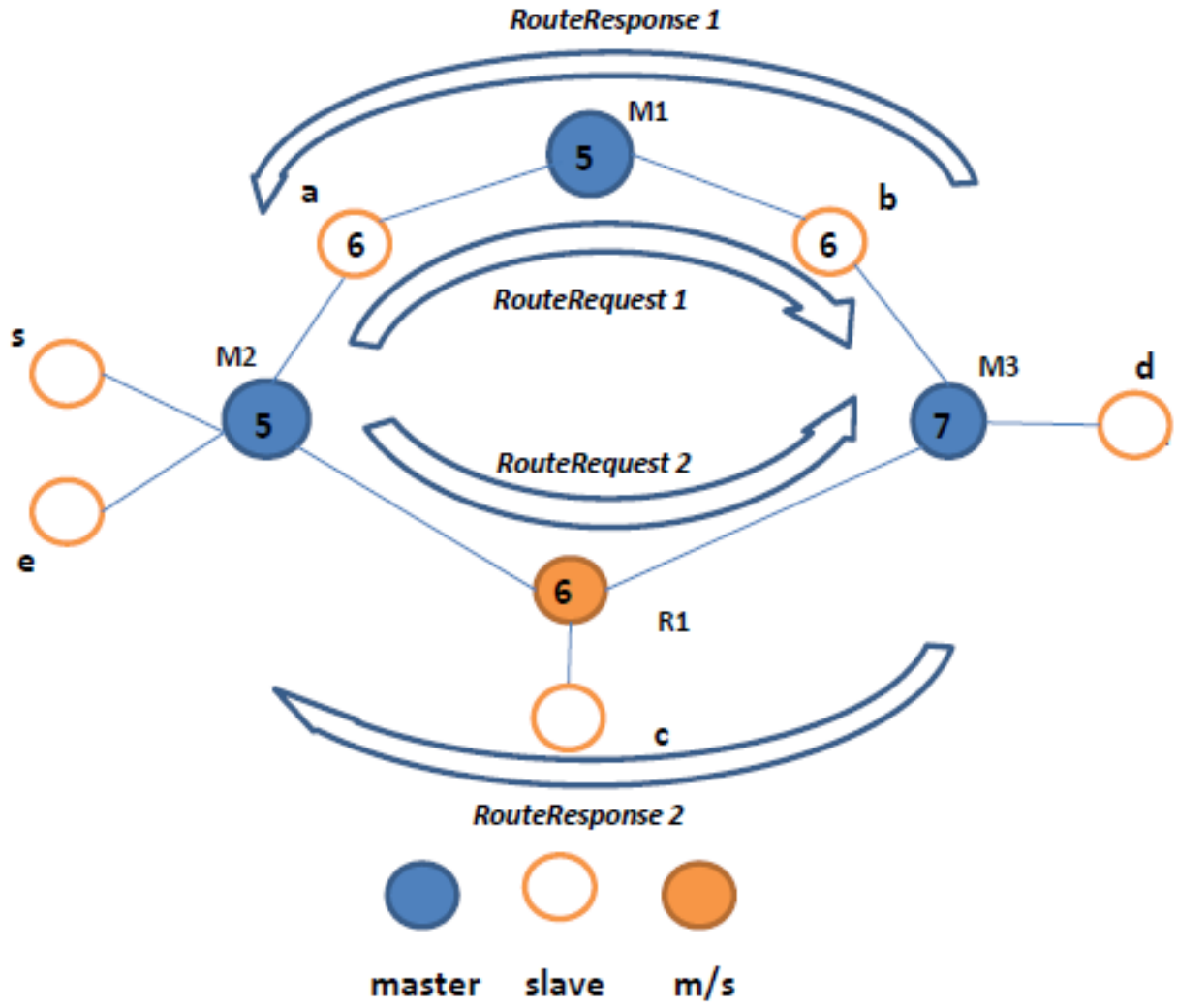


Figure 4.9 Basic Route Discovery Procedure

If there is no available route from source node to destination node due to a power constraint (even after the threshold value is adjusted to a lowest tolerable value) then the source node will split the workload among multiple routes. These routes utilize nodes

which are marked as failure nodes but are still within the *bypass node list*. Although these failure nodes have insufficient power to transmit the entire message, they may have enough power to transmit a portion of the message. Multiple routes will be utilized to ensure that the sum power available for all routes meets the power threshold for the full message. In this way, our algorithm will ensure successful message delivery even in the event of bypass nodes with low battery power.

Figure 4.9 shows a simple example of the route discovery procedure when each bypass node adequately meets the power threshold. The power threshold in this example is denoted as 5 (not shown). Nodes can hold one of three designations: master, slave, or master/slave. In this figure, s is the source node which broadcasts the RouteRequest packet, and d is the destination node. As the master of node s, node M2 will first determine whether node d is in its slave list. If not, M2 will add its ID to bypass node list of the RouteRequest packet along with its residual battery power of 5 and the difference between the residual battery power and required power threshold (0 in this case). M2 then forwards the RouteRequest to its adjacent neighbor piconets M1 and R1. In a similar manner the RouteRequest message will be forwarded to other piconets until node d receives the RouteRequest. In this example, node d will receive two RouteRequest packets. Node d will read the bypass nodes list {(M2, 5, 0), (R1, 6, 1), (M3, 7, 2)} from one RouteRequest packet and the list {(M2, 5, 0), (a, 6, 1), (M1, 5, 0), (b, 6, 1), (M3, 7, 2)} from the other. Node d will add these route lists to the RouteResponse packets and return the RouteResponse packets to node s using the respective route list. When s receives the RouteResponse, s will sort the route by hops and by residual battery power, updating its

route table. After this is done, node s will send the message to destination d using the best available route.

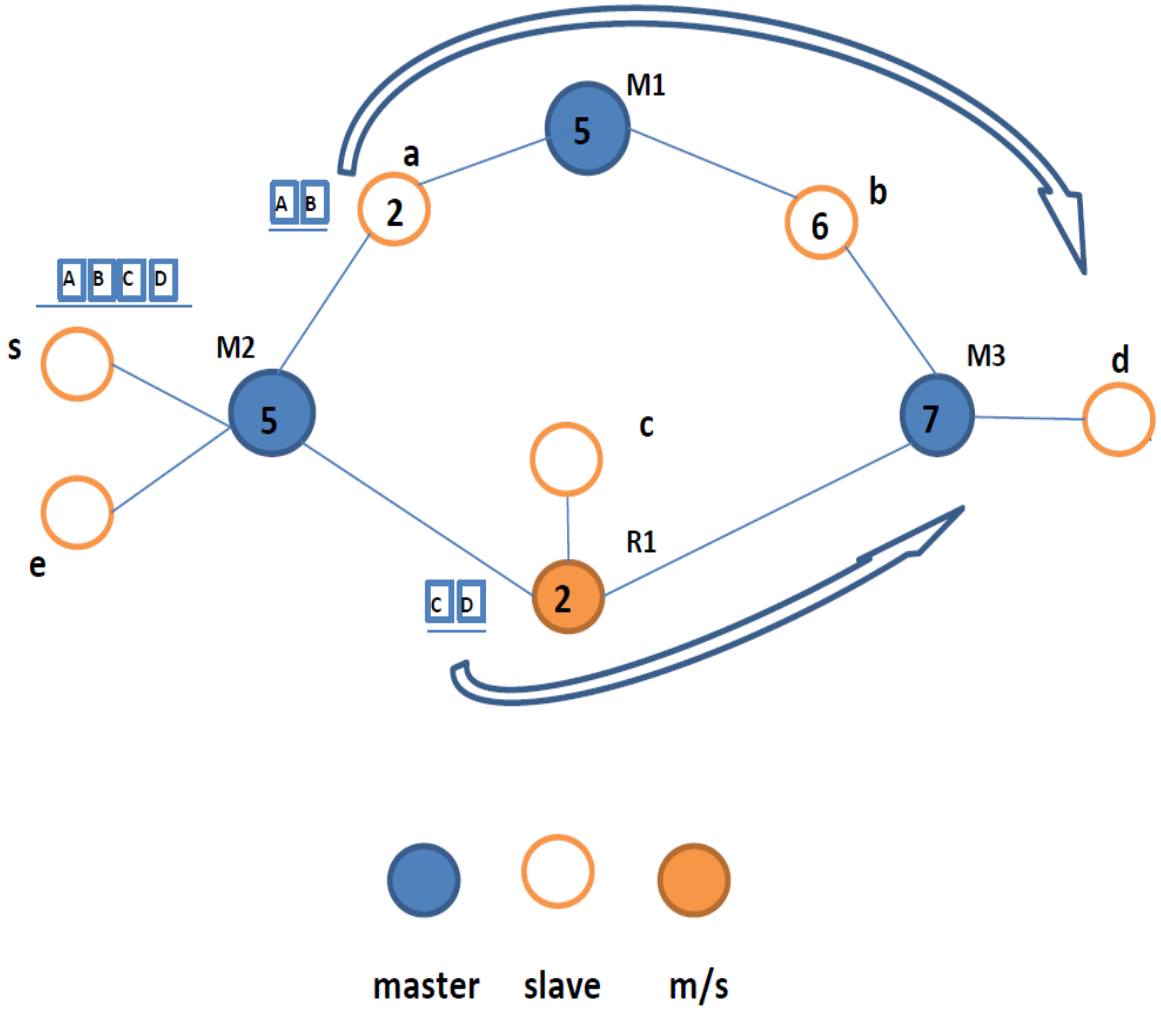


Figure 4.10 Route Discovery Procedure with Workload Split

Figure 4.10 shows an example of route discovery when a failure node is present. The power threshold for this example is again denoted as 5. Both possible paths fail to meet this minimum power threshold due to a low battery condition of nodes R1 and a. Consequently, R1 and a are marked as failure nodes. The route list in the *RouteResponse* packets received by s will be {(M2, 5, 0), (R1, 2, -3), (M3, 7, 2)} and {(M2, 5, 0), (a, 2, -3), (M1, 5, 0), (b, 6, 1), (M3, 7, 2)}. The entry corresponding to each failure node contains a negative number

indicating the difference between available power and threshold. Based on the power information in each route list, node s will determine that power constraints will be met if the message data is split into two smaller units which can each be sent along different paths.

```
x$RouteDiscovery ( d, Pt, R){
1.    if x==d
2.       append (R, x, Pr-Pt );
3.       createRouteResponse(R);
4.       sendonRoute(RouteResponse);
5.       return;
6.    if x$master == d
7.       append (R, x$master, Pr-Pt);
8.       createRouteResponse(R);
9.       sendonRoute(RouteResponse);
10.      return;
11.   if checkbypassnode (x,R) is true
12.       discard(RouteRequest);
13.   else
14.       foreach slave of x.master
15.             slave$RouteDiscovery (d, Pt, R);
            }
```

