SeamBlue: Seamless Bluetooth Low Energy Connection Migration for Unmodified IoT Devices

Syed Rafiul Hussain[1], Shagufta Mehnaz[1], Shahriar Nirjon[2], Elisa Bertino[1]
[1]Purdue University, [2] UNC Chapel Hill

hussain1@purdue.edu, smehnaz@purdue.edu, nirjon@cs.unc.edu, bertino@purdue.edu

Abstract

At present, Bluetooth Low Energy (BLE) is dominantly used in commercially available Internet of Things (IoT) devices – such as smart watches, fitness trackers, and smart appliances. Compared to classic Bluetooth, BLE has been simplified in many ways that include its connection establishment, data exchange, and encryption processes. Unfortunately, this simplification comes at a cost. For example, only a star topology is supported in BLE environments and a peripheral (an IoT device) can communicate with only one gateway (e.g. a smartphone, or a BLE hub) at a set time. When a peripheral goes out of range, it loses connectivity to a gateway, and cannot connect and seamlessly communicate with another gateway without user interventions. In other words, BLE connections are not automatically migrated or handed-off to another gateway. In this paper, we propose *SeamBlue*, which brings seamless connectivity to BLE-capable mobile IoT devices in an environment that consists of a network of gateways. Our framework ensures that unmodified, commercial off-the-shelf BLE devices seamlessly and securely connect to a nearby gateway without any user intervention.

1 Introduction

The Internet of Things (IoT) has entered the commercial market much faster than expected. The IoT industries predict that the total number of 'smart things' will be more than 30 billion [9] by the year 2020 – which will outnumber the total number of smartphones. IoT technology is already being adopted in many places such as factories, airports, offices, homes, hospitals, and schools, and is being used in applications such as asset tracking, health monitoring, predictive maintenance, environmental monitoring, energy metering, and elder care. In a typical scenario, an IoT device connects to a gateway (e.g., a smartphone or a smart hub) over a low-power wireless network, and the gateway enables its access to the Internet. Because the connection process between an IoT device and a gateway requires the active engagement of a user, *seamless connectivity of mobile IoT devices in a network of gateways* is still not happening. Ideally, an IoT device should be able to seamlessly communicate with a nearby gateway, without requiring an end-user to enter pins and passwords every time it moves near a different gateway in the same trusted network environment.

There are a number of wireless protocols such as Bluetooth LE (BLE) [3], ZigBee [18], and NFC [12], that have been used in different IoT communication scenarios. Among these, BLE has become the most popular choice because of its simplicity, openness, and a promised battery life of multiple years. The BLE protocol allows multiple devices ('peripherals') to attach themselves to a single gateway (the 'central'), but it restricts the mobility of the peripherals outside and into the range of a gateway. Carrying the gateway along with a mobile IoT device seems like an option, but it is not always feasible as it causes disconnections of other IoT devices that are either static or moving in a different direction from the gateway. For instance, if a personal smartphone is used as a gateway for the IoT devices deployed for a home automation system, BLE-enabled IoT devices may get disconnected when the smartphone is taken outside of the home. Also, patients wearing BLE enabled IoT devices may move inside and outside of the hospitals for which simple smartphones may not be used as a BLE gateway. Furthermore, IoT devices and gateways deployed in battlefields and agricultural farms can be mobile for which continuous connectivity through smartphones may not be a viable solution.

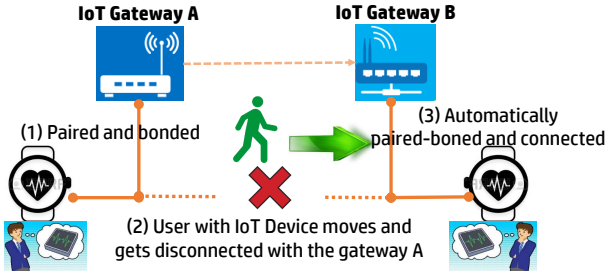


Figure 1: Seamless BLE connectivity architecture.

In order to ensure continuous BLE connectivity [3], Zachariah et al. [38] proposed an architecture where an IoT device may connect to multiple gateways located at different places. However, establishing a distinct connection with every gateway requires a peripheral to reset and broadcast advertising signals separately for all the gateways. This behavior is observed in many BLE devices including Moto 360 and Samsung Gear watches. Some of the Android Wear watches are so aligned with their proprietary smartphone applications that these devices do not even allow themselves to pair with non-proprietary smartphones. In fact, the scenario is so restricted in its functionality that even the model of a smartphone matters. For example, the latest Gear devices only

pair with a Samsung S4 and above. In such scenarios, the possibility of a Utopian environment where smart devices seamlessly communicate with each other seems a far reality.

Even if connections with multiple gateways are made possible by hacking [29], it comes at the cost of disconnecting the device from its previous gateway and then connecting to a new one. This incurs significant CPU, memory, energy, and bandwidth overhead in resource constrained IoT devices as even a single connection establishment requires advertisements, discovery, pairing and bonding [37], and several mutual agreements in different layers of BLE protocol stack. Consequently, connection establishment with multiple devices is neither a time efficient nor a cost effective process. In addition, the process requires repeated manual interventions that disrupt the ongoing communication between a device and a remote service. Because of these practical issues, we argue that an IoT device should be able to seamlessly communicate with different gateways [38] without requiring to create a separate connection with each of them.

Our vision of a seamless BLE migration is illustrated in Figure 1, where a user at first connects (pairs) his fitness tracker to gateway *A* like he does for any BLE device. When he moves to gateway *B*, connection states are automatically migrated from gateway *A* to gateway *B* over a different communication channel, without interrupting ongoing communications between the device and any remote service it is talking to. Finally, when he enters into the range of *B*, the fitness tracker is completely handed-off to gateway *B*, without requiring the user to manually pairing with it. While this seems similar to hand-offs [32, 30, 36] in cellular or WiFi networks, a major distinction between BLE migration and a cellular/WiFi connection migration is that in case of BLE, we are constrained by the billions of already deployed IoT devices and many other legacy devices that are running Bluetooth 4.0, for which we cannot change their BLE implementation. This practical constraint makes it difficult even to detect the presence of a device at the time it is in the connected state with a central. In addition to this, migrating a connection requires transferring a set of state variables between gateways that define the state of a BLE connection. Finding the set of state variables by browsing a large code base is itself a time-consuming and error-prone process. Finally, selecting the next gateway among the available ones and then securely transferring the connection states poses further challenges to seamless BLE migration.

In this paper, we propose *SeamBlue*, which addresses these challenges and enables seamless BLE connection migration for mobile IoT devices in a network of static or mobile BLE gateways. Several salient features combined together make *SeamBlue* unique. First, we develop a systematic approach based on static program analysis which automatically finds a set of variables and objects in the BLE code base that define a connection state. Second, we propose two modes of connection state extraction: *full stack cloning* and *partial stack cloning* in order to support connection migration for a wide range of IoT devices. Third, we leverage existing approaches to user movement prediction [34], and propose a mechanism to select the best candidate gateway for a connection migration. Fourth, while transferring connection from one gateway to another we consider both trusted and untrusted gateways and ensure secure connection migration. Fifth, we have developed a testbed that consists of unmodified, BLE-capable IoT devices (e.g., Android smartphones, and tablets) and BLE gateways (e.g., customized smartphones acting as central). We perform an in-depth evaluation of *SeamBlue* in this testbed to quantify its effectiveness as well as its overhead. In summary, the contributions of this paper are the following:

- We propose a framework that ensures seamless communication between an unmodified, BLE-enabled mobile IoT device and a remote service in a network of static or mobile BLE gateway environment, without requiring pairing-bonding and connections to individual gateways.
- We develop a systematic approach based on static program analysis to identify the state variables in the BLE code base that are required for transferring pairing-bonding and connection information from one gateway to another gateway.
- We propose two approaches – *full stack cloning* and *partial stack cloning* for capturing a snapshot of connection states at the current gateway and then updating them at the next gateway during BLE connection migration.
- We propose a gateway selection mechanism for transferring the connection state to the most suitable gateway when an IoT device requires to migrate its connection and there are multiple gateways in its range.

