Evaluating Bluetooth Low Energy for IoT

Jonathan Fürst*, Kaifei Chen†, Hyung-Sin Kim† and Philippe Bonnet*

*IT University of Copenhagen, †UC Berkeley
jonf@itu.dk, {kaifei, hs.kim}@berkeley.edu, phbo@itu.dk

***Abstract*—Bluetooth Low Energy (BLE) is the short-range, single-hop protocol of choice for the edge of the IoT. Despite its growing significance for phone-to-peripheral communication, BLE's smartphone system performance characteristics are not well understood. As others, we experienced mixed erratic performance results in our BLE based smartphone-centric applications. In these applications, developers can only access low-level functionalities through multiple layers of OS and hardware abstractions. We propose an experimental framework for such systems, with which we perform experiments on a variety of modern smartphones. Our evaluation characterizes existing devices and gives new insight about peripheral parameters settings. We show that BLE performances vary significantly in non-trivial ways, depending on SoC and OS with a vast impact on applications.**

I. Introduction

Bluetooth Low Energy (BLE) is popular as a building block for the edge of the Internet of Things (IoT). IoT applications rely on BLE for local, energy-efficient data exchange between *smartphones* and resource constrained peripherals. Common uses of BLE are iBeacon based localization [1], smart wristbands/watches [2] and environmental sensors and actuators [3]. Most *smartphones* are readily equipped with BLE; This sets BLE apart from other low power wireless technologies such as ZigBee or Thread [4].

Our work is motivated by our own development experience of several smartphone applications using BLE [5], [6]. We observed that different smartphone models provide significantly different BLE performance. As an illustration, Figure 1 depicts that BLE advertisement latencies vary by an order of magnitude for different models. This variation can lead to non-predictable application behavior without extensive testing on all potential smartphone models in deployment.

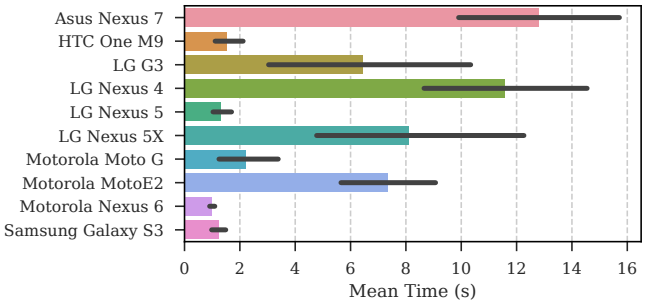


Fig. 1. BLE Advertisement Latency. Advertising interval is set to 1280 ms while smartphones scan in the default balanced mode. Smartphones are placed in 2 m distance. We perform 20 repetitions.

While others have experienced similar phenomena [7], we could not find systematic studies of BLE characteristics on *smartphones* in the literature. Most existing work focus on the BLE performance of BLE SoCs [8], [9], [10], [11], [12], without quantifying BLE performance in a smartphone context. However, smartphones-peripheral systems represent a large set of BLE applications in the wild [1], [2], [3]. In our work, we thus focus on two questions: (1) Can we show systematic variations in BLE application behavior across smartphone models? (2) Can we define a model to characterize BLE performance on smartphones, that could inform application design?

We identify three major factors that could possibly impact BLE performance on smartphones: (1) *OS implementation* that provides BLE abstractions to applications, (2) *SoC implementation* that schedules how multiple wireless modules, such as WiFi and BLE, share a single antenna and (3) *hardware components* such as amplifier, antenna, and case.

We formulate hypotheses about the impact of these factors on BLE performance, and we propose an open-source benchmarking framework, called *BLEva*, to verify them. We use BLEva to compare the performance of different BLE hardware and software combinations. More specifically, we evaluate BLE performance of ten different Android smartphone models which have various SoCs and support a range of different Android OS versions. Our results show that BLE provides totally different performance in terms of packet reception ratio, latency, and received signal strength indication (RSSI) on different smartphone models. Because of the complexity and number of combinations of the underlying impact factors, we argue that these different performance characteristics cannot be captured by a simple performance model.

We summarize our contributions as follows:

- We analyse the working of BLE on Android by tracing high-level API calls down the OS stack (Section II).
- We design and implement a benchmarking framework to easily perform and reproduce BLE related benchmarks on Android (Section III).
- And finally, we evaluate BLE experimentally on a variety of smartphone models (Section IV).