Figure 4.11 Pseudocode of *RouteDiscovery( )*

Figure 4.11 shows the pseudocode of the *RouteDiscovery( )* function. This function takes the following arguments: destination node s, power threshold Pt and route list R. In Figure 4.11, the $ symbol represents the node executing the discovery procedure. *RouteDiscovery* determines if the current node x is the expected destination node d in Line 2. If node x is the destination node, then x's ID together with the difference between its residual battery power Pr and the power threshold will be appended to the route list R. As shown in Lines 2 to 4, the complete route information will be used to generate a

RouteResponse packet which will be returned to the source node along the reverse order of route R. If node x is not the destination node, then x's master will be queried in a similar manner. This is shown in lines 7 through 9. Line 11 checks for cycles by determining if node x is already present in the route by examining the bypass node list of R. If no cycle is detected, then the remaining slave nodes of x's master will be recursively queried until the destination node d is discovered or the time period runs out. This is shown in lines 14 and 15.

```
routeSort(routelist R, destination d){
1.  if (route_list R == Ø or check (R,d) is
    false)
2.      exit (0);
3.  else{
4.      Success = Ø;
5.      Failure = Ø;
6.      for each route to destination d in R {
7.          if ( checkpower ( route, Pr) > = 0)
8.              Success = Success ∪ route;
9.          else
10.             Failure = Failure ∪ route;
11.     removeRoute (R, route);
        }
    }
12. sortbyhops (Success, ascending);
13. sortbypwr (Success, descending);
14. sortbypwr (Failure, descending);
15. Combination (Failure, Pt);
16. R = R ∪ (Success + Failure);
17. return R;
}
```

Figure 4.12 Pseudocode of *routeSort( )* running at source node

After route discovery is complete, node s will collect the route information to destination to d from all *RouteResponse* messages. The source node will then cache the

routes in its route table for subsequent use. These routes will be sorted and the one with the lowest cost will be used to send the data. The sort procedure initially divides routes into two categories: routes meeting the power requirement (power success routes) and those which do not (power failure routes). The routine sorts the power success routes first by the number of hops and then by residual battery power. The power failure routes are sorted in descending order based on the negative value of the power difference. The combination of the top several routes whose battery power sum meets the power threshold will be together selected to split the message workload.

Figure 4.12 shows pseudocode for *routeSort( )*. Line 1 checks whether there are any potential routes from source s to destination d in the route list R. These routes are divided into two groups based on the power threshold Pt in lines 7 through 10. Success is the set used to store the routes for which all of the bypass nodes meet the power threshold. Failure is the set used to store the routes for which one or more bypass nodes fail to meet the power threshold and are labeled failure nodes. Line 11 remove the route which has been checked from routelist R. Line 12 sort the routes in Success in ascending order by number of hops. Line 13 sort the routes again in descending order by the minimum battery power of nodes in each route. Line 14 processes Failure by sorting the routes in a descending order by the battery power of the failure nodes. Line 15 calls the combination function, used to select multiple routes in Failure such that the sum of the battery power of failures node from each route meets the power threshold. These routes will later be used to split the workload. In line 16, the route list R is updated by replacing R with the concatenated lists of newly sorted routes.

4.3.3 Evaluation

In this subsection we evaluate our experimental results. Instead of simulating the results on platform such as ns-3 [16], our protocol is implemented on real hardware, the Broadcom combo chipset BCM434X. Our use of real hardware produces more accurate results when compared to the use of simulation which does not fully support critical features of the Bluetooth specification and cannot mimic real world complexities such as clock synchronization and scheduling. The combo chipset we utilize is well accepted in the marketplace and is integrated into a number of popular products including the MacBook Air and Iphone 6. A high-level schematic of the test board used, which includes the chipset, is shown in Figure 4.13. This chipset supports Bluetooth Core Specification 4.1, Bluetooth low energy, and provides low power at low cost for wearable devices.

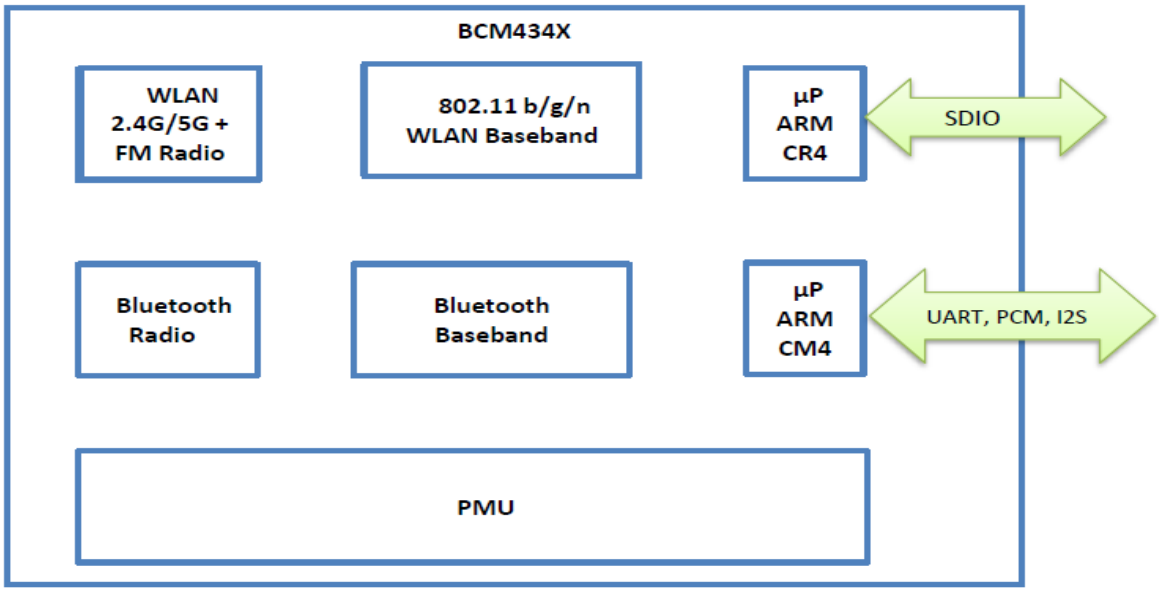


Figure 4.13 BCM 434x Chipset Test Board

A. Formation and On-demand Routing

We perform a set of experiments in order to evaluate the quality of our scatternet formation and routing approach. In each experiment a number nodes are placed randomly

in a 5x15 square meter area. Over the set of experiments, we vary the number of nodes from 5 to 10. Scatternet formation is performed by having each node execute NodeDiscovery. RouteDiscovery is executed each time a new route is required. Each message is 1 Kbits large and messages are issued at a rate of 1Mbits per second. For each experiment we evaluate two metrics of the network topology, number of piconets and average piconet size, and two performance metrics, average message delay (source to destination) and average network throughput.

1) Number of Piconets, Size of Piconets

Figure 4.14 shows the average number of piconets in a scatternet. As the number of nodes increases, those nodes are partitioned into a higher number of piconets. The average number of slaves in each piconet is showed in Figure 4.15. It is clear that the number of nodes per piconet is fairly constant over the range of network sizes which we have evaluated.