2 Usage Scenarios

We briefly describe some of the use cases of seamless BLE connection migration in this section.

- *In Hospitals:* Patients wearing BLE devices in hospitals can be localized and tracked, and their heart rate and other physiological signals can be monitored using a network of gateways deployed at different locations in the hospital. Even if the patient moves, these devices will provide continuous monitoring and uninterrupted services by connecting to nearby gateways.
- *In Airports:* Many airports [8] use BLE enabled tracking devices to monitor the location and movement of passengers and airport equipment. Upon arrival at the airport, passengers (and baggage) which are equipped with BLE beacons can voluntarily report their location and status to the deployed gateways from anywhere within the airport, and in return, they get personalized services and notifications.
- *In Theme Parks:* With the help of a BLE-enabled wristband worn by children and static gateways deployed at different locations inside a theme park, parents can monitor and locate their missing kids via their mobile phone. The IoT wristband concept has already been implemented and tested with much success at Disney World in Orlando [7], Florida. Disneys MagicBand is a customizable wristband that functions as a passport for just about everything in the park. These bands can serve as a digital entrance ticket for guests or even store credit card information to facilitate transactions.

3 BLE Background

3.1 Roles of BLE Device

A BLE device assumes either a *peripheral* or a *central* role. A peripheral, typically an IoT device, such as a heart rate monitor, a blood pressure monitor, a smart lock, or a smart watch, has limited capabilities and contains advertisement information. A central device, such as an access point, a personal computer, or a smartphone, scans for BLE advertisements, receives an advertisement, and initiates a connection.

3.2 BLE Protocol Stack

Similar to classic Bluetooth [2], the BLE protocol stack [4] is composed of two main parts: a *controller* and a *host* as shown in Figure 2.

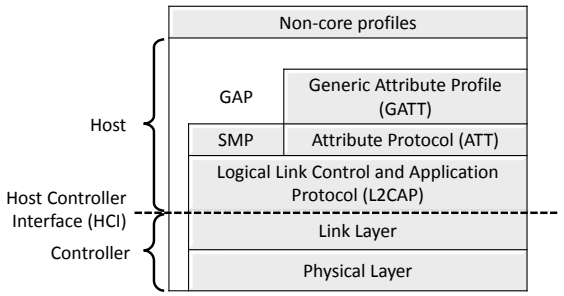


Figure 2: The Bluetooth LE Protocol Stack.

Physical Layer: BLE operates in the 2.4 GHz Industrial Scientific Medical (ISM) band and defines 40 Radio Frequency (RF) channels with 2 MHz channel spacing. There are two types of BLE RF channels: (1) three advertising channels used for device discovery, connection establishment and broadcast transmission, and (2) thirty-seven data channels used for bidirectional communication between connected devices. In order to avoid interference, an adaptive frequency hopping pattern consisting of 37 frequencies is used for data channels.

Link Layer: BLE defines two device roles at the Link Layer for a connection: the *master* and the *slave*. Once a connection between a master and a slave is created, the physical channel is divided into non-overlapping time units called connection events. In order to allow bit error detection, all data units include a 24-bit Cyclic Redundancy Check (CRCInit) code. For a new connection event, master and slave use a new data channel frequency, which is computed using the frequency hopping algorithm. `Access Address (AA)`, embedded in a Link Layer packet is used to identify communications on a physical link, and to exclude or ignore packets on different physical links that are using the same physical channels in physical proximity.

L2CAP: It works as a logical link layer and multiplexes the data of higher layers on top of a Link Layer connection.

ATT: The ATT defines the communication between two devices playing the roles of server and client, respectively. The server maintains a set of attributes. An attribute is a data structure that stores the information managed by the GATT.

GATT: A framework defined by GATT uses the ATT for the discovery of services that includes characteristics. A characteristic is a set of data which includes a value and a set of properties. The data related to services and characteristics are stored in attributes. For example, a server that runs a heart rate monitoring service may account with a heart rate characteristic that uses an attribute for describing the sensor, another attribute for storing heart rate measurement values and a further attribute for specifying the measurement units.

GAP and Application Profiles: GAP specifies device roles, modes, and procedures for the discovery of devices and services, the management of connection establishment and security. A device may support various roles, but only one role can be adopted at a given time. Application profiles specify general behaviors that Bluetooth-enabled devices use to communicate with other Bluetooth devices. For example, a heart-rate monitor is able to send its sensor values to a gateway only if the corresponding gateway implements a heart-rate profile.

3.3 Modes of Communication

Two modes of communication are available: *broadcast* and *connected* modes. The broadcast mode enables a peripheral to send data to any other device listening for transmissions. If two devices need to exchange data they can use the *connected* mode. In this mode, a peripheral device broadcasts its presence by sending advertisement packets. The central can initiate a connection following a received broadcast. Once a connection is established the devices can exchange data.

3.4 BLE Security and Privacy

Pairing: In *connected* mode, if two devices want to exchange data securely, they perform a *pairing* process where as a first step, the parties involved in the communication exchange their identity information to set up the trust and then establish the encryption keys for future data exchange. The Security Manager Protocol (SMP) used for the pairing procedure results in the following keys that are shared between the peripheral and the central.

- *Identity Resolving key (IRK)*: 128-bit key used to generate and resolve a random address.
- *Connection Signature Resolving Key (CSRK)*: 128-bit key used to sign data and verify signatures on the receiving device.
- *Long Term Key (LTK)*: 128-bit key used to generate the session key for an encrypted connection.
- *Encrypted Diversifier (EDIV)*: 16-bit stored value used to identify the LTK. A new EDIV is generated each time a new LTK is distributed.
- *Random Number (RAND)*: 64-bit stored value used to identify the LTK. A new RAND is generated each time a new LTK is distributed.

Bonding: Bonding is the process of storing the keys created during pairing for use in subsequent connections in order to form a trusted device pair.

Privacy: BLE can use *Random Device Addressing* to help increase the privacy [24, 22] of connections and prevent 'tracking' based on the assumption that eavesdropping did not occur during the pairing process.

If the advertising device is previously discovered and has returned to an advertising state, the device must be identifiable by trusted devices in future connections without going through discovery procedure again. The *IRK* stored in the trusted device is used to identify the advertiser as a trusted device.

4 Challenges to BLE Connection Migration

Although BLE connection migration for IoT devices seems similar to that of traditional networks, e.g., WiFi or TCP connection migration, it poses some unique challenges specific to IoT.

Unmodified IoT Devices: Since billions of IoT devices are already in use, it is not a feasible proposal to demand changes in their BLE implementation. Hence, we are constrained to only change the BLE implementation of the gateways for a seamless connection migration.

Identifying State Variables: We need to identify the set of variables that uniquely define a state of a BLE connection. Manually browsing a large code base (up to 100K lines of code) to find state variables is an impractical, error-prone, and time-consuming process. Hence, automatically finding state variables for connection is essential.

Gateway Selection for Connection Transfer: While transferring the pairing-bonding information, not all the gateways in a network should be a receiver of this information. For example, if we require syncing all the gateways for every pairing between a gateway and an IoT device, it would require a substantial amount of time and bandwidth and incur significant communication overhead. Hence, it is necessary that a subset of gateways are selected for sharing the pairing-bonding information.