II. BLE Software and Hardware Abstractions

To understand possible sources for BLE performance variance, we consider the question: "What happens in a smartphone when a BLE control command is executed?"—After analysing various smartphone architectures and the Android

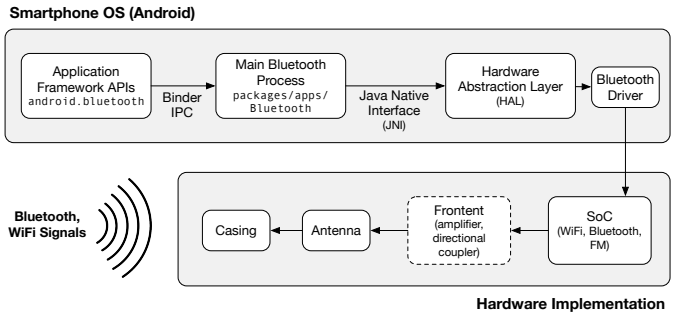


Fig. 2. Android BLE Stack: Tracing BLE down the software and hardware abstractions.

source code [13], we get a rough answer as shown in Figure 2. When an application executes a BLE command, it sequentially passes through smartphone OS, BLE driver, SoC including BLE chip, amplifier, smartphone antenna, smartphone cover, and finally becomes a BLE wireless signal in a channel. We now discuss how each of these components can impact BLE behavior. This analysis provides a set of hypotheses that drives our experiments in Section IV.

A. Smartphone OS (Android)

Android leverages modules from the Linux kernel for several hardware components like memory and power management, audio, video, WiFi and also Bluetooth. However, a developer is *not able to directly access that native Bluetooth stack* (`system/bt`) on non rooted, off-the-shelf devices. Further, device manufacturers implement their own proprietary Bluetooth drivers and can make extensions to the default stack.

Developers access BLE through several abstraction levels (see Figure 2) by calling the Bluetooth application framework APIs (`android.bluetooth`). Internally this code then calls a single, running Bluetooth process (`packages/apps/Bluetooth`) through interprocess communication (Binder IPC). The Bluetooth process then coordinates the different requests by possibly multiple applications to the HAL layer using the Java Native Interface (JNI). Bluetooth events are communicated back using callbacks (e.g., an advertisement that matches a filter is discovered). Even the Android source code is available, manufacturer specific extensions are *closed source* and specific Bluetooth hardware implementations are *undocumented*. Another problem is that the preemptive scheduling of the Linux kernel and Android OS specific energy saving features might block an application process for up to $110\,\mathrm{ms}$ as shown in [14]. Besides that, Android OS versions in use are highly segregated. BLE was introduced into Android in 2013 (API level 18). Since then, its API has iteratively evolved.

Hypothesis (1): BLE is accessed 'indirectly', through several (partly hidden) abstraction layers and proprietary drivers, which leads to unpredictable BLE behavior across smartphone models with different OS versions.

B. Hardware Implementation

After passing through smartphone OS, we still have several additional steps before transmitting a BLE wireless signal, which are related to *hardware*. First, the BLE driver cannot directly control the BLE chip but delivers a BLE command to the *SoC* which includes the BLE module. Interestingly, most modern SoCs in smartphones have other wireless modules, such as WiFi, together with BLE in a single chip, and use only a single antenna for all wireless modules. BLE must contend with other wireless modules in the SoC to occupy the antenna; (1) *BLE operation can be affected by SoC implementation, especially how to schedule many tasks for heterogeneous wireless modules*.

After a BLE task is executed, the next step is to pass the BLE chip implementation, which depends on Bluetooth version and chip manufacturer. (2) *Each BLE chip has its own way of frequency hopping, time synchronization, connection management, and etc., resulting in different BLE behaviors*. The next step is passing through the amplifier and the antenna to generate a BLE wireless signal, where (3) *different amplifiers and antennas will give different transmission power*. Finally, the BLE wireless signal passes through the smartphone cover as the first medium and enters into the air; (4) *different case designs may cause different signal attenuation*.

Unfortunately, the details of these hardware implementations are mostly not open, which requires an extensive experimental study to analyse their impact on BLE operation in smartphones. As such, our hypotheses are:

Hypothesis (2): SoC and BLE chip implementations are two major factors that lead to different BLE performance.

Hypothesis (3): Hardware components, such as amplifier, antenna, and cover, impact BLE performance.