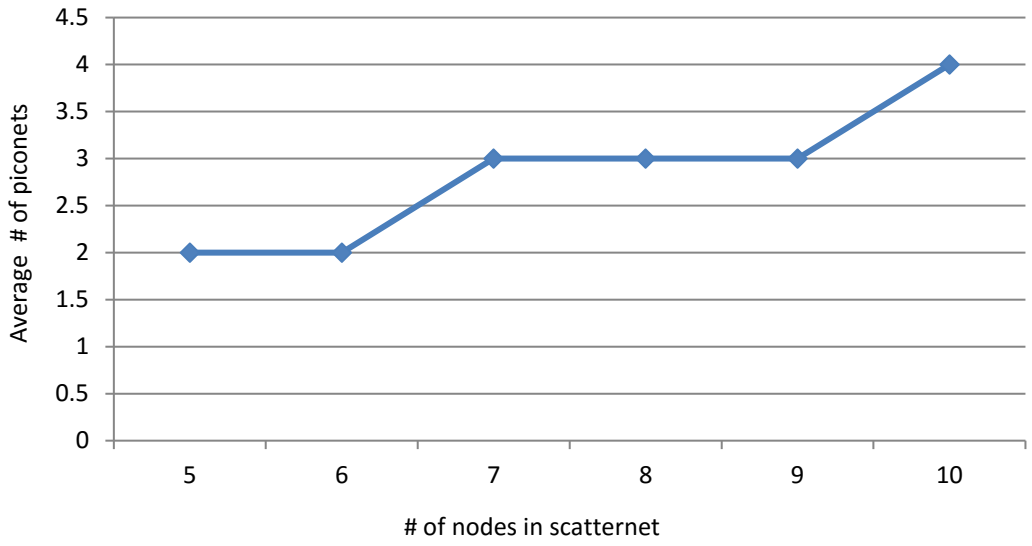


Figure 4.14 Average Number of Piconets

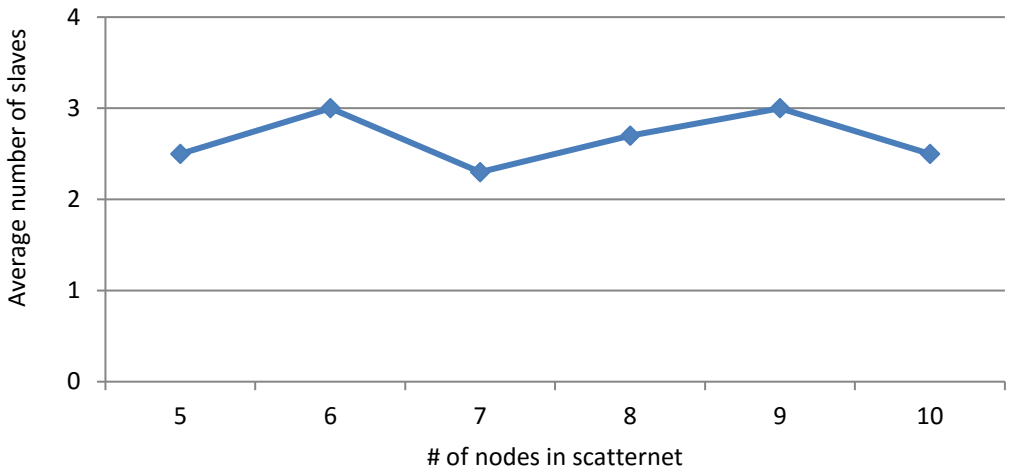


Figure 4.15 Average Number of Slaves per Piconet

2) Total Delay

The delay in a network is the time taken for a data packet to traverse through the network and reach the destination. The average delay for a source node to send a 1Kb data packet to a destination node is shown in Figure 4.16.

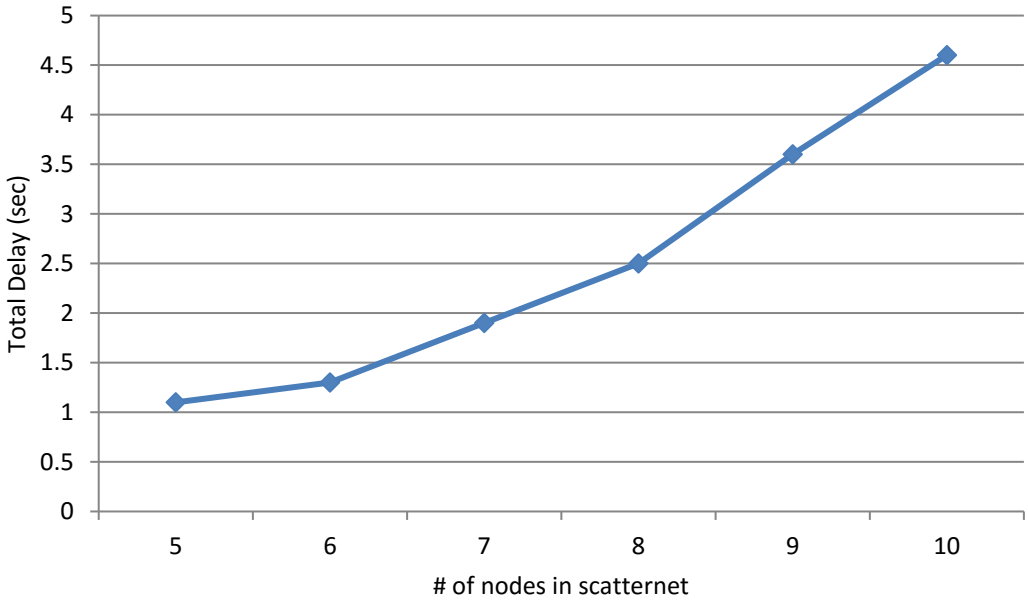


Figure 4.16 Average Delay

3) Throughput

Figure 4.17 shows the network throughput achieved using our on-demand routing protocol for BLE scatternet.

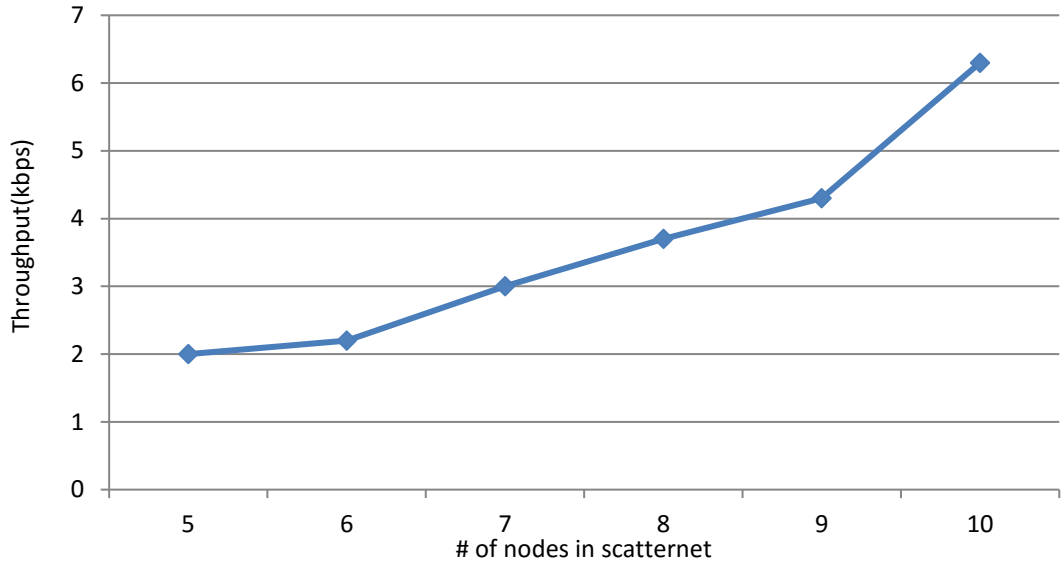


Figure 4.17 Throughput of Scatternet

In this section, we have described a novel scatternet formation and multi-hop routing protocol used for BLE-based sensor network, based on the new features in Bluetooth Specification version 4.1. We implemented this protocol using real hardware. Our protocol does not incur large delay and achieves better resource utilization by shortening the route path.

B. BSBR

As for our BSBR algorithm, we perform a set of experiments in order to characterize our routing approach and scatternet formation. In each experiment a number nodes are placed randomly in a 5x15 square meter area. Over the set of experiments, we vary the number of piconets from 2 to 7. The energy capacity of each participating node is represented with an integer value between 1 watt-hour (Wh) and 10 watt-hours (Wh), which is associated with battery energy. For each experiment we evaluate three

performance metrics, average message delay (source to destination), network lifetime, and throughput.

In order to evaluate the performance, we have the following assumptions. The power consumption rate is set to be 0.005 Wh per min when a node wakes up and 0.001 Wh per min when it sleeps.

The following results show the differences between our battery-aware algorithm and the regular on-demand routing algorithm implemented previously. In the following figures, the Battery-aware trace represents our battery-aware algorithm while the on-demand trace represents the on-demand BLE-based routing presented in previous section.