Secure and Fast Connection Transfer: IoT gateways need to distribute the pairing-bonding and connection state information to the candidate gateways (or to a central authority) in a secure manner so that an adversary cannot obtain this information and impersonate a legitimate gateway. The connection transfer should be fast enough so that the services are not disrupted. In addition, if a gateway is not part of the trusted cluster of IoT devices, the current gateway may need to establish a trust relationship with that untrusted gateway[1] before seamless connection migration begins.

5 *SeamBlue* Overview

SeamBlue addresses the challenges mentioned in the previous section and enables seamless BLE connection migration for mobile IoT devices in a network of BLE gateways. This section provides an overview of *SeamBlue* by briefly describing the sequence of steps for connection migration, which comprises of identifying bonding/connection related state variables, establishing a connection with a gateway, transferring the bonding/connection information, and establishing a connection with a subsequent gateway.

[1]We use the terms 'untrusted gateways' and 'the gateways that do not belong to a group of already trusted gateways' interchangeably. Trusted gateways are of two types: (1) gateways sharing the same group key, (2) previously untrusted gateways whose public key certificates have been validated

5.1 Basic Workflow

SeamBlue ensures that an IoT device is always connected to a gateway, as long as it is within the range of any gateway. Figure 3 depicts the basic workflow of the connection migration process which we describe next.

1. In an off-line, one-time step, we analyze the BLE source code statically to identify the set of variables required for both pairing-bonding and connection information transfers. The *BLE Static Analysis (BSA)* module performs this static analysis on BLE source code and the identified set of variables (that are required for connection transfers) are stored in each of the gateways. This step is described in Section 6.1.
2. An IoT device advertises its information and a nearby gateway establishes a connection by pairing-bonding. We name the gateway to which the IoT device is currently connected the *current* gateway. The current gateway extracts a set of pairing-bonding information from the connection in order to share it with other nearby gateways. Section 6.2 describes this information extraction step.
3. The current gateway disseminates the pairing-bonding information to a set of gateways that are candidates for the subsequent gateway (the next gateway to which the IoT device may connect). This pairing-bonding information consists of both the bonded device's information as well as a subset of state variables. Section 6.3 describes this step in details.
4. Upon reception of this information, a set of candidate gateways add the IoT device as a bonded device, initializes a subset of state variables for bonding, but do not initiate a connection.
5. Perceiving the fact that the IoT device is moving out of its range, the current gateway selects a set of candidate gateways as the subsequent gateway (according to the movement of the IoT device). This step is described in Section 6.4.
6. The current gateway identifies the current state (or snapshot) of the connection and transfers the required state variables to the subsequent gateway so that the subsequent gateway can reconstruct the connection state with the same peripheral. Extraction of these variables is described in Section 6.2, and Section 6.5 describes how these transfers are done securely.
7. Upon reception of the connection state information, the subsequent gateway creates required objects related to connection, updates the connection state variables, and stores the connection information into gateway's non-volatile memory (NVRAM). As a result, the peripheral gets seamlessly connected to the subsequent gateway, without interrupting ongoing services.

5.2 BLE Stack Cloning

SeamBlue provides two modes for connection state extraction: *full stack cloning*, and *partial stack cloning*. Full stack cloning refers to cloning states of all the layers of Bluetooth stack starting from the application layer down to the link layer whereas partial stack cloning refers to the cloning

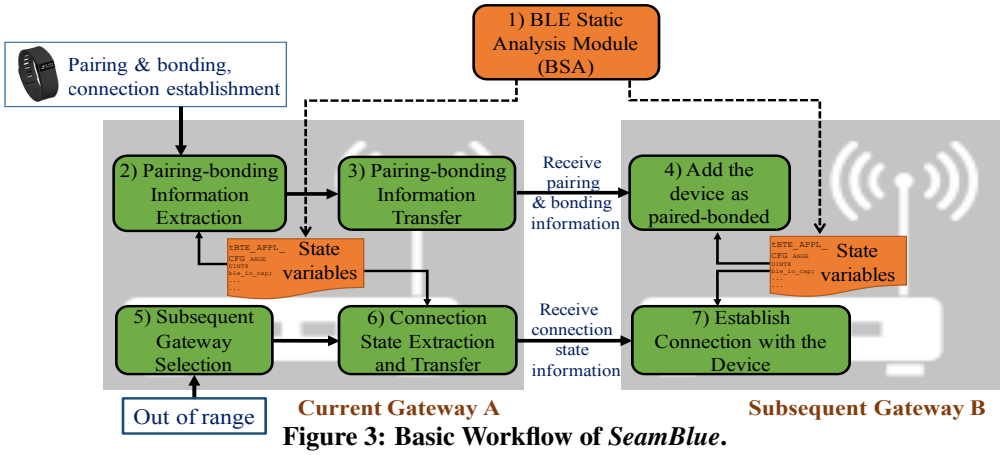


Figure 3: Basic Workflow of *SeamBlue*.

of Bluetooth stack starting from the application layer down to the L2CAP layer. Details of these modes are presented in Section 6.6.

5.3 Information Dissemination

SeamBlue supports two different strategies for the dissemination of both pairing-bonding and connection state information. The the initial gateway either (1) pushes this information to the cloud from where the candidate gateway(s) can sync this information periodically, or - (2) transfers this information directly to the selected gateway(s). Independent of the dissemination strategy, the current gateway transfers pairing-bonding and connection state information in a secure manner.

5.4 Threat model

We consider a strong threat model where only IoT gateways and IoT devices are parts of a trusted computing base. *SeamBlue* ensures confidentiality, integrity and authenticity of the control plane and the data plane messages. We consider both on-path and off-path attackers who have the capability of injecting unauthenticated packets, modifying legitimate packets, or sniffing end-to-end messages.

6 *SeamBlue* Design Details

This section describes how *SeamBlue* addresses the challenges in BLE connection migration by adding new functionality into the BLE stack of only the IoT gateways.

6.1 Identifying Connection State Information

Through our manual analysis we found that the open source BLE implementations [14, 1] are fairly complex. We also observed that the internal states of the BLE protocol consist of many different program variables and objects that are spread over different layers of BLE stack. This manual analysis costed more than 80 man-hours. However, the resultant set of connection state variables obtained by the manual analysis was not complete since a number of variables were left unrecognized. As a result, connection migration was not successful. Therefore, finding internal states (i.e., variables and objects that define the connection state) through manual analysis is not a viable solution. Furthermore, variants of a BLE implementation adopted for different IoT devices may have different sets of variables and objects for defining protocols' internal states. In order to address these challenges, we develop a systematic approach using static code analysis to find the minimal state information for BLE connection migration. Though static analysis [35, 21] have been used to find states for virtual machine (VM) migration in network function virtualization (NFV) [26], ours is different due to the IoT context and results in a more precise set of state variables. We define the variables of a program as state variables whose values are different in different connection states, get updated as the state changes, and whose scope lasts throughout the lifetime of a connection.

In order to identify the state variables, as a first step, we execute the following steps on the BLE source code:

- The BLE source code is first converted to intermediate representation (IR) using LLVM compiler [10].
- We build the control flow graph (CFG) [5] from the IR [11].
- We build a complete call graph consisting of both direct function calls and indirect function calls. Though it is possible to identify the direct function calls from CFG, identifying indirect function calls requires further analysis. Therefore, we use points-to analysis [28] to find the all possible targets of any indirect call in the program.

We partition the further analysis for extracting the state variables into two parts: *accept path* analysis, and *process path* analysis. The first part of the analysis considers the BLE source code portion from receiving a packet to accepting it. The second part of the analysis considers the BLE source code portion from accepting a packet to the end of processing the packet.

Accept Path Analysis: For a packet reception, the packet has to pass a set of checks (in other words, path conditions or constraints) and reach the program point where the packet is accepted using the `accept_payload()` call instruction. In this paper, we refer to these paths as accept paths. From our analysis of BLE source code, we have found that the variables and objects associated with the path constraints that are spread from the reception point of a packet to `accept_payload()` belong to the set of variables that define the connection state. The following are the steps of *accept path* analysis.