III. BLEva

To test our hypotheses, we developed *BLEva*, a BLE benchmarking framework for smartphone-peripheral systems.[1]

A. BLEva Overview

BLEva allows a simple and reproducible evaluation of BLE on different smartphones (see Figure 3 for system overview). We implement BLEva using the widely available BLED112 that uses TI's CC2540 SoC. This chip is used in various BLE products ([15]), and provides all Bluetooth 4.0 features through a virtual serial port to the host while providing a simple application interface, its retail price is roughly $10 [16]. All benchmark results are committed to a repository together with the used configuration files.

Our implementation automatically maps JSON encoded benchmark configuration instructions to native parameters and function calls of Android and the BLED112/CC2540 programming interface. Each CC2540 is orchestrated through a separate Python process. On Android we streamline the differences between the APIs of Android 4.4, 5 and 6+ through an abstraction library that checks the device API, selects the correct API call and potentially emulates newer features for older devices (e.g., packet batching, filtering). This is especially important considering the high fragmentation

[1] BLEva is open-source and can be found together with all experimental data at http://github.com/jf87/BLEva

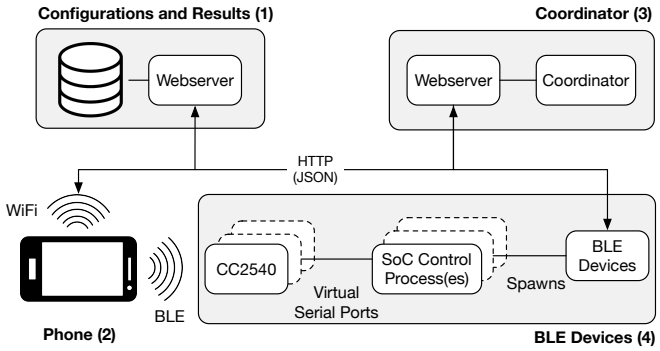


Fig. 3. BLE Benchmarking Architecture with four Components. (1) A repository that stores experiment configurations (these define the setup and different steps of a benchmark for phone and peripheral(s) in JSON) and benchmarking results. (2) The BLEva phone application reads these configurations and coordinates the execution of a benchmark through (3) a coordinator with (4) multiple BLE devices (CC2450), that connect to a Linux box via virtual serial ports (USB).

of Android. Our Android implementation uses foreground services to run multiple benchmarks reliably over an extended time period without the need for user interaction.

IV. Opening the Black Box

We now evaluate our three hypotheses through extensive experiments. Several BLE metrics impact application performance and will be used in our subsequent experiments:

- **Advertisement Packets Latency.** The time it takes to receive the first peripheral advertisement packet.
- **Advertisement Packets Reception Ratio (PRR).** Percentage of peripheral advertisement packets received over a fixed period.
- **Advertisement RSSI.** Received Signal Strength Indication of advertisement packets.
- **BLE Characteristic Write/Read Latency.** Latency of writing/reading a BLE characteristic.

Note, that we do not evaluate throughput as it is not a major design goal of BLE and not relevant for its applications [17].

A. Experimental Setup

We performed experiments on ten different Android models from 2013 to 2016 and from different performance and price ranges (see Table I). For the Nexus 7, Moto E2, Nexus 6 and Nexus 5 we use two devices each, for the other models we use one device. All models implement WiFi and Bluetooth on the same SoC. If not mentioned otherwise, experiments have been conducted in our lab where phones are placed in $1\,\mathrm{m}$ distance to a CC2540. Smartphones have been reset to factory state before experimentation. We use our own local WiFi network for communication (coordinating start of experiments on multiple phones, publishing benchmark results). RSSI related experiments for different distances have been conducted in a ca. $50\,\mathrm{m}$ long corridor where one of the sides is adjacent to a wall with doors to classrooms and labs, while the other side opens up to a ca. $30\,\mathrm{m}$ wide atrium. For RSSI, smartphones are placed on a movable tray and

TABLE I
SoCs found on Different Smartphone Models

BLE Chip	Phone Models and API Versions
WCN3620	Motorola Moto G (API 22), Moto E2 (API 23)
WCN3660	LG Nexus 4 (API 22), Asus Nexus 7 (2013) (API 23)
BCM4339	LG Nexus 5 (API 23), LG G3 (API 21)
BCM4356	Motorola Nexus 6 (API 24), HTC One M9 (API 23)
QCA6174	LG Nexus 5X (API 25)
BCM4330	Samsung Galaxy S3 (API 19)

we perform experiments simultaneously (all smartphones scan at the same point in time) to ensure that we have similar environmental conditions for each run. We conduct latency and packet reception ratio (PRR) related connection-less and connection-oriented experiments sequentially to avoid mutual interference[2].