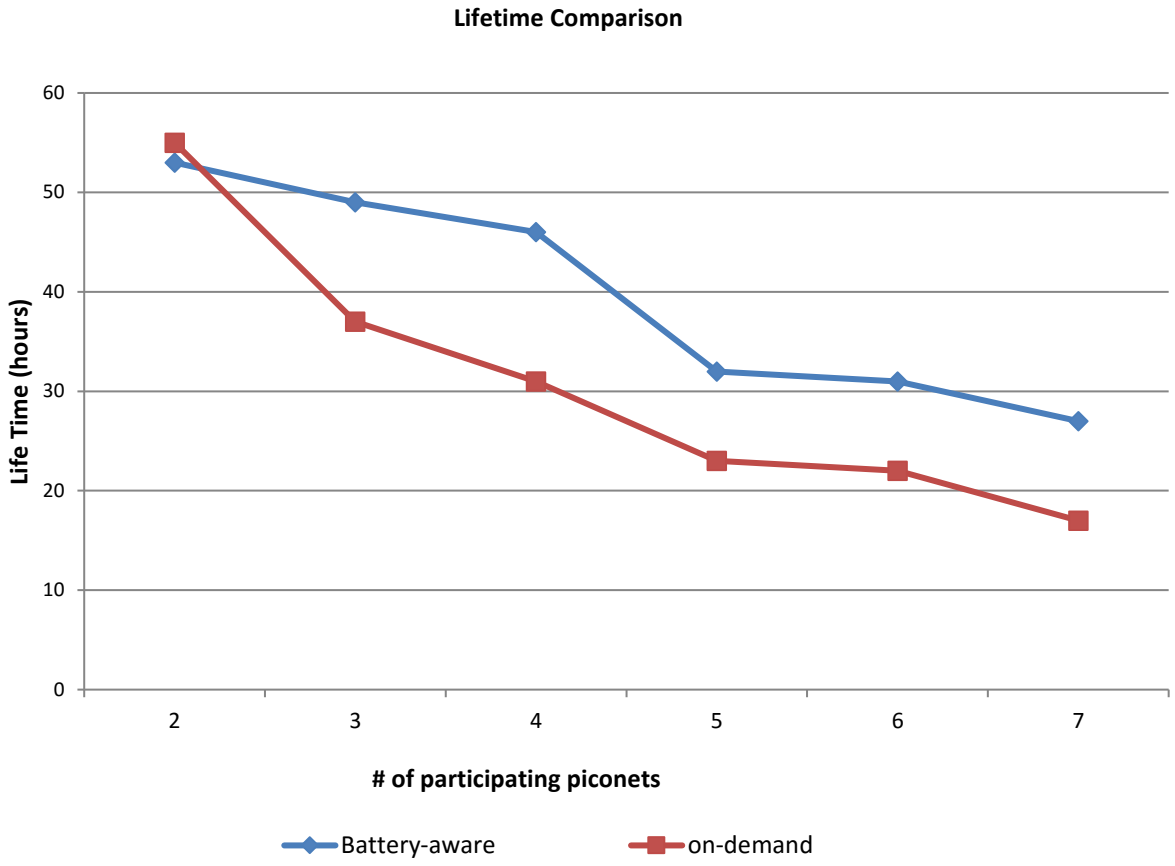


Figure 4.18 Lifetime Comparison

Figure 4.18 compares the lifetimes of the BLE scatternets. Lifetime is the defined time from when a route is first established until there are no available routes from the source node to destination node. This indicates that one or more of the path nodes is depleted of energy. If a scatternet is alive, it should meet the following requirements:

- All nodes which transmit/receive messages must maintain battery levels at or above the threshold.
- At least one route exists for a source-destination pair such that all intermediate path nodes have sufficient power to guarantee the successful delivery of messages.

Over this scatternet, we randomly choose several nodes as source and destination pairs which are in two different piconets and are located at different ends of the scatternet.

The source node continues to transmit packets from source node to destination after a route is established until a route can no longer be maintained due to insufficient power in a critical path node. We then reset the network restart the trial with a new source/destination pair. In this way, we calculate the average lifetime of the scatternet. The lifetime calculation procedure is described in Figure 4.19. Input P is a set of source/destination pairs and m is the test message. Line 1 initializes the system and resets the residual battery power indicator to a default value. Each pair in P will be used to send the test message once a route is discovered and connection is established. When the route is broken, it will attempt to use the next available route in route table (shown in line 8) until all routes are explored. Once all routes are exhausted the value of Timer is recorded and stored in the timer list T_{list} in line 13. All nodes are then reinitialized to default status

and next pair in P will be tested. Line 15 calculates the average of T_{list} which is denoted as the average lifetime of the scatternet.

```
LifetimeCal (source_destination_pair P, m){
1.  Init( );
2.  foreach (source s, destination d) pair in P {
3.      s$RouteDiscovery (d, Pt, R);
4.      routeSort (R, d);
5.      while (R->head != R-> bottom
    or !DisconnectSignal) {
6.          initTimer(T);
7.          if (DisconnectSignal)
8.                  R-> head = R->head->next;
9.           if ( R-> head = R-> bottom )
10.                  break;
11.          sendmsg(s, d, R, m);
        }
12.     recordTimer(T);
13.     Tlist += T;
14.     Init();
    }
15. Return Average(Tlist);
}
```

Figure 4.19 Pesudocode of Scatternet Lifetime Calculation

From the presented results, we concluded that our BSBR algorithm achieves longer lifetime as the size of the scatternet increases. On average, our algorithm prolongs the lifetime by 16%. This is observed to be due to the algorithm choosing the best route with highest residual battery energy at the beginning, in addition to using multiple nodes to share the transmission workload at the tail end of the algorithm. This combines to achieve the superior lifetimes observed for the scatternet.

Since the network throughput represents the rate of successful message delivery, we use this metric to evaluate the performance efficiency. We calculate the scatternet

throughput by dividing the overall data packets received from the moment when the first message is sent until all messages have been received. This result is presented in Figure 4.20. From the data we observe that our battery-aware approach achieves better performance by up to 50%. This is especially pronounced as the size of scatternet increases. The longer the lifetime of the scatternet, the more data which will be successfully delivered on the data channel.

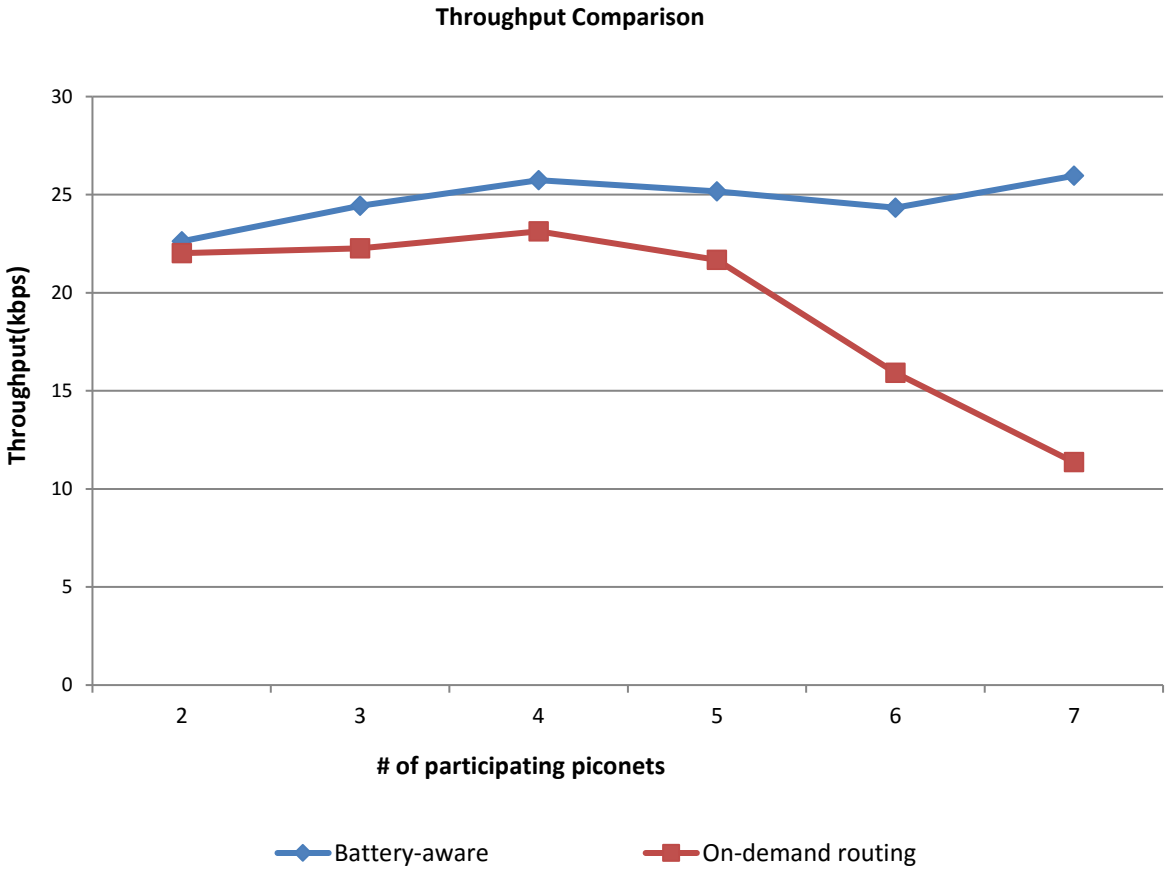


Figure 4.20 Throughput Comparison

In addition, we use attenuators to mimic the disturbance and the negative effects on scatternet performance caused by distance and physical obstructions in order to compare the robustness of our battery-aware approach with previous work. The attenuator is an

electronic device reducing the amplitude of the electronic signal in such a way that it emulates a disturbance or obstruction in the network. Figure 4.21 shows that as the attenuation level increases our battery-aware approach achieves consistently better throughput compared to a non-battery aware on-demand algorithm.