- We first identify the functions containing `accept_payload()` call instruction.

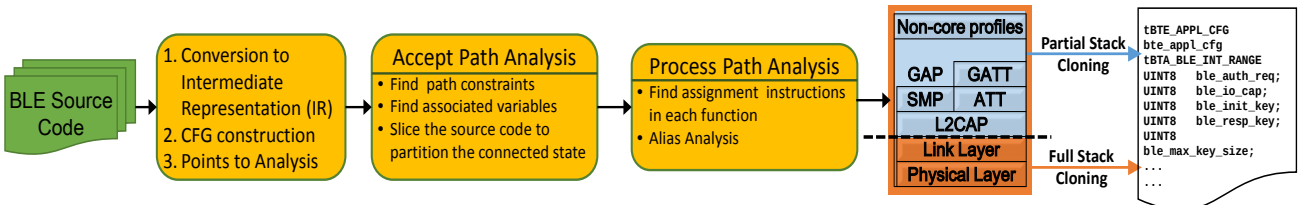


Figure 4: Static analysis of BLE source code for state variable identification.

- We use context and field sensitive inter-procedural points-to analysis [28] to build a bottom-up call graph to identify the functions and the corresponding callsites that are in call chain from `recv_packet()` to `accept_payload()`.
- We perform intra-procedural post-dominance frontier (PDF) [20] analysis to find the set of basic blocks on which the basic blocks containing `accept_payload()` instruction are control dependent. We also do the same intra-procedural PDF analysis for each callsite in the call chain computed from bottom-up call graph analysis. Thus we compute the inter-procedural path constraints that lead the program execution from `recv_packet()` to `accept_payload()`.
- We extract the path constraints by analyzing branch instructions located at the end of each control dependent basic block. The variables associated with those path constraints are considered as part of the connection state variables.

Process Path Analysis: After a packet gets accepted, the gateway starts processing the packet and takes actions (i.e., generates events) depending on the type and content of the packet. The gateway initializes and updates the program variables and objects specific to that connection. We need to identify those program variables whose liveness lasts throughout a BLE connection. We execute the following analysis to find those program variables:

- We use inter-procedural forward static program slicing technique to slice the code base into specific portions which process the packet. In order to do that, we first use the same context and field sensitive points-to analysis [28, 33] technique as used in the *accept path* analysis. Points-to analysis also helps in resolving the alias problems. From the results of the points-to analysis we create a call graph which includes all direct and indirect function calls starting from `accept_payload()` to the end of packet processing functions.
- We traverse each statement of the functions found in the call graph and identify the variables that are defined (i.e., assigned to some values) within that functions. This includes both local and global variables which are considered as possible connection state variables. Generally the scope of a local variable is only within the function unless it is an alias of a global variable and, therefore, it should not be considered as a connection state variable. However, if a local variable is an alias of a global variable, we consider that local variable as one of the state variables. We compare the points-to set of the local variables defined in a function with the points-to set of the global variables. If the points-to sets are identical, we infer that the local variable is an alias of the corresponding global variable.

Identifying the connection state variables is done offline and it is a one-time cost operation. *SeamBlue* does not need to compute the state variables every time it needs to migrate the connection. We present an excerpt of the resultant set of connection state variables for the BLE implementation in Table 1.

Items	Layers where used
Device Type	All layers
Device Address Type	All layers
Bluetooth device pseudo address	All layers
Long-Term Key (LTK)	SMP
Identity resolving key (IRK)	SMP
Connection Signature Resolving Key (CSRK)	SMP
EDiv	SMP
RAND	SMP
Access Address	Link Layer
Hop Interval	L2CAP, Link Layer
Hop Increment	L2CAP, Link Layer
CRCInit	Link Layer
Slave Latency	L2CAP, Link Layer
WinOffset	Link Layer
Channel Map	Link Layer
UUID	GAP, GATT
Characteristics Info	GAP, GATT

Table 1: An excerpt of the set of connection state variables of the BLE stack.

6.2 Extracting Pairing-Bonding/Connection State Variables

We instrument the BLE implementation so that we obtain the runtime values of the pairing-bonding/connection state variables for a connected IoT device (as shown in steps (2) and (6) in Figure 3). The current gateway stores the extracted information into memory and sends them to subsequent gateways. The runtime for this extraction module is distributed across different layers of BLE protocol stack. For runtime implementation, we add *SeamBlue* APIs so that the different layers can interact among themselves. Note that this instrumentation is performed only at the BLE gateways. Thus our proposed system does not modify the BLE implementation of IoT devices which allows the already deployed billions of IoT devices to integrate to the *SeamBlue* system without further modification.

6.3 Sharing Pairing-Bonding Information with Candidate Gateways

BLE central serves as a gateway and scans for peripheral devices so that it can connect with them and receive the desired GATT services. In order to ensure secure data transfer to the server through gateways, the current gateway initiates pairing and bonding procedures as shown with (1) in Figure 5. After creating connection through pairing and bond-

ing, the current gateway extracts the pairing-bonding related information for transfer to the possible subsequent gateways.

The current gateway then transfers the pairing-bonding related information to the gateways that are candidates for the subsequent gateway that the IoT device may connect next. In Figure 5, the gateways that are in the vicinity of gateway A are the gateways B, C, D, and E. Therefore, the candidate gateways are B, C, D, and E but not the gateways F, G, or H. As shown with (2) in Figure 5, gateway A sends the pairing-bonding information to the gateways B, C, D, and E.

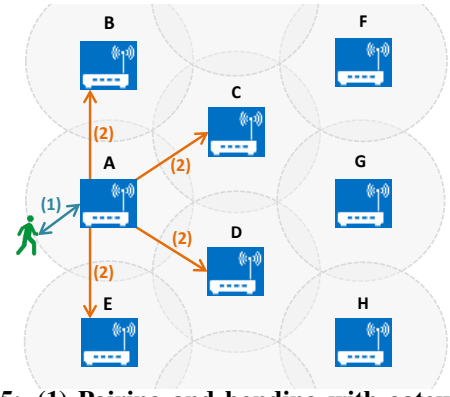


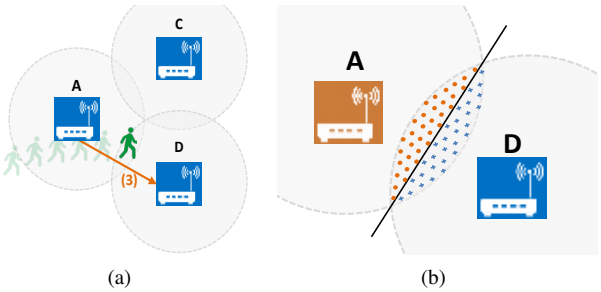
Figure 5: (1) Pairing and bonding with gateway A, (2) gateway A shares pairing-bonding information with the gateways B, C, D, and E.

Upon receiving the pairing-bonding information, the candidate gateways B, C, D, and E store this information mapped with the Bluetooth device address of that IoT device so that whenever that device needs service from these gateways, they do not have to execute the pairing-bonding procedures. Note that the candidate gateways do not initiate connection at this stage since they do not have connection state information.

6.4 Select Gateways for Connection Transfer

If an IoT device moves during or after connection establishment, the current gateway or the IoT service providing cloud system is able to estimate the device's moving direction [34]. *SeamBlue* uses this mechanism and examines a device's locations at recent timestamps to infer the moving direction. Location information of IoT devices can also be obtained using existing indoor and outdoor localization techniques [23]. By analyzing the movement direction and speed of the IoT devices, the current gateway or the service provider selects the subsequent gateway among the candidate gateways to whom the connection information will be transferred. As shown in Figure 6 (a), the current gateway A transfers connection information to the subsequent gateway D.