B. Impact of OS Implementation

We first investigate *Hypothesis (1)*: Does smartphone OS implementation impact BLE performance? To this end, we installed four different Android versions (API version 19, 22, 23, and 24) in the same smartphone to exclude other impact factors (*Hypotheses (2) and (3)*). Version 19 was published in 2013 and version 24 in 2016[3]. The experiment is run on Asus Nexus 7, LG Nexus 5 and Motorola Nexus 6.

Figure 4 plots advertisement PRR with various Android OS versions. Smartphones use `balanced` scan mode with scan period of $30\,\mathrm{s}$ and the CC2540 uses an advertising interval of $160\,\mathrm{ms}$. Each result is based on 20 repetitions. First of all, Asus Nexus 7 (with Qualcomm SoC) shows substantial different PRR from the other two (with Broadcom SoC) even with the same API version. This is because a single API version has different BLE driver implementations for each SoC, especially depending on its vendor; The same API call can result in different BLE behaviors based on its SoC. Specifically, we confirmed that the Android BLE driver for Asus Nexus 7 (Qualcomm SoC) provides only one advertising packet for all API levels during the whole scan period, while other drivers (Broadcom SoC) provide all received packets. This causes extremely low PRR for the Asus Nexus 7 and reveals that *even a recent Android OS implementation fails to abstract various BLE chips for application developers*.

Furthermore, when looking at LG Nexus 5 and Motorola Nexus 6 which have Broadcom SoCs, we observe that the same smartphone exhibits varying performance with different OS implementations. Surprisingly, API 19, the oldest one, achieves much higher PRR ($\sim$100%[4]) than other versions. The

[2]Android devices can not passively scan for advertisement packets, but always scan actively (requesting a scan response packet from advertisers).

[3]We test 3 API versions for each model since none supports all 4 versions.

[4]Note that we define a successful packet reception when BLE receiver receives one out of the three packets sent out during an advertising interval on the three different channels.

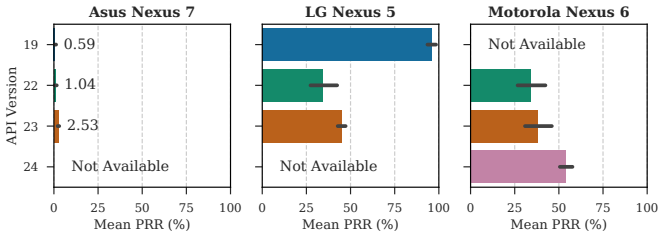


Fig. 4. Advertisement Packet Reception Ratio (PRR) for the Same Smartphone on Various OS Versions: OS versions (and BLE driver implementations) significantly impact BLE performance.

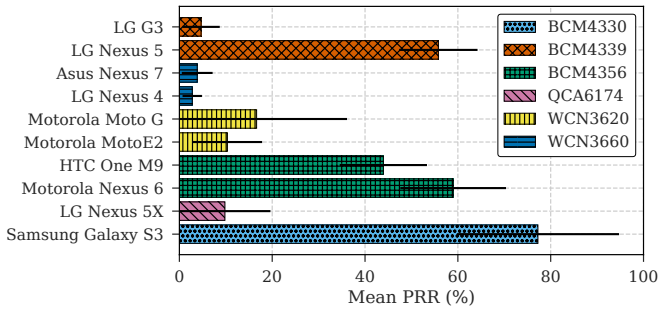


Fig. 5. Advertisement PRR of Phone Models: SoC significantly impacts PRR. Phones with the same SoC exhibit similar performance, except for LG G3 because of highly customized OS.

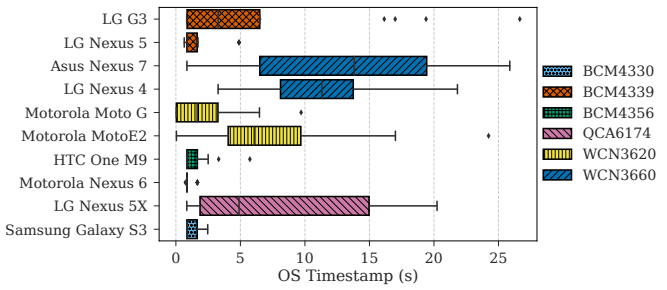


Fig. 6. Advertisement Latency of Phone Models: Advertising latency varies with both SoC vendor and SoC model of a same vendor.