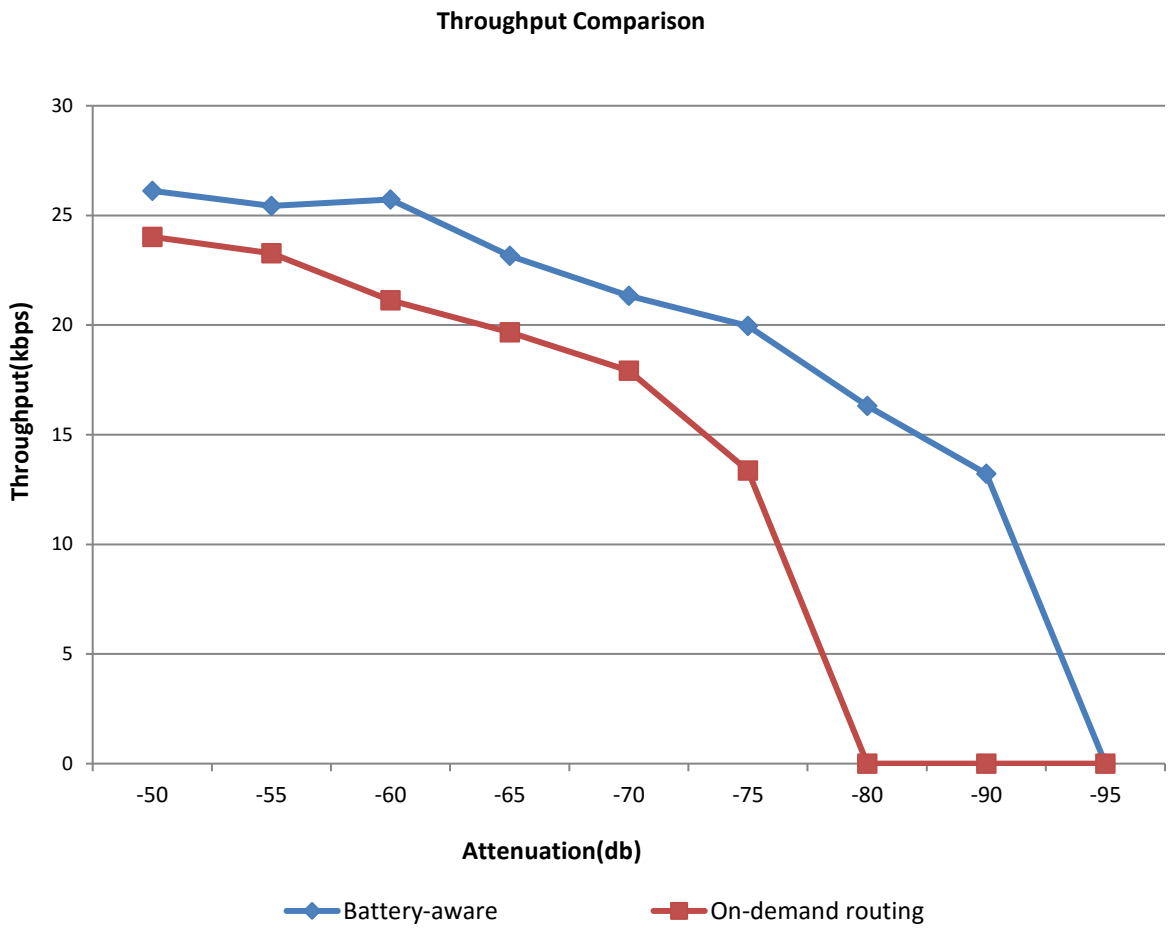


Figure 4.21 Throughput Comparison

Finally, we compare the delay between our approach and a strictly on-demand algorithm. The result is shown in Figure 4.22. As our battery-aware approach integrates with battery checking and balance of workload considerations, it exhibits a longer delay as a trade-off. This is both expected and acceptable due to the fact that it not only prolongs the lifetime but also increases the performance efficiency and robustness of the scatternet.

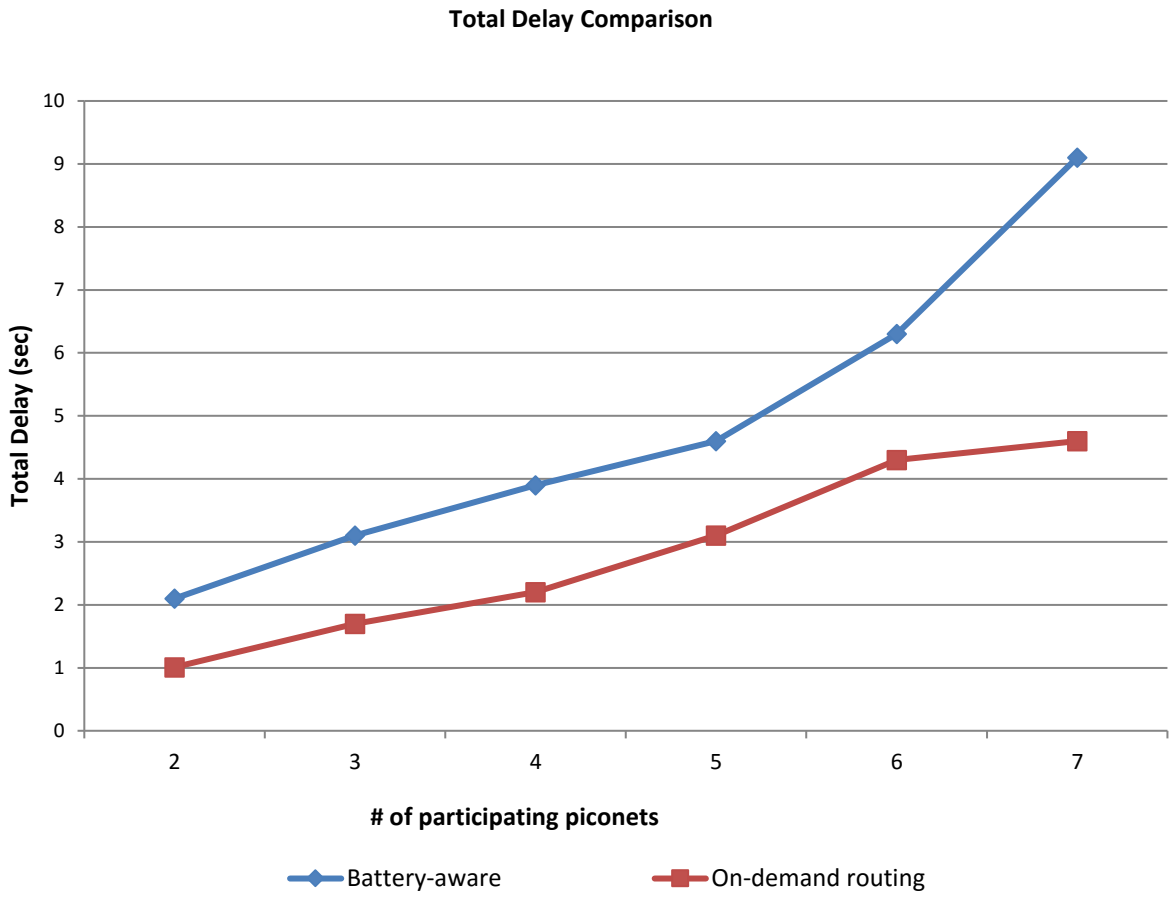


Figure 4.22 Delay Comparison

In this section, we have presented a novel battery aware multi-hop routing algorithm specifically designed for BLE-based sensor networks. This approach achieves better performance and prolongs the lifetime of mesh network significantly.

4.4 Security of BLE based IoT

This section will describe in detail how the attack works and how the attacker forces the network to consume power.

BLE security mechanism is enabled to protect communication between devices at different levels of the stack. As for common types of attacks against wireless communication networks, such as Man-in-the-Middle (MITM) attack, passive eavesdropping and privacy tracking, BLE efficiently addresses them.

A) MITM

Attacker needs to have the ability to secretly relays and possibly alters messages transmitting on a communication channel between two parties. Active eavesdropping is a typical example, in which the attacker makes independent connections with the victims and relays messages between them to make them believe they are talking directly to each other over a private connection, when in fact the entire conversation is controlled by the attacker. The attacker must be able to intercept all relevant messages passing between the two victims and inject new ones.

BLE protocol deals with MITM in the following ways. Prior to 4.2, devices can choose between Just Works, Passskey Entry and Out of Band. In LE legacy pairing, MITM protection could be obtained by either using the Passkey Entry pairing method or using the Out of Band pairing method. In addition, Bluetooth 4.2 introduces a new security model, LE Secure Connections. In LE secure connections pairing, besides using the previous two methods, MITM protection could be obtained by using the Numeric Comparison method. To ensure that the protection from MITM through authentication is generated, the selected authentication requirements option is required to have MITM protection specified.

B) *Passive Eavesdropping*

Passive Eavesdropping is to use a sniffing device to secretly listen to the private communication of others without consent. LE Secure Connections uses an algorithm called

Elliptic curve Diffie–Hellman (ECDH) for key generation, and a new pairing procedure for the key exchange. This ECDH public key cryptography is a means to thwart passive eavesdropping attacks.

C) Privacy Tracking:

Most of IoT devices, in their normal operation, broadcast constantly. And these devices can be detected at a great range of 100m in an open area. Scanning for these broadcasts is easy which allows attacker to identify and locate particular devices within a limited range.

Most of the BLE advertisement and data packets have the source addresses of the devices which are sending the data, third-party devices could associate these addresses to the identity of a user and track the user by that address. BLE can protect against this potential attacks by frequently changing private addresses so that only the trusted parties could resolve them.

With those new features added, BLE protocol has a better mechanism to defend against these common attacks to wireless communication networks, however its vulnerability to DoS attacks aiming to accelerate battery depletion remains unchanged. And therefore our research is concentrated on this field of finding an effective approach to defend BLE sensor networks against battery exhaustion attack.

4.4.1 Threat Model

Here we elaborate the levels of battery drain attacks with the below three degrees.

1) Level 1: This takes place before the establishment of connections. Though the attacker is not allowed to participate into the mesh network and cannot request services, it can repeatedly send connection/disconnection requests to the victims.

2) Level 2: This takes places after the attacker get the access to the mesh network. The attacker sends service requests which are not provided by the mesh network repeatedly to deplete victims' batteries.

3) Level 3: To be different from level 2, the attacker sends service requests which are allowed by the mesh network repeatedly to deplete victims' batteries, which makes it hard to be detected.

In order to defend the mesh networks against those three levels of threats, our approach consists of three procedures accordingly. Firstly, we require all of the new devices to register themselves before they participate into the networks. This serves to stop the level 1 attacks from unknown devices. Meanwhile, the mesh network maintains a blacklist filled with the unique bluetooth address of the malicious nodes. Secondly, the service requested by the newly participated node will be verified. If the node keeps sending the same service request which is not provided by the mesh network, this new node will be removed from the registration list and regarded as unknown node. This step serves to prevent the level 2 attack. Thirdly, the mesh network maintains a service priority list. The priority of devices which have their requested services completed will be lowered.