Ping-Pong Effect: An IoT device may move back and forth in a region shared by multiple gateways. As shown in Figure 6 (b), the gateways A and D share a common region which is partitioned using a line. The bronze markers denote the area where the signal strength (RSSI) of gateway A is greater than that of gateway D. Conversely, the blue markers denote the opposite case. According to *SeamBlue*, the current gateway A initiates a connection transfer as soon as the IoT device moves out of the A dominant area. However,



(a) (b)

Figure 6: (a) Subsequent gateway selection and connection transfer as the user with IoT device moves from gateway A to gateway D. (b) Gateways selection and ping-pong effect.

if an IoT device moves back and forth in a shared region, the BLE connection might also switch between the corresponding gateways. To reduce this effect, *SeamBlue* uses a motion prediction mechanism [34] that leverages statistical data of the IoT device's movements. If the device predominantly moves back and forth, *SeamBlue* gateways do not transfer the connection as soon as it goes beyond the half of the shared region. *SeamBlue* uses a delay tolerant approach to see if the device moves in the direction of the current gateway. If not, the current gateway transfers connection to the subsequent gateway.

6.5 Secure Connection Information Sharing

The gateway with which the peripheral is currently connected needs to distribute the bonding and connection information to candidate gateways and the subsequent gateway, respectively. Such sharing can be obtained either by pushing the bonding and connection information to the cloud from where all other gateways can fetch this information or by directly disseminating the bonding and connection information to the appropriate set of gateways. With respect to sharing of this information, the receiver gateways can be categorized into two groups: *trusted* and *untrusted* gateways.

Trusted Gateway: If the receiver gateways belong to the same cluster of gateways as the current gateway, the receiver gateways do not need to further authenticate themselves. They already share a secret group key with which they encrypt the data and then distribute the data in encrypted form securely. This secret group key can be shared between gateways through WiFi or 4G/LTE communication network and thus do not require any change in the existing bluetooth protocol. An example of such data transfer is following:

$$Enc\_D \leftarrow E_{K_{grp}}(D||nonce) \quad (1)$$

$$D||nonce \leftarrow D_{K_{grp}}(Enc\_D) \quad (2)$$

where D is the data to transfer, and K_{grp} is the symmetric group key. In Eqn. 1, the current gateway encrypts the data using K_{grp} and transfers Enc_D to the receiver gateway(s). A trusted receiver gateway has knowledge of K_{grp}, and thus decrypts Enc_D to D as shown in Eqn. 2.

Untrusted Gateway: If the subsequent gateway to which the BLE connection is going to be migrated is not already trusted, e.g., in the case of wide area networks (WANs), the current gateway needs to verify the public key certificate [16] of the subsequent gateway. Upon certificate validation, the

current gateway may establish a shared secret key with the receiver gateway using Diffie Hellman key exchange protocol [6] or just use the public key cryptography protocol [17] to share the bonding and connection information securely. Thus SeamBlue allows the BLE connection to migrate to a gateway that may be in a different LAN or WAN where the gateways are initially untrusted. However, Diffie-Hellman protocol might be time and energy inefficient for such scenarios as it requires multiple message communications to establish a secret key and to verify the certificates. In comparison with Diffie-Hellman, public key cryptography would result in less latency if the gateways know the public keys of their surrounding gateways. An example of data transfer with public key cryptography is following:

$$Enc\_D \leftarrow E_{PK_{rcv}}(D||nonce) \quad (3)$$

$$D||nonce \leftarrow D_{SK_{rcv}}(Enc\_D) \quad (4)$$

where D is the data to transfer, and PK_{rcv} is the public key of the receiver gateway. In Eqn. 3, the current gateway encrypts the data using PK_{rcv} and transfers Enc_D to the receiver gateway. The receiver gateway has knowledge of the corresponding secret key, SK_{rcv}, and thus decrypts Enc_D to D as shown in Eqn. 4. More efficient approaches, e.g., the pairing-free certificateless hybrid signcryption (pCL-HSC) [31] scheme, can also be used in the case of untrusted gateways which combines pCLSC-TKEM with a data encryption mechanism (DEM).

6.6 Cloning Connection Information to Subsequent Gateways

Upon reception of the connection information, the subsequent gateway does not initiate the pairing-bonding procedure, since the IoT device is already added as a bonded device into the subsequent gateway's NVRAM. The *SeamBlue* module running on the gateway updates the connection related parameters for communicating with the IoT device.

Since the subsequent gateway does not scan, discover, or create a new pairing and bonding with the IoT device, the BLE peripheral does not add the subsequent gateway as a bonded central device. Therefore, the IoT device does not replace the current gateway's device address with that of the next gateway in its memory. As a result, to send/receive packets to/from the peripheral, the subsequent gateway impersonates current gateway's device address. We instrument the BLE implementation on the gateway only so that the communication between the subsequent gateway and the IoT device is done with the current gateway's device address. Along with the device address, the subsequent gateway impersonates connection related information which are already shared with the IoT device. The connection related information is spread across both host and controller parts of the BLE stack (shown in Figure 2). Since some gateway devices, e.g., Android smartphones use proprietary Bluetooth device drivers, they do not allow one to change any variables located at the Link Layer in the controller part. Due to this limitation, we propose two approaches for cloning connection information to the subsequent gateway: *full* and *partial stack cloning*.

Full Stack Cloning: The *full stack cloning* approach allows the subsequent gateway to impersonate connection related information from the application layer to the link layer of the BLE stack. Therefore, the current gateway's `Access Address (AA)`, `connection interval`, `slave latency`, `channel map`, `CRCInit` values used in the link layer are impersonated by the next gateway. Note that channel maps can be updated to handle collisions. Since the current and the next gateways use the same `AA` for sending and receiving packets, the full stack cloning does not require any new connection request message from the next gateway to the peripheral device.

Partial Stack Cloning: In *partial stack cloning*, the subsequent gateway impersonates the connection related variables that are spread across the application layer to L2CAP layer of the BLE stack. The subsequent gateway adds the peripheral as a bonded device and impersonates the current gateway's device address. However, due to the proprietary nature of some Bluetooth device drivers, the subsequent gateway cannot change the values of `Access Address (AA)`, `connection interval`, `slave latency`, `channel map`, and `CRCInit` in the link layer of the BLE stack for completely impersonating the current gateway. We address this challenge by using an additional connection request from the subsequent gateway to the IoT device. In this procedure, the current gateway disconnects the connection with the IoT device and provides a control signal to the next gateway for sending connection request(s) to the peripheral. Since the subsequent gateway is already stored as the bonded device in IoT device's NVRAM, according to the BLE protocol, upon reception of the connection request the IoT device just updates the connection related parameters (i.e., `Access Address (AA)`, `connection interval`, `slave latency`, `channel map`, and `CRCInit`) for that bonded device. Hence, *partial stack cloning* requires an extra connection request for connection migration to the next gateway without modifying the IoT devices.

6.7 Implementation Notes

We briefly discuss some of the key implementation issues.

6.7.1 Static Analysis

We implemented the static analysis for finding state variables using LLVM 3.8 compiler infrastructure [10] by directly following the design from Section 6. The LLVM passes operate on the LLVM intermediate representation (IR), which is a low level strongly typed language-independent program representation tailored for static analyses and optimization purposes. The LLVM IR is generated from the C source code by clang. We used Bluedroid [1] for Android smartphones and BlueZ [14], an open source implementation, for many other BLE devices as the BLE protocol implementation.

6.7.2 Runtime Value Extraction

We instrument the Bluedroid (Android 4.2 and later) and BlueZ (Android 4.1 and before) for extracting the runtime values of the bonding and connection related state variables.