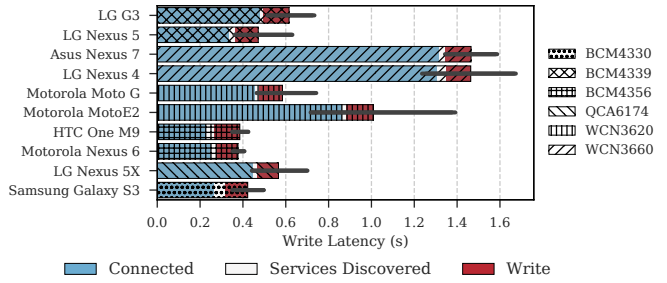


Fig. 7. Write Latency of Phone Models: Write latency is SoC-bound, but varies significantly across SoCs.

reason is that API 19 always scans the channel as a default (the behavior of `low_latency` scan mode). Newer API versions (22, 23, and 24) support a differentiated operation for each scan mode. This sacrifices PRR of `balanced` mode for saving battery, but Android gradually improves the PRR performance as its API version is upgraded. Overall, *smartphone OS implementation significantly impacts BLE performance*.

C. Impact of OS/SoC Implementations

We now look at *Hypotheses (1) and (2)* together. Note that it is impossible to nullify the OS impact on BLE performance (by installing the same OS on all smartphones) because most smartphones have their own, customized and closed sourced Android OS implementations and do not allow any change.

a) ***Connection-less and Connection-based Services****:* Figure 5 plots advertisement PRR of the ten smartphone models in *observer* role with `balanced` scan mode and 30 s scan period.[5] The CC2540, located 1 m from the smartphones, uses an advertising interval of 1280 ms, all three advertising channels and a transmission power of 3 dBm. It shows that *SoC significantly impacts advertisement PRR*. Performance varies across SoC vendors, and Soc models of a same vendor.

Conversely, different smartphone models with the same SoC exhibit similar performance. An exceptional case is the pair of LG G3 and LG Nexus 5 that have BCM4339 SoC: their PRR varies by around 50%. LG G3 provides quite different performance from other smartphones with Broadcom SoCs. We suspect that LG G3's OS implementation is largely customized from the standard API 21 and provides similar behavior as BLE drivers for Qualcomm Soc do (i.e., returning only one advertising packet on APIs).

With the same setup, Figure 6 plots advertisement latency (the time until a smartphone received the first advertisement after starting to scan). We observe that *advertising latency varies with both SoC vendor and SoC model of a same vendor*. It seems that PRR and latency are inversely correlated. Furthermore, it turns out that current Android OS poorly supports Qualcomm SoCs since they provide both significantly longer latency and lower PRR than Broadom SoCs. On the other hand, Android OS implementation for Broadcom SoCs provides low and stable advertising latency regardless of its API version: even the oldest API at Samsung Galaxy S3 provides low latency. Only LG G3 behaves differently and more similar to Qualcomm SoCs.

Lastly, we performed connection-based experiments, where the CC2540 chip takes the *peripheral* role with advertising interval of 160 ms and requests a connection interval of 20 ms when a smartphone connects to it. We disable slave latency and use a supervision timeout of 300 ms. Then, a smartphone triggers a write operation to change a BLE characteristic of the CC2540 chip, which requires to (i) connect, (ii) discover services, and (iii) exchange data with the CC2540 chip given that it has already discovered the peripheral.

Figure 7 shows the latency for each of the three steps. Interestingly, the write latency is SoC-bound. Smartphones with same SoC show nearly the same latency values. However, the write latency varies significantly across SoCs. For example, HTC One M9 and Motorola Nexus 6 (BCM4356) are more

[5]Android implements `balanced` scan mode with a window size of 2 s and a scan interval of 5 s.