4.4.2 Detection and Prevention Approach

In order to launch the battery exhaustion attacks to the target varying from level 1 to level 3 to the target, the attacker needs to repeatedly request service from the target node like pairing with the victim devices by sending advertisement before the establishment of the connection. Also the attacker may repeatedly send specific service request to the target mesh network like using a printer after the connection is established. In order to avoid this, we propose the algorithm which assigns a priority sequence number

to the new node intending to join the existing network and requires the new node to switch the role with the node which receives the request in a pre-determined time period based on the need.

In the design of our mesh network, there is a guard node playing the role of service administrator (SA) maintaining a service priority list. All new nodes which request services need to be registered with the SA. During the registration procedure, the administrator will first check what kind of service this new node is requesting. If the service this new node requesting is not supported by the network, the administrator node will ignore this node and not assign a service priority sequence number to it. In this way, this new node's address will not be included in service priority list. This priority service list will be shared by all of the nodes in the mesh network. Once there are new changes made by the service administrator, it will broadcast the announcement to the network to have the nodes update their service list. When a new node which is not contained in the service list keeps sending a request to the network, this new node will be blacklisted.

Figure 4.23 shows the procedure of a new node participating in the network after being registered at service administrator. The scatternet shown in Figure 4.17 consists of three piconets- M1, M2 and M3. SA is the service administrator driven by AC power instead of battery, which all of the new nodes need to register at. SA keeps the original priority service list and it is mutual slave to piconet M1, M2 and M3.

After this new node is assigned with a priority sequence number, it will be allowed to communicate with the network. And other node in the mesh network will take turns to contact with this new node until the requested service is done. After the service is finished,

the priority of this new node will be put at the bottom of the service list. Otherwise, this new node will be blacklisted based on our algorithm.

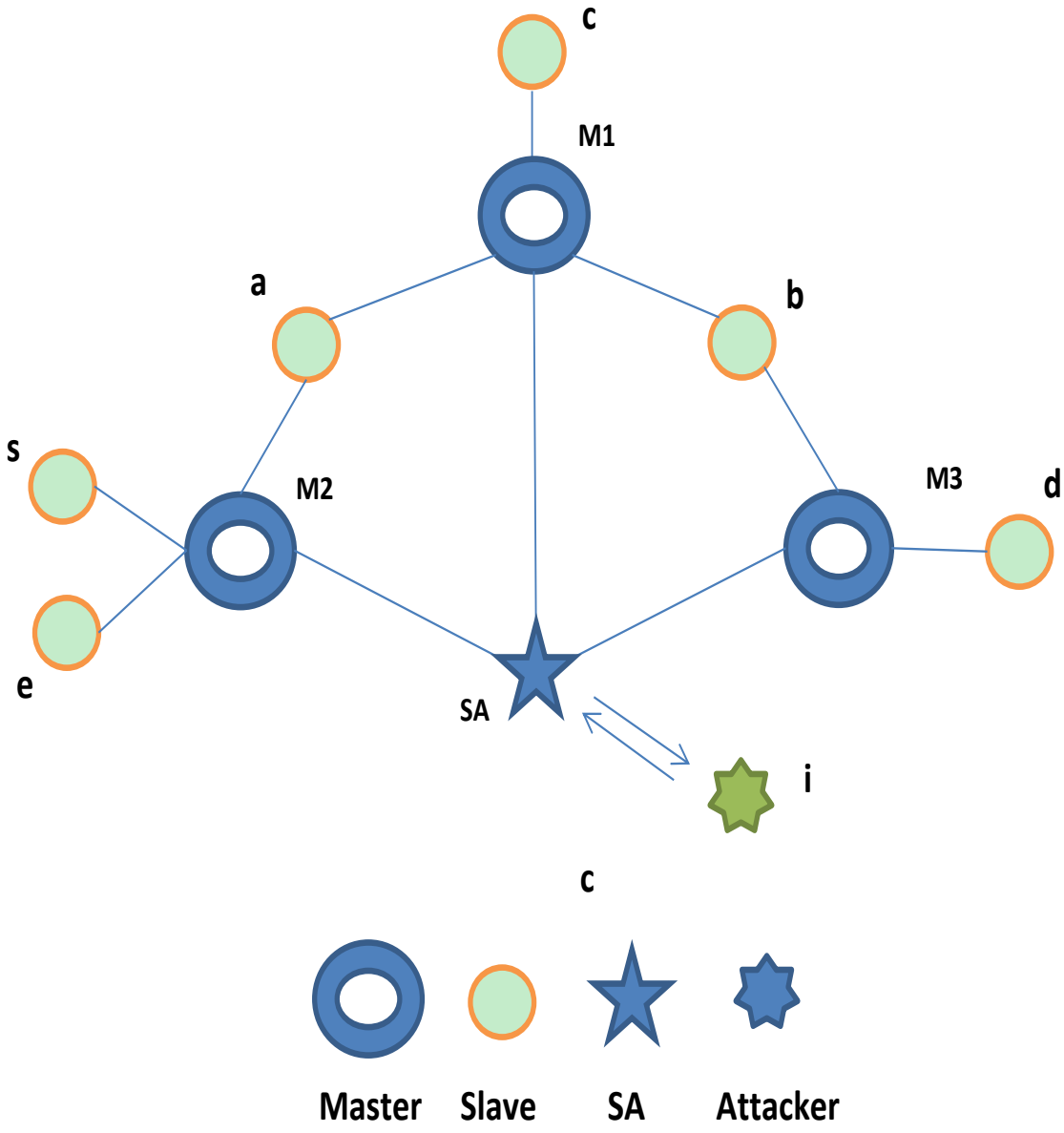


Figure 4.23 Registration Procedure

Figure 4.24 shows our algorithm dealing with the new node when it is registered successfully at the administrator end and assigned a service sequence number. If N is the potential attacker, it will keep sending advertisements to the target device or BLE network

without keep scanning on the advertisement channel and responding to the request from other devices.

```
S.nodeValidate( N ) {
1.    if ( N.addr belongs to Blacklist R or N.service is not included in Servicelist L )
2.        Ignore (N);
3.    else
4.    {  establishConnection( S, N );
5.       While pre_timer1 do
6.           communicate (N, S);
         end while
7.       endConnection (S, N);
8.       if (N.service is completed)
9.          return update_PriorityList(L);
10.      else
11       {   K = leastBusyNeighbor(S);
12.         K. init ( );
13.         S. notify (N);
14.         Timer(pre_timer2);
15.         if ( K receives no response from N)
16.             add N to Blacklist R;
17.         else
18.             K.nodeValidate ( N );
          }
        }
    }
```

Figure 4.24 Pseudocode of nodeValidation( )

As shown in Figure 4.24, S is the node residing in the target network. N is a new node trying to send advertisement requests to S. The function takes the advertisement request from N as a parameter. Line 1 looks into S's blacklist R which is shared with all of the other nodes in this piconet and checks the service type requested by N when a random node S in a piconet/scatternet receives service request in advertisement from node N. If line 1 returns false, line 2 will ignore this request and stop responding to node N. Otherwise, line 4 will establish connection from S to N by responding to node N with connection request. From line 5 to line 6, S and N will keep communicating with each other

till pre_timer1 runs out. Then the connection will be terminated in line 7. Line 8 will check whether the service requested by node N has accomplished. If the service is done, then the priority list L will be updated and node N's priority will drop down. Otherwise, the least busy neighbor of S will be assigned to K in line 11 and node K will prepare to take turns to communicate with node N and continue the services by calling the function K.init() in line 12. Meanwhile Node S will notify node N to send the advertisement directly to node N and switch to connect with node K in line 13. After pre_timer2 runs out, if node K does not receive any response from node N, N will be blacklisted. Otherwise, if node K receives connection request from node N and node K will recursively call this validation function.