6.7.3 SeamBlue App

Our custom written *SeamBlue* application with Java (J2SE) implements the gateway selection algorithm and uses

OpenSSL libraries [15] for performing the cryptographic operations. The *SeamBlue* application running on the gateways use TCP connection to transfer bonding and connection related information and to exchange the control signals.

7 Evaluation

This section starts with the experimental setup followed by two sets of evaluations. First, the success rate of BLE connection migration is measured and the ping-pong effect is evaluated. Second, the overhead of *SeamBlue* is measured with the test bed we build.

7.1 Experimental Setup

7.1.1 Devices

We use five Nexus 5 phones as gateways (i.e., BLE centrals), one Nexus 6 phone and one Alcatel Onetouch tablet as IoT devices. The Nexus 5 phones have only the BLE central feature whereas the Nexus 6 and the Alcatel Onetouch tablet have both the BLE central and the BLE peripheral capabilities. The configuration of Nexus 5, Nexus 6, and Alcatel Onetouch Pixi tablet is summarized in Table 2.

Nexus 5	**Nexus 6**	**Alcatel Onetouch**
Qaud-core Krait CPU	Quad-core Krait 450 CPU	Quad-Core CPU
2.3 GHz	2.7 GHz	1.2 GHz
2 GB RAM	3 GB RAM	1 GB RAM
WiFi	WiFi	WiFi
4G LTE	4G LTE	4G LTE
BLE Central	BLE Central	BLE Central
	BLE Peripheral	BLE Peripheral
Android Lollipop OS	Android Marshmallow OS	Android Lollipop OS

Table 2: Configurations of Nexus 5 and Nexus 6 smartphones and Alcatel Onetouch Pixi tablet.

7.1.2 Testbed

We have built a testbed (as shown in Figure 7) by hanging the Nexus 5 smartphones as IoT gateways on the walls along the hallways in our department. In this testbed, any physical space is shared by at most two gateways. To evaluate the ping-pong effect, we make changes to the topology to arrange the gateways in a manner so that some areas fall within the ranges of more than two gateways.

7.1.3 Datasets and Applications

We use the *nRF Connect* application [13] downloaded from the Google Play store for BLE peripherals. For the gateways, we have developed a custom application. We use the heart rate monitoring service that periodically sends heart rate measurement in a single BLE packet of size 20-bytes every second. The heart rate monitoring service is one of the representative applications of SeamBlue where patients wearing BLE-enabled heart rate monitoring devices may move indoor or outdoor and may require to migrate the BLE connection from one gateway to another for continuous connectivity. Each data point reported in the experiment is obtained by taking the average of at least five runs.

7.2 BLE Connection Migration Success Rate

A connection gets migrated from the current gateway to subsequent gateways based on the location and movement direction of the user carrying the BLE peripheral enabled IoT devices/smartphones. We found every connection migration request successful when IoT devices are both static and moving at different speeds. Hence, the success rate we observe for *SeamBlue* in our testbed is 100%.

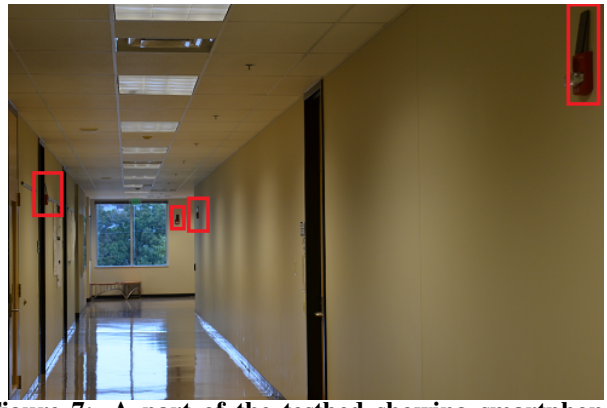


Figure 7: A part of the testbed showing smartphones (hanging near the fire alarms) that served as gateways.

7.2.1 Ping-Pong Effect

To evaluate the ping-pong effect we arrange the gateways in a formation so that the ranges of more than two gateways overlap. Users equipped with IoT devices pass through those shared regions while moving from one gateway to another.

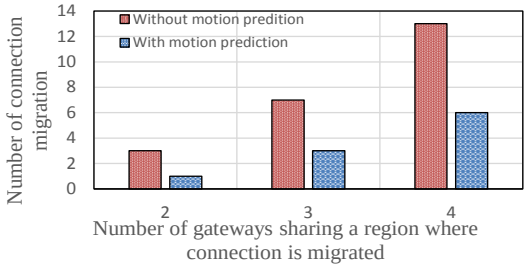


Figure 8: *SeamBlue* handles ping-pong effect by performing motion prediction during connection migration.

Figure 8 shows that if users move randomly every after two seconds, the number of times connection has been switched among gateways increases almost at 2X rate with the number of gateways sharing common regions. With the *SeamBlue*'s motion prediction mechanism, the number of connection switches among gateways reduces almost half times than the number without using motion prediction. In the case of four overlapping gateways, the users random movement direction causes a higher number of connection migration using our simple motion prediction mechanism. This can be improved by using sophisticated motion prediction techniques that leverage more information about users previous movements, geographical map, and applications.

7.3 BLE Connection Migration Cost

7.3.1 Extra bytes required for connection migration

In both *partial stack cloning* and *full stack cloning*, the current gateway sends a 512-bytes blob containing the bonding related information to each of its neighbors. However, for *full stack cloning*, the current gateways sends a 2048-bytes blob containing values of all the connection related variables to the next gateway where the connection is going to be migrated.

7.3.2 Time required for adding a peripheral as a bonded device

As part of the connection migration procedure, the current gateway sends bonding related information to its neighboring gateways. Upon reception of these information they

add the BLE peripheral as a bonded device into its NVRAM when they receive the bonding related information from the current gateway. To add the peripheral as a bonded device requires the IoT gateway to load the device information, e.g., device address, device type, address type, and keys from the main memory, and then store this information into the gateway's NVRAM for use in future communications. Table 3 shows the mean time required by IoT gateways of different device types to load a peripheral.

Gateway	Loading Time (ms)	Storing Time (ms)	Total Time (ms)
Nexus 5	40.5	19.1	60.4
Nexus 6	36.7	17.4	54.1
Alcatel OneTouch	43.2	20.3	63.5

Table 3: Time required for adding a peripheral as a bonded device.

Table 3 shows that the Nexus 6 smartphone requires the least amount of time to add a peripheral as a bonded device due to its higher CPU speed and memory capacity compared to other devices as shown in Table2.

7.3.3 Time required for transferring state variables

Figure 9 shows that the time required to transfer the state variables to a trusted and an untrusted gateway over WiFi for both *full stack cloning* and *partial stack cloning* approaches. We assume that the symmetric group key of trusted gateways and public keys of untrusted gateways are already known by the respective gateways beforehand. Figure 9 shows that the time required for transferring the state variables for untrusted gateways is almost 2X of trusted gateways for different speeds of the user with IoT device.

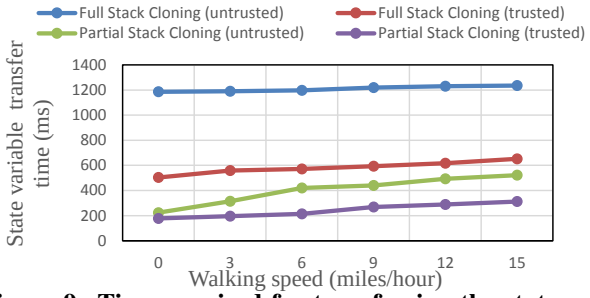


Figure 9: Time required for transferring the state variables.

Note that the *partial stack cloning* sends only the bonding related information and requires the next gateway to send connection request to the IoT devices for connection migration. Therefore, the *partial stack cloning* always requires less time for transferring state variables than *full stack cloning*. Also, since the control plane packet losses increases with users mobility, the time required for all the scenarios (shown in Figure 9) increases slightly with the increase of the users moving speed.