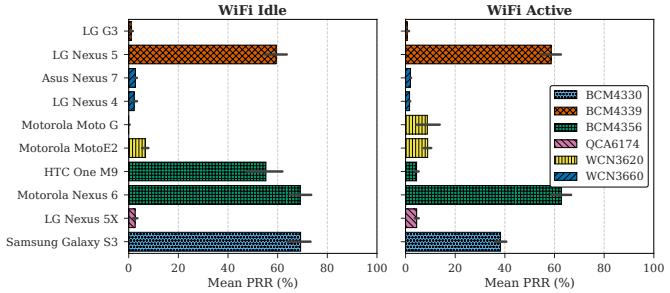


Fig. 8. Advertisement PRR with Idle/Active WiFi: Antenna scheduling performance is highly affected by SoCs but also by different OS versions (BLE driver implementations).

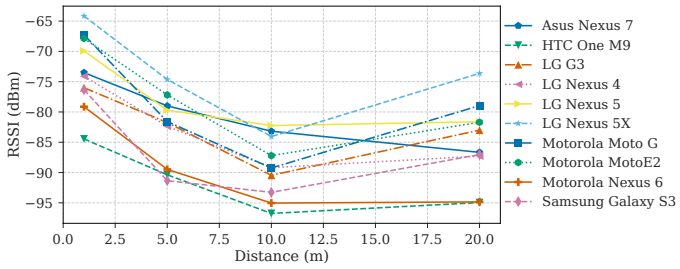


Fig. 9. Distance vs. Mean RSSI of Phone Models: The impact of the smartphone model nullifies the impact of distance on RSSI in many cases.

than three times faster than Asus Nexus 7 and LG Nexus 4 (WCN3660). More specifically, latency for the connection step is the longest among the three steps and shows the highest variance (across models) of all steps. The reason is that the connection step needs to be initiated as central and peripheral are not time synchronised. Despite that, the time for establishing a connection should be close to the advertising interval (160 ms). This is not the case for most phones, especially for those with Qualcomm SoCs.

*b) **Antenna Sharing with WiFi**:* Next, we investigate how OS and SoC implement antenna scheduling for multiple wireless modules, such as WiFi and BLE, to share a single antenna. We conduct a set of connection-less experiments while WiFi radio is (i) idle or (ii) active to receive downlink transmissions (from access point to smartphone). For BLE operation, the CC2540 chip advertises with interval of 160 ms and a smartphone uses scanning period of 30 s with `balanced` mode. At the same time, a smartphone keeps downloading a 1 MB file using its WiFi module from a local server. We perform 20 repetitions for each result.

Figure 8 shows the advertisement PRR results for our ten different smartphone models. As expected, simultaneous WiFi operations decrease PRR. Furthermore, in most cases, performance trends among smartphone models when WiFi is active is similar to those when WiFi is idle; different SoCs give different results. This reveals that *antenna scheduling performance is highly affected by SoC*. Surprisingly however, different smartphones, with the same SoC, show highly different performance for antenna scheduling. For example, the HTC One M9 and the Motorola Nexus 6 (with BCM4356) have a PRR of $> 50\%$ when WiFi is idle. However, when WiFi is active, PRR of the M9 drops under 10% while that of the Nexus 6 is marginally degraded. Given that these two smartphones have different Android API versions, we can see that *OS implementation also significantly impacts antenna scheduling behaviors.* Another interesting discovery was that when a smartphone scans with `low_latency` mode, the antenna can be fully occupied by BLE for scan period and any WiFi operation is blocked (figures omitted for brevity).

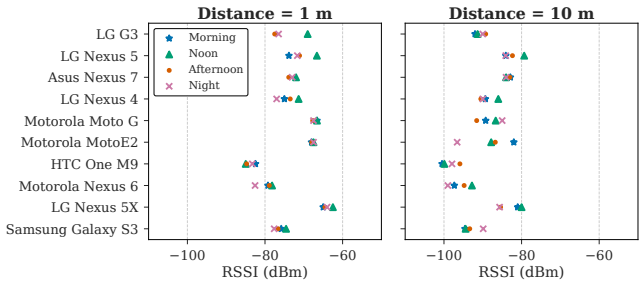
D. Impact of OS/SoC/Hardware

We explore all the three hypotheses together. To this end, we vary the distance between smartphone and CC2540 chip, and we measure RSSI, the key link quality metric for iBeacon like localization systems.