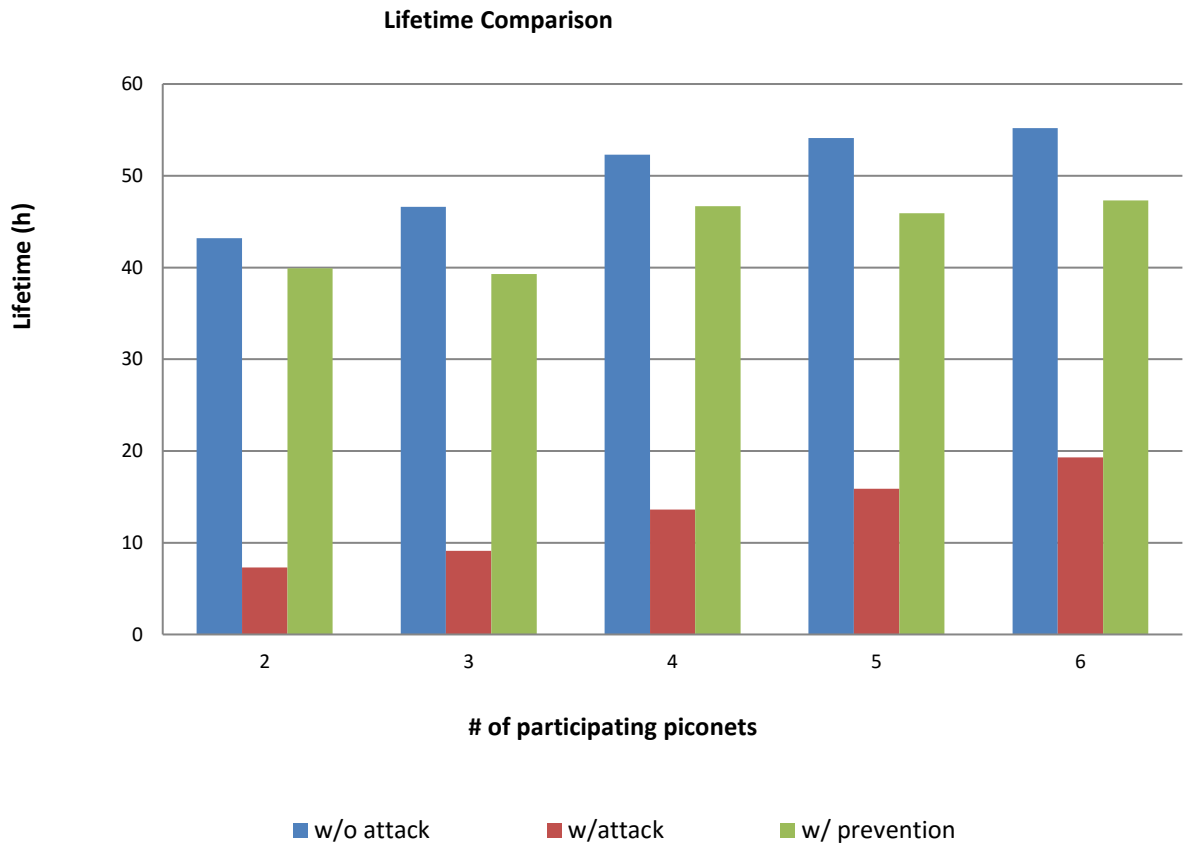


Figure 4.25 Lifetime Comparison

This validation procedure will be executed recursively. Pre_timer1 is the timer used for switch the node which will communicate with the request node N in round robin manner. Pre_timer2 starts to work once the establishment of the connection between nodes is completed.

4.4.3 Evaluation

In our experiment, we adopt Broadcom chipset 2070X which supports the latest Bluetooth core specification 4.2. We choose one development board as the attacker to the mesh network using 2070X chipset. The attacker will launch different attacks varying from level 1 to level 3, and targeting at different victims in the mesh network until the lifetime of the network ends.

We perform a set of experiments in order to characterize our scatternet formation, routing approach and prevention mechanism. In the experiment, we vary the number of piconets from 2 to 6. We randomly pick a pair of nodes from difference piconets and use them as source-destination pair nodes to route the messages. During this repeated procedure, we collect the data such as lifetime and delay to evaluate our algorithm.

Figure 4.25 below show the differences between whether our prevention mechanism is adopted or not. In Figure 4.25, the w/o attack bar represents the scenario of the scatternet not under battery exhaustion attack while the red bar represents the scatternet under attack and green bar represent the prevention mechanism against attack. Figure 4.26 compares the lifetime of the BLE scattternets. Lifetime of network means the defined time from the moment when a route is first established till the moment when there is no available route from the source node to destination node. Lifetime indicates that one

or more of the path nodes is depleted of energy. If a scatternet is alive, it should at least meet the below requirements as described previously:

1) All nodes which transmit/receive messages must maintain battery levels at or above the threshold.

2) At least one route exists for a source-destination pair such that all intermediate path nodes have sufficient power to guarantee the successful delivery of messages.

We have presented an efficient approach for BLE based sensor network to prevent against battery exhaustion attack. This approach significantly identifies the attacker and achieves longer lifetime. In future work, we will introduce signatures of link layer or lower layer into our mechanism to detect potential spoof attack.

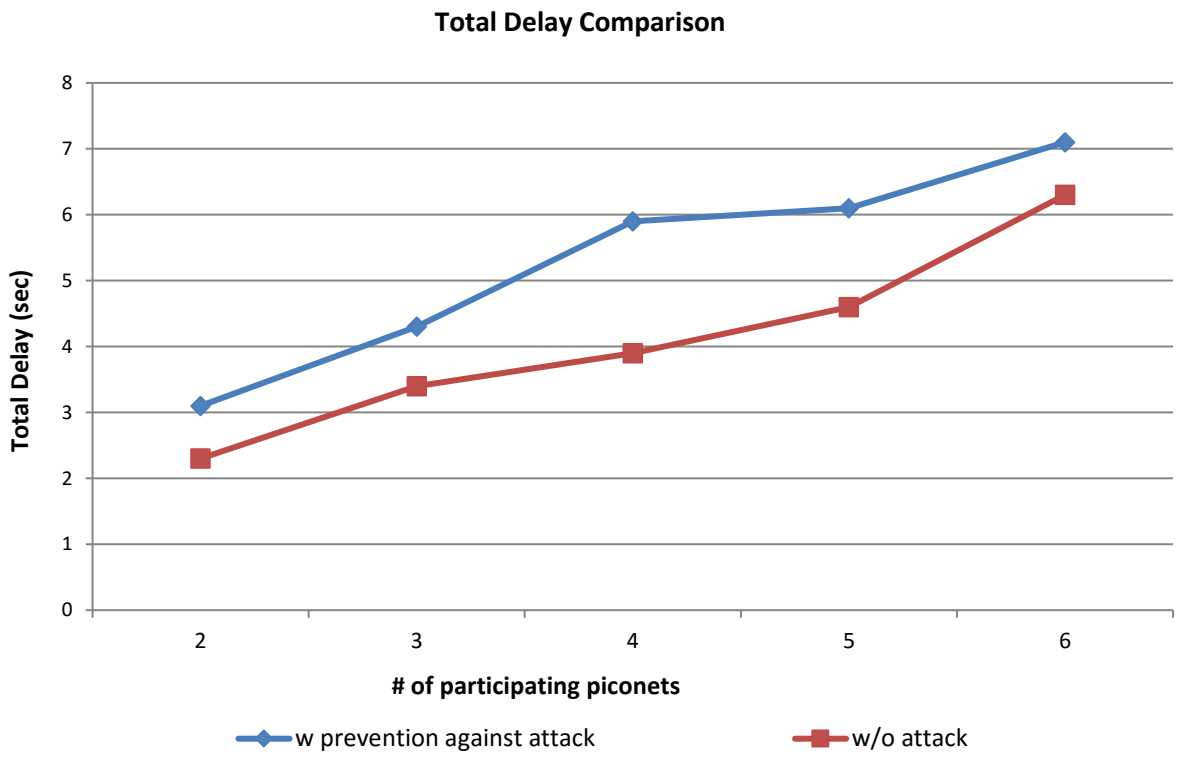


Figure 4.26 Total Delay Comparison

4.5 Conclusions

In this chapter, we study the IoT network established by BLE protocol. Since the BLE scatternet is new and there is no existing approach addressing this problem, we construct a BLE based sensor network in order to do the research on this. In the meantime, since there exist no simulation tool for us to use, we implement our approaches on real hardware. During this procedure, we firstly introduce a scatternet formation protocol which is used to establish a dynamic, distributed scatternet supporting multi-hop scenarios. Then we propose a reactive routing algorithm enabling the BLE nodes in the networks to communicate with each other. Our protocol does not incur large delay and achieves better resource utilization by shortening the route path. Next, due to the fact that BLE devices are mostly driven by battery, we improve our routing algorithm by introducing the battery power as a main factor to decide the routings. This approach significantly extends the lifetime, as well as identify and prevent battery draining attacks. With our detection approach of defend against battery draining attack, the network sustains a much longer time.

Chapter 5

Conclusions and Future Work

In this chapter, we first conclude our contributions in defend against intrusion attacks to IoT devices and BLE based IoT sensor networks. Then we identify and present a few open areas which have not been touched or completely solved in this dissertation.

5.1 Conclusions

The primary goal of our research is to enhance security for small wireless devices that operate in IoT networks and for the BLE based sensor networks. In this dissertation, we have systematically explored the host-based as well as network based intrusion detection, particularly for the BLE devices and BLE based IoT sensor networks.

First, we explore how malware, as the most prominent security threat to computing devices, attack the target devices. Then we study the host-based intrusion detection techniques. Specifically, we have studied how to leverage control flow integrity to defend against malicious attacks. Based on the IoT devices have a high requirement for speed and cost. We proposed a hardware-based approach following the COTS design strategy and running at runtime.

Second, we study the network based intrusion. Due to the fact that so far there is no simulation tools or platform supporting BLE protocol, we construct a BLE sensor network. Particularly based on BLE features and mobility of BLE networks, we propose an on-demand scatternet formation and multi-hop routing protocol. In order to make our

approach to be more compatible with real scenarios, we take the battery lifetime into consideration to improve our routing algorithm.

Finally, we study the security threat to this BLE based IoT networks. We particularly investigate battery draining attack, one of the most common attack, to BLE networks. We present an approach to identify the malicious nodes repeatedly requesting service from BLE networks and blacklist these nodes. Our approach effectively prevents this attack and extend the networks lifetime.

5.2 Future Research Directions

In the future, this research could be expanded to investigate and further develop a robust BLE based IoT networks. In this procedures, more effort needs to be made to secure IoT-related data to ensure the privacy of consumers and the functionality of businesses and corporations.

We identify a few open research directions listed below. The directions extend our research in this dissertation.

- Our work about "CFI checking for Intrusion Detection via a Real-Time Debug Interface" is just a first step to defend against intrusion attacks aiming to alter control flow. The future work can explore how the characterization used to generate the branch destination table, as well as the mechanism to ensure quick validation of branches can be improved to further achieve better latency at runtime.
- In the work "formation and routing approach for BLE sensor network", we only take the hops and battery life as main factors in making formation and routing decisions. In the future, one research direction is to take the service priority and workload in

to consideration to select the master of the each piconet. Another direction is to explore how to enable additional features to NS3 platform to support BLE protocol in order to achieve large scale simulations.