7.3.4 Time required for connection migration

The time required for connection migration in *full stack cloning* is computed by considering the time required to (1) extract the values of connection related variables, (2) send this information to the next gateways securely, (3) decrypt the received information, and (4) update the connection related state variables. On the contrary, the time required for connection migration in *partial stack cloning* is only the time to establish a new connection without further creating any pairing and bonding between the subsequent gateways and the IoT devices.

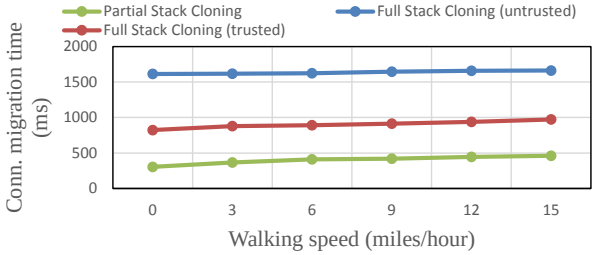


Figure 10: Time required for connection migration.

Figure 10 shows that the connection migration time increases with the increase of users speed as there are more packet losses associated with increased mobility of users. Also, the connection migration time for the *partial stack cloning* is smaller than that of the *full stack cloning* because creating a new connection between the subsequent gateway and the IoT device does not require any cryptographic operation in case of the *partial stack cloning*. However, we observed that the BLE connection between a peripheral device and a gateway is sometimes unstable. As a result, a few times the *partial stack cloning* required multiple connection requests for a single connection migration.

There is a trade-off between the full stack cloning and the partial stack cloning. The *SeamBlue*'s connection migration mechanism with *full stack cloning* does not require a new connection request from the target/next BLE gateway as opposed to the *partial stack cloning*. Thus *full stack cloning* requires a smaller number of message transmissions than the *partial stack cloning*. As a result, seamless migration with *full stack cloning* has lower power consumption compared to the *partial stack cloning*. However, since the *full stack cloning* requires a higher number of instrumented instructions as it needs to extract a higher number of connection state variables, it results in higher delays than the *partial stack cloning* for transferring connection. Since the *partial stack cloning* requires further connection request(s) from the subsequent gateway to an IoT device for connection migration, it incurs higher power consumption for sending/receiving more number of messages than the *full stack cloning*.

7.3.5 Packet loss

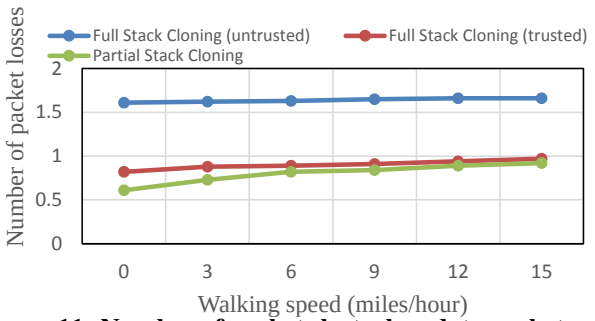


Figure 11: Number of packets lost when data packets are sent with 1 second time interval.

Figure 11 shows the number of packets lost as an impact of BLE connection migration when the heart rate monitoring

application running on an IoT device sends data to the gateway after every 1 second. *Full stack cloning* with untrusted gateway causes loss of at most 2 packets which are about 2X of the other scenarios. Since the number of packet losses has a direct relationship with the connection migration time, the trend is similar to the trend shown in Figure 10.

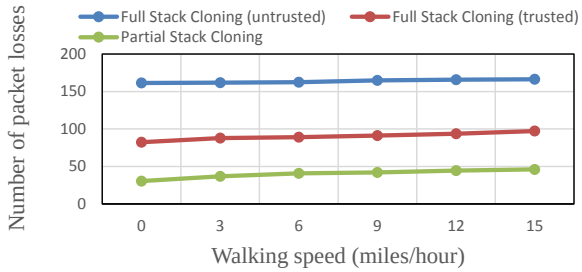


Figure 12: Number of packets lost when data packets are sent with 20 ms time interval (minimum connection interval for BLE connection).

Figure 12 shows a stress testing of packet losses when the heart rate data is sent every after 20 ms intervals which is the minimum connection interval for BLE devices. For *full stack cloning*, around 160 data packets were lost during connection migration which span the user's heart rate information for only about 1.5 minutes.

8 Security Analysis

In this section, we analyze the security features of our *SeamBlue* system.

- *Adversaries cannot inject or modify packets. SeamBlue* ensures that the gateways always perform secure communication with BLE devices through the long term keys established through pairing-bonding procedure and later on shared with the subsequent gateways during connection migration. *SeamBlue* uses AES-128 encryption mechanism for cryptographic operation which is already proven to be secure. Therefore, *SeamBlue* ensures that every packet sent or received by client and server is encrypted, authenticated, and integrity-protected. Thus maliciously injected or modified packets by adversaries are always identified.
- *Adversaries cannot derive paring-bonding and connection information. SeamBlue* gateways uses non-deterministic encryption to securely transfer the pairing-bonding information. Therefore, adversaries cannot derive the long term keys and the connection related values through which they can impersonate a legitimate gateway.
- *Adversaries cannot impersonate legitimate gateways.* During connection migration an adversary may try to impersonate a trusted subsequent gateway by mimicking the subsequent gateway's Bluetooth device address. However, if the legitimate subsequent gateway is one of the gateways which share the secret group key, the adversary cannot decrypt the pairing-bonding or connection related information since it does not have the secret group key. In the case of impersonating a legitimate subsequent gateway for which the public key certificate is already validated, the adversary cannot decrypt the pairing-bonding or connection related information since it does not know the private key of the legitimate gateway.
- *Adversaries cannot identify the gateways to which an IoT device is connected.* For an IoT device, each gateway in the *SeamBlue* system impersonates the initial gateway to which that device connects first. As a result, every packet destined to that BLE device includes the same sender address which baffles the adversaries to detect the gateway to which a benign BLE device is currently connected.

9 Related Work

Despite the heavy use of BLE for numerous smart applications, few research efforts [25, 19, 22] have been devoted to enable seamless connectivity for IoT devices. Zachariah et al. [38] addresses the problems of running different applications on a single gateway for different IoT services (e.g., heart rate monitoring, activity monitoring, smart home appliance monitoring, etc) and envision an application-agnostic connectivity for worldwide deployment of IoT gateways. In contrast, SeamBlue addresses the existing limitation of seamless connectivity and propose a framework for seamless connection migration for unmodified IoT devices. Kodeswaran et al. [27] identify timely maintenance of failed sensors as a critical task to ensure minimal disruption to monitoring services, and propose an approach to optimize maintenance scheduling. However, their approach does not consider the case when sensor devices moves out of the communication range of gateways or when the gateways suddenly fail.

Levy et al. [29] present a new hardware interface, *Beetle* that virtualizes peripherals at the application layer, allowing safe access by multiple programs without requiring the operating system to understand hardware functionality, fine-grained access control to peripheral device resources, and transparent access to peripherals connected over the network. Fawaz et al. [24] propose BLE-Guardian, a device-agnostic system that protects the privacy of the users/environments equipped with BLE devices/IoTs. However, neither of these approaches address the problem of seamless connectivity of IoT devices.

Mashable [37] proposes a mobile application that enables members of a secret community to discover other affiliates that are in proximity to their mobile devices. Das et al. [22] present a measurement-driven study of privacy leakage from communication between wearable fitness trackers and smart phones. These fitness trackers mostly use BLE for communicating and syncing the data with the user's smart phone. Albazrqaoe et al. [19] propose a bluetooth traffic sniffer, *BlueEar* where two Bluetooth-compliant radios can coordinate to learn the hopping sequence of indiscoverable Bluetooth network, to predict adaptive hopping behavior, and mitigate the impacts of RF interference.