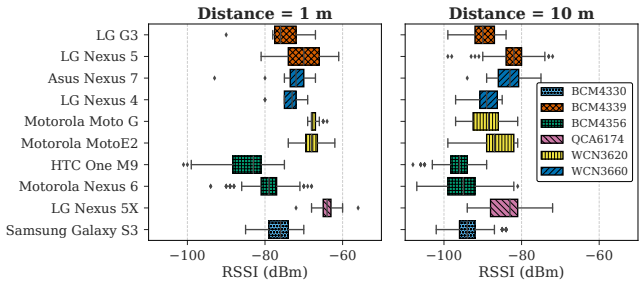
*a) **Distance vs. Model**:* Figure 9 depicts the mean RSSI of each smartphone with varying distance for all three advertisement channels. We first observe that as distance increases, the mean RSSI first decreases then increases again. This is due to indoor channel environments that degrade localization accuracy as discussed in [18]. More importantly, different smartphones provide significantly different RSSI. Maximum RSSI difference among smartphone models is around 20 dBm for every distance. Also there is no simple mapping between RSSI and smartphone models. Furthermore, different results at 1 m imply that a single reference RSSI value of iBeacon cannot be applied to all smartphones. In practice, unfortunately, iBeacon provides only one reference RSSI (i.e., optimized for a single smartphone model, e.g., the iPhone). As an extreme case, the smallest RSSI values of LG Nexus 5 and LG Nexus 5X (at 10 m) are even larger than the largest one of HTC One M9 (at 1 m): for these smartphones, *the impact of smartphone model nullifies the influence of distance on RSSI.*

*b) **Link Fluctuation vs. Model**:* Based on the above findings, we ask another question, "do the different RSSI readings *really* come from different smartphone models rather than fluctuating wireless links?" [19]. To seek an answer, Figure 10(a) plots the mean RSSI values of each smartphone model at 1m and 10m at four different times of day. Interestingly, this figure shows that almost all smartphone models provide consistent mean RSSI values even at different times. We conjecture that these stable results come from frequency hopping of BLE, which minimizes the impact of channel fading or wireless interference on RSSI values. In contrast, different smartphone models exhibit quite different mean RSSI values, up to 20 dBm at the same distance.

To take a deeper look, Figure 10(b) plots every RSSI sample during the whole experiment period with a boxplot. The variation patterns of RSSI vary across models. For example, RSSI values of LG Nexus 5X, Motorola Moto G, and Motorola MotoE2 are relatively consistent at 1 m but vary significantly at 10 m. On the other hand, LG Nexus 5 shows an opposite behavior: its RSSI is fluctuating at 1 m but stable at 10 m. The other smartphones provide diverse RSSI values for both 1 m and 10 m. These results reveal that *RSSI fluctuation over time highly depends on smartphone model*. Furthermore, even though Figure 10(b) plots all RSSI sample values, the RSSI



(a) Mean RSSI value at each of four measurement times



(b) Every RSSI sample value during the whole experiment period

Fig. 10. RSSI of Each Phone Model at Different Distances and Time of day: Smartphone model impact on RSSI is larger than the impact of wireless link fluctuation.

readings are still distinguished among different smartphone models. separated from that of LG Nexus 5X, Motorola Moto G, and Motorola MotoE2; not even a single RSSI sample value is overlapped together. Overall, Figures 10(a) and 10(b) confirm that *the impact of smartphone model on RSSI is larger than that of wireless link fluctuation* and should be addressed first for accurate localization.

Another observation from Figure 10(b) is that when different smartphone models have the same SoC, their RSSI readings become similar even with different OS implementations and hardware components (e.g., Motorola Moto G and MotoE2). This reveals the impact of SoC on RSSI. A notable exception is the case of LG G3 and LG Nexus 5 that have BCM4339, which provide quite different RSSI values even with the same SoC. It is not surprising because they also show different behaviors in Figures 5 through 8. Compared to other factors, it seems that hardware components (casing, antenna, frontend) have less impact on BLE performance.

*c) **OS, SoC, Hardware, and Instance**:* Finally, we compared the performance of two smartphones of the same model. Figure 11 shows that their performance is similar, both for RSSI and latency. This confirms that *variations in BLE performance come from different 'smartphone models' rather than 'smartphone instances'*.

V. Discussion and Conclusion

Our experiments show that performance varies significantly with OS version on a same smartphone model, and across SoCs, while other hardware factors do not have any measurable impact. Put differently, our experiments tend to support our first two hypotheses but not the third one.