- In the work “security of BLE sensor networks”, we only discuss how to validate the unique Bluetooth ID to defend against the common battery exhaustion attacks. In the future, more information of devices, such as link layer signature and lower layer signature can be explored. Another limitation is so far the BLE based IoT networks is still in development and not so common, the potential security vulnerability can be further studied in the future. Future work beyond this dissertation effort include large scale simulations in which the BLE based sensor network can share the validation information effectively and timely.

Bibliography

[1] U. Ahmed and A. Masood. Host based intrusion detection using RBF neural networks. In ICET International Conference on Emerging Technologies, pages 48 – 51, 2009.

[2] D. Arora, S. Ravi, A. Raghunathan, and N.K. Jha. Secure embedded processing through hardware-assisted run-time monitoring. In Proceedings of the conference on Design, Automation and Test in Europe- Volume 1, pages 178–183. IEEE Computer Society, 2005.

[3] D. Arora, S. Ravi, A. Raghunathan, and N.K. Jha. Hardware-assisted run-time monitoring for secure program execution on embedded processors. IEEE Transactions on VLSI Systems, 14(12), December 2006.

[4] M. Eby, J. Werner, G. Karsai, and A. Ledeczi. Integrating security modeling into embedded system design. In Engineering of Computer- Based Systems, 2007. ECBS '07. 14th Annual IEEE International Conference and Workshops on the, pages 221 – 228, 2007.

[5] H.Shacham. The geometry of innocent esh on the bone: Return-intolibc without function calls (on the x86). ACM Conf. on Computer and Communications Security, CCS., Jul 2007.

[6] IEEE- Industry Standards and Technology Organization (IEEE-ISTO). The Nexus 5001 Forum Standard for a Global Embedded Processor Debug Interface, 2003.

[7] A.-R. Sadeghi. L. Davi, P. Koeberl. Hardware-assisted fine-grained control-flow integrity: Towards efficient protection of embedded systems against software

exploitation. In 51st Design Automation Conference (DAC) - Special Session: Trusted Mobile Embedded Computing, Jun 2014.

[8] V. Pappas, M. Polychronaks, and A. Keromytis. Transparent rop exploit mitigation using indirect branch tracing. In USENIX conference on Security, SSYM, 2013.

[9] M. Rahmatian, H. Kooti, I. G. Harris, and E. Bozorgzadeh. Hardwareassisted detection of malicious software in embedded systems. IEEE Embedded Systems Letters, 4(4), 2012.

[10] R. Roemer, E. Buchanan, H. Shacham, and S. Savage. Return-oriented programming: Systems, languages, and applications. ACM Trans. Inf. Syst. Secur., 15(1), March 2012.

[11] X. Wang and R. Karri. Numchecker: Detecting kernel control-flow modifying rootkits by using hardware performance counters. In 50th Design Automation Conference, June 2013.

[12] H. Chen Y. Xia, Y. Liu and B.Zang. Cfimon: Detecting violation of control flow integrity using performance counters. In 42nd IEEE International Conference on Dependable Systems and Networks, June 2012.

[13] T. Zhang, X. Zhuang, S. Pande, and W. Lee. Anomalous path detection with hardware support. In CASES '05: Proceedings of the 2005 international conference on Compilers, architectures and synthesis for embedded systems, pages 43–54, New York, NY, USA, 2005. ACM.

[14] T. Zhang, X. Zhuang, S. Pande, and W. Lee. Anomalous path detection with hardware support. In Proceedings of the 2005 International Conference on Compilers, Architectures and Synthesis for Embedded Systems, 2005.

[15] Bluetooth Core Specification, Version 4.0, SIG, June 2010.

[16] "The ns-3 Network Simulator," Project Homepage. [Online]. Available: http://www.nsnam.org [Accessed: July, 2014]

[17] D.B. Johnson and D.A. Maltz, "Dynamic source routing in ad-hoc wireless networks", In Imielinski and Korth, editors, Mobile Computing, pp. 153-181. Kluwer Academic Publishers, 1996.

[18] C. Aichele, M. Lindner, A. Nuemann and S. Wunderlich, "Better Approach to Mobile Ad-hoc Networking(BATMAN)," ietf, 2008.

[19] S. J. Lee, W. Su, J. Hsu, M. Gerla and R. Bagrodia, "A performance Comparison of Ad Hoc Wireless Multicacst Protocols", Proc. IEEE Conference Computer Communication, pp. 565-574, 2000.

[20] C. Petrioli, S. Basagni, "Degree-constratined multihop scatternet formation for bluetooth networks, in IEEE GLOBECOM, 2002.

[21] C. Petrioli, S. Basagni and I. Chlamtac, "Bluemesh: Degree-constratined multihop scatternet formation for bluetooth networks", ACM/Kluwer/SPIE Mobile Networks and Applications 9(1) (2004) 33- 47 (Special Issue on Advances in Research of Wireless Personal Area Networking and Bluetooth Enabled Networks)

[22] G. Zaruba, S. Basagni and I. Chlamtac, "Bluetrees scatternet formation to enable bluetooth-based ad hoc networks," in Communications, 2001. ICC 2001. IEEE International Conference on, vol. 1, 2001, pp. 273- 277.

[23] F. Cuomo, G. Di Bacco and T. Melodia, "Shaper: a self-healing algorithm producing multi-hop bluetooth scatternets," in IEEE GLOBECOM, 2003.

[24] "Statista: the statistics portal", Project Homepage [Online], Available: www.statista.com [Accessed: July 2014]

[25] R. Whitaker, L. Hodge and I. Chlamtac, "Bluetooth scatternet formation: a survey", Ad Hoc Networks, vol. 3, no. 4, pp. 403-451, 2005.

[26] M. Methfessel, S. Peter and S. Lange, "Bluetooth scatternet tress formation for wireless sensor networks", International conference on mobile Ad-hoc and sensor systems, 2011.

[27] C. Perkins and P. Bhagwat, "Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers", Comp. Comm. Rev., Oct. 1994.

[28] S. Murthy and J.J. Garcia-Luna-Aceves, "An Efficient Routing Protocol for Wireless Networks", ACM Mobile Networks and App. J., Special Issue on Routing in Mobile Communication Networks, 1996.

[29] M. Joa-Ng and I. Lu, "A Peer-to-Peer zone-based two-level link state routing for mobile Ad Hoc Networks", IEEE Journal on Selected Areas in Communications, Special Issue on Ad-Hoc Networks, Aug. 1999

[30] C. Perkins, E. Royer and S. Das, "Ad Hoc On-demand Distance Vector Routing", Oct, 99.

[31] Bluetooth SIG, "Bluetooth Core Specification, Version 4.2, Dec 2014.

[32] IPv6 over BLUETOOTH(R) Low Energy, IETF Draft, June 2015.

[33] S. Jankowski, "The Sectors where the Internet of Things really matters", Harvard Busines Review, October, 2014, [Online]. Available: https://hbr.org/2014/10/the-sectors-where-the-internet-of-things-reallymatters/ [Accessed: July, 2014]

[34] F. Stajano and R. Anderson, “The resurrecting duckling: Security issues for ad hoc wireless networks,” in Proceedings of the 7th International Workshop on Security Protocols, Lecture Notes in Computer Science volume 1796, pp. 172-194, April 1999.

[35] Bluetooth SIG, “Bluetooth Core Specification, Version 4.1, Dec 2013.

[36] “Control-flow Checking for Intrusion Detection via a Real-Time Debug Interface.” Z. Guo, R.Bhakta and I.G. Harris. Smart Computing Workshops, 2014 International Conference on IEEE.

[37] “An on-demand scatternet formation and multi-hop routing protocol for BLE-based wireless sensor networks.” Z. Guo, I.G. Harris, L. Tsaur and X. Chen. Wireless Communications and Networking Conference (WCNC), 2015 IEEE.

[38] “A Residual Battery-Aware Routing Algorithm Based on DSR for BLE Sensor Networks.” Z. Guo, I. G. Harris, C. Harris, Y. Jiang and L. Tsaur. Wireless Telecommunications Symposium (WTS), 2016 IEEE.

[39] T. Martin, M. Hsiao, D. Ha, J. Krishnaswami, “Denial-of-Service Attacks on Battery-powered Mobile Computers”, Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communication, 2004.

[40] G. A. Jacoby and N. J. Davis, "Battery-based intrusion detection," in Global Telecommunications Conference, 2004. GLOBECOM '04. IEEE, pp. 2250- 2255 Vol.4, 2004.

[41] D.C. Nash, T. Martin and D. Ha and M. Hsiao “Towards an intrusion detection system for battery exhaustion attacks on mobile computing devices” Pervasive Computing and Communications Workshops, 2005.

[42] C. Aichele, M. Lindner, A. Nuemann and S. Wunderlich, “Better Approach to Mobile Ad-hoc Networking(BATMAN),” ietf, 2008.

[43] R. Racic, D. Ma, et al., "Exploiting MMS vulnerabilities to stealthily exhaust mobile phone's battery," in the 15th Annual USENIX Security Symposium Vancouver, BC, 2006.

[44] T. K. Buennemeyer, F. Munshi, et al., "Battery-sensing intrusion protection for wireless handheld computers using a dynamic threshold calculation algorithm for attack detection," in the 40th Annual Hawaii International Conference on System Sciences (HICSS-40) Waikoloa, Hawaii, 2007.

[45] M. Brownfield, G. Yatharth, et al., "Wireless sensor network denial of sleep attack," in the Sixth Annual IEEE Systems, Man and Cybernetics (SMC) Information Assurance Workshop. West Point, NY, 2005.

[46] J. Decuir, “Introducing Bluetooth Smart: A look at both classic and new technology”, IEEE Consumer Electronics Magazine, 2014.