As opposed to the cellular-handovers [30, 32, 36], the *SeamBlue* does not require modifications to the IoT devices for BLE connection migration. Like cellular-handovers, the *SeamBlue* reallocates BLE channels in the *partial stack cloning* through new connection establishment. However, in the *full stack cloning*, the *SeamBlue* transfers the BLE channels without creating a new connection.

10 Conclusion

In this paper, we focus on the problem of IoT devices being unable to connect to multiple gateways seamlessly and thus propose a framework that ensures seamless communication between a mobile IoT device and a remote service in a network of BLE gateway environment. Our framework consists of a static analysis module for the automatic extraction of the state variables required during a connection transfer. Moreover, we design a gateway selection mechanism that transfers connection related information to an optimal set of gateways and thus reduces both communication overhead and latency.

Acknowledgments

We would like to thank our Shepherd, Wen Hu, and the anonymous reviewers for their valuable comments and suggestions to improve the paper. For the work reported in this paper, the second author is supported by the Schlumberger Foundation under the Faculty for the Future Fellowship.

11 References

[1] *Bluetooth - Android Open Source Project*. http://llvm.org/.

[2] *Bluetooth 4.2 Core Specification*. https://www.bluetooth.com/specifications/bluetooth-core-specification/technical-considerations.

[3] *Bluetooth Low Energy*. https://www.bluetooth.com/what-is-bluetooth-technology/bluetooth-technology-basics/low-energy.

[4] *Bluetooth Low Energy*. http://groups.inf.ed.ac.uk/teaching/slipb13-14/Ewan/.

[5] *Control Flow Graph*. https://en.wikipedia.org/wiki/Control_flow_graph.

[6] *DiffieHellman key exchange*. https://en.wikipedia.org/wiki/Diffie-Hellman_key_exchange.

[7] *Disneys $1 Billion Bet on a Magical Wristband*. https://www.wired.com/2015/03/disney-magicband/.

[8] *The future of IoT in airports Lessons from London City Airport*. http://www.totalbluesky.com/2015/03/10/future-iot-airports-lessons-london-city-airport/.

[9] *How far is the hype surrounding claims of up to 50B IoT and machine-to-machine devices by 2020 away from reality?* http://www.rcrwireless.com/20160628/opinion/reality-check-50b-iot-devices-connected- 2020-beyond-hype-reality-tag10.

[10] *LLVM*. http://llvm.org/.

[11] *LLVM Language Reference Manual*. http://llvm.org/docs/MIRLangRef.html.

[12] *Near Field Communication*. http://nearfieldcommunication.org/.

[13] *nRF Connect for Mobile*. https://play.google.com/store/apps/details?id=no.nordicsemi.android.mcp&hl=en.

[14] *Official Linux Bluetooth Protocol Stack*. http://www.bluez.org/.

[15] Openssl. Technical report. https://www.openssl.org/.

[16] *Public Key Certificate*. https://en.wikipedia.org/wiki/Public_key_certificate.

[17] *Public Key Cryptography*. https://en.wikipedia.org/wiki/Diffie-Hellman_key_exchange.

[18] *Zigbee*. http://www.zigbee.org/what-is-zigbee/.

[19] W. Albazrqaoe, J. Huang, and G. Xing. Practical bluetooth traffic sniffing: Systems and privacy implications. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '16, pages 333–345, New York, NY, USA, 2016. ACM.

[20] A. W. Appel. Ssa is functional programming. volume 33, pages 17–20, 1998.

[21] Q. A. Chen, Z. Qian, Y. J. Jia, Y. Shao, and Z. M. Mao. Static detection of packet injection vulnerabilities: A case for identifying attacker-controlled implicit information leaks. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 388–400, New York, NY, USA, 2015. ACM.

[22] A. K. Das, P. H. Pathak, C.-N. Chuah, and P. Mohapatra. Uncovering privacy leakage in ble network traffic of wearable fitness trackers. In *Proceedings of the 17th International Workshop on Mobile Computing Systems and Applications*, HotMobile '16, pages 99–104, New York, NY, USA, 2016. ACM.

[23] R. Faragher and R. Harle. An analysis of the accuracy of bluetooth low energy for indoor positioning applications. In *Proceedings of the 27th International Technical Meeting of The Satellite Division of the Institute of Navigation (ION GNSS+ 2014)*, pages 201–210, 2014.

[24] K. Fawaz, K.-H. Kim, and K. G. Shin. Protecting privacy of ble device users. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 1205–1221, Austin, TX, Aug. 2016. USENIX Association.

[25] C. Gomez, J. Oller, and J. Paradells. Overview and evaluation of bluetooth low energy: An emerging low-power wireless technology. *Sensors*, 12(9):11734, 2012.

[26] J. Khalid, A. Gember-Jacobson, R. Michael, A. Abhashkumar, and A. Akella. Paving the way for nfv: Simplifying middlebox modifications using statealyzr. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 239–253, Santa Clara, CA, Mar. 2016. USENIX Association.

[27] P. A. Kodeswaran, R. Kokku, S. Sen, and M. Srivatsa. Idea: A system for efficient failure management in smart iot environments. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '16, pages 43–56, New York, NY, USA, 2016. ACM.

[28] C. Lattner, A. Lenharth, and V. Adve. Making context-sensitive points-to analysis with heap cloning practical for the real world. In *Proceedings of the 28th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '07, pages 278–289, New York, NY, USA, 2007. ACM.

[29] A. A. Levy, J. Hong, L. Riliskis, P. Levis, and K. Winstein. Beetle: Flexible communication for bluetooth low energy. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '16, pages 111–122, New York, NY, USA, 2016. ACM.

[30] A. Rath and S. Panwar. Fast handover in cellular networks with femtocells. In *2012 IEEE International Conference on Communications (ICC)*, pages 2752–2757, June 2012.

[31] S.-H. Seo, M. Nabeel, X. Ding, and E. Bertino. An efficient certificateless cryptography scheme without pairing. In *Proceedings of the Third ACM Conference on Data and Application Security and Privacy*, CODASPY '13, pages 181–184, New York, NY, USA, 2013. ACM.

[32] A. Sgora and D. D. Vergados. Handoff prioritization and decision schemes in wireless cellular networks: a survey. *IEEE Communications Surveys Tutorials*, 11(4):57–77, Fourth 2009.

[33] B. Steensgaard. Points-to analysis in almost linear time. In *Proceedings of the 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '96, pages 32–41, New York, NY, USA, 1996. ACM.

[34] Y. Tao, C. Faloutsos, D. Papadias, and B. Liu. Prediction and indexing of moving objects with unknown motion patterns. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, SIGMOD '04, pages 611–622, New York, NY, USA, 2004. ACM.

[35] F. A. Teixeira, G. V. Machado, F. M. Q. Pereira, H. C. Wong, J. M. S. Nogueira, and L. B. Oliveira. Siot: Securing the internet of things through distributed system analysis. In *Proceedings of the 14th International Conference on Information Processing in Sensor Networks*, IPSN '15, pages 310–321, New York, NY, USA, 2015. ACM.

[36] D. Wong and T. J. Lim. Soft handoffs in cdma mobile systems. volume 4, pages 6–17. IEEE, 1997.

[37] J. L. Yan Michalevsky, Suman Nath. Mashable: Mobile applications of secret handshakes over bluetooth le. ACM, July 2016.

[38] T. Zachariah, N. Klugman, B. Campbell, J. Adkins, N. Jackson, and P. Dutta. The internet of things has a gateway problem. In *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications*, HotMobile '15, pages 27–32, New York, NY, USA, 2015. ACM.