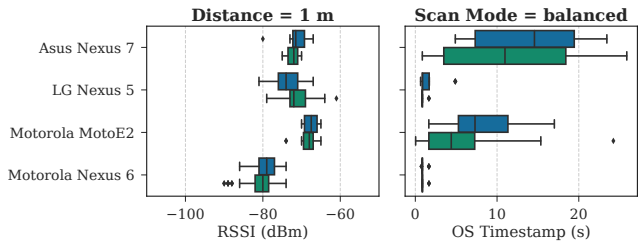


Fig. 11. RSSI and iBeacon Latency for Different Phone Instances of the Same Model: Different instances of the same model exhibit similar performance.

Should applications adapt to the smartphone models they run on? Our experiments suggest that introducing a different reference RSSI for each smartphone model would improve iBeacon localization accuracy. This illustrates the potential benefits of BLE-conscious applications.

What does it take for applications to become BLE-conscious? Our results do not provide conclusive evidence that a simple performance model (based on SoC and Android version) could represent BLE characteristics and thus inform adaptive applications. We conjecture that new OS abstractions, based on Quality of Service constraints, are needed to support BLE-conscious applications. This is a topic for future research.

References

[1] P. Martin, B.-J. Ho, N. Grupen, S. Munoz, and M. Srivastava, "An ibeacon primer for indoor localization," in *BuildSys'14*. ACM, 2014.

[2] "Fitbit," https://www.fitbit.com, 2017.

[3] Eqiva, "Bluetooth smart radiator thermostat," http://www.eq-3.com/products/eqiva/bluetooth-smart-radiator-thermostat.html, 2017.

[4] Thread group, "Thread 1.1.1 specification," 2017.

[5] J. Fürst, K. Chen, M. Aljarrah, and P. Bonnet, "Leveraging physical locality to integrate smart appliances in non-residential buildings with ultrasound and bluetooth low energy," in *IoTDI'16*. IEEE, 2016.

[6] J. Fürst, A. Fruergaard, M. H. Johannesen, and P. Bonnet, "A practical model for human-smart appliances interaction," in *BuildSys'16*, 2016.

[7] J. Adkins and P. Dutta, "Monoxalyze: Verifying smoking cessation with a keychain-sized carbon monoxide breathalyzer," in *SenSys*, 2016.

[8] S. Kamath and J. Lindh, "Measuring bluetooth low energy power consumption," *Texas instruments application note AN092, Dallas*, 2010.

[9] M. Siekkinen, M. Hiienkari, J. K. Nurminen, and J. Nieminen, "How low energy is bluetooth low energy? comparative measurements with zigbee/802.15. 4," in *WCNCW'12*. IEEE, 2012.

[10] J. Liu, C. Chen, Y. Ma, and Y. Xu, "Energy analysis of device discovery for bluetooth low energy," in *Vehicular Technology Conference (VTC Fall), 2013 IEEE 78th*. IEEE, 2013, pp. 1–5.

[11] S. Silva, S. Soares, T. Fernandes, A. Valente, and A. Moreira, "Coexistence and interference tests on a bluetooth low energy front-end," in *SAI'14*. IEEE, 2014.

[12] T. Lee, M.-S. Lee, H.-S. Kim, and S. Bahk, "A synergistic architecture for rpl over ble," in *SECON'16*, June 2016.

[13] Google, "Android Source," https://android.googlesource.com, 2016.

[14] M.-M. Moazzami, D. E. Phillips, R. Tan, and G. Xing, "Orbit: a smartphone-based platform for data-intensive embedded sensing applications," in *IPSN'15*. ACM, 2015.

[15] Texas Instruments, "Cc2540," http://www.ti.com/product/CC2540, 2015.

[16] Bluegiga, "BLED112 Bluetooth Smart Dongle," https://www.bluegiga.com/en-US/products/bled112-bluetooth-smart-dongle/, 2015.

[17] R. Heydon, *Bluetooth low energy: the developer's handbook*. Prentice Hall Upper Saddle River, 2013, vol. 1.

[18] J. Paek, J. Ko, and H. Shin, "A measurement study of ble ibeacon and geometric adjustment scheme for indoor location-based mobile applications," *Mobile Information Systems*, 2016.

[19] H.-S. Kim, H. Cho, M.-S. Lee, J. Paek, J. Ko, and S. Bahk, "Marketnet: An asymmetric transmission power-based wireless system for managing e-price tags in markets," in *SenSys'15*. ACM, 2015.