One GPU to Snoop Them All: a Full-Band Bluetooth Low Energy Sniffer

Marco Cominelli
Dept. of Information Engineering
University of Brescia
Brescia, Italy
m.cominelli006@unibs.it

Paul Patras
School of Informatics
The University of Edinburgh
Edinburgh, Scotland, UK
ppatras@inf.ed.ac.uk

Francesco Gringoli
Dept. of Information Engineering
CNIT/University of Brescia
Brescia, Italy
francesco.gringoli@unibs.it

***Abstract*—Sniffing Bluetooth data sessions is considered a difficult task, because of the frequency-hopping channel access scheme this technology implements. In this paper we present a novel open-source sniffer that can monitor Bluetooth Low Energy (BLE) traffic on all channels in real time. The sniffer builds on an Software-Defined Radio (SDR) framework to capture the entire BLE spectrum and exploits Graphics Processing Unit (GPU) capabilities to channelise and process BLE traffic in real-time. We show that our sniffer can easily and reliably detect active BLE connections, and infer their properties, including Access Address, CRC values and hopping sequences. From a general standpoint, we show that tracking many BLE data sessions at the same time becomes feasible even with relatively inexpensive equipment, as we are able to discover up to 24 simultaneous sessions within 80 ms on average.**

***Index Terms*—Bluetooth Low Energy, graphics processing unit (GPU), real-time sniffer, software-defined radio (SDR)**

I. Introduction

As the Internet of Things (IoT) gains popularity in many application domains, the number of smart devices with wireless connectivity is soaring [1]. Bluetooth technology will play a crucial role in the growing ecosystem of connected devices, with market reports forecasting more than 4 billion Bluetooth-powered devices shipped every year [2]. Starting with version 4.0 of the Bluetooth Core Specification, the Bluetooth Special Interest Group (SIG) introduced a new version of the protocol called BLE, which is not backward-compatible with the previous versions of the standard. BLE is better suited for devices that have low data rate requirements and strict constraints on power consumption, hence becoming the *de facto* standard for many IoT applications.

Despite the commercial success of BLE, open-source tools that provide a complete and efficient framework for sniffing BLE communications over-the-air are yet to appear. BLE technology operates in the unlicensed 2.4 GHz ISM (Industrial, Scientific and Medical) band and employs an Adaptive Frequency-Hopping Spread Spectrum (AFH) technique for accessing the physical medium, which makes debugging BLE data sessions a difficult endeavour. In fact, while AFH has been adopted to combat interference from other wireless devices, it also makes eavesdropping on radio signals difficult.

In general, we identify two challenges that arise when attempting to snoop BLE communications: 1) the central frequency of successive transmissions is not fixed but is rapidly switched between a set 40 narrow-band channels, according to a pseudo-random hopping sequence known only by the transmitter and the receiver; and 2) multiple data sessions (i.e. connections between different devices) can be active at the same time on different channels. Therefore, the only way to reliably track all BLE sessions in a target area is to capture a wide-band 80 MHz signal corresponding to the entire 2.4 GHz ISM band and then recover the BLE traffic, as we have demonstrated recently for the case of Bluetooth Classic [3].

The work of F. Gringoli and M. Cominelli was partially supported by the European Commission in the framework of the H2020 ORCA project (Grant agreement no. 732174) through the Third ORCA Open Call for Experiments ORCA-OC3-EXP project CSI-MURDER.

A. BLE Debugging Tools

The increasing interest towards BLE technology in the last few years led to the development of a number of sniffing platforms that differ in complexity, cost and sniffing capabilities. Here, we briefly review the most popular ones, highlighting their shortcomings.

The most common devices used to analyse BLE communications are usually small USB dongles connected to a host computer. The Ubertooth One for example—originally designed for eavesdropping on Classic Bluetooth and recently updated to partially support also BLE—has been used to capture active data sessions in [4]. Both passive and active attacks, such as connection hijacking, have been demonstrated using the Micro:bit system [5]. Similar platforms that offer the same capabilities are BLE development kits from Nordic Semiconductor or the Adafruit Bluefruit LE sniffer [6].

These devices share the advantage of being inexpensive and are well supported by the community, with many open-source applications. Their main disadvantage, however, is that they can only listen on a single channel, therefore it is hard to discover connection parameters of existing data sessions and impossible to track more than one connection at a time. Moreover, since all use a commercial Bluetooth transceiver embedded in the hardware platform, they cannot be updated to support more recent versions of the standard. At the time of writing, the latest version of the Core Specification is 5.2; BLE has changed drastically with version 5 (released in 2018), introducing support for higher data rates and coded physical layers for transmissions with improved reliability. It is important to note that the transceivers used by all these systems cannot support the new physical layers introduced with BLE 5.

To overcome these limitations, we consider SDRs as the radio front-end, which offer superior performance and more flexibility. A framework specifically designed for sniffing wireless protocols in the 2.4 GHz ISM band is presented in [7]; however, the proposed system cannot operate in real-time and requires to store large amounts of data before processing. To our knowledge, no implementation capable of decoding all

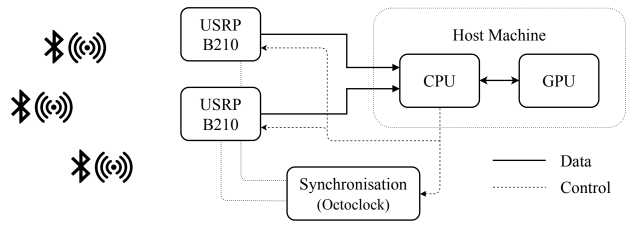

Fig. 1. Overview of the proposed BLE sniffer architecture.

BLE channels concurrently in real-time has been developed to date. One shortcoming of SDR-based applications is that they are demanding in terms of computational power. A common solution to ensure real-time operation would be to employ an Field-Programmable Gate Array (FPGA) for the processing stage. This in fact the approached used by commercial products, such as the Ellisys [8], yet performance comes with a substantial price tag, opaque design, and no ability to customise the behaviour of the platform.

B. *Contribution*

To the best of our knowledge, we are the first to develop a practical open-source framework based on SDRs that can simultaneously track the traffic on all the 40 BLE channels in real time. Rather than relying on complex FPGA designs for processing the radio signal, we use a general-purpose GPU, which can be easily installed on a host machine, can be rapidly and fully re-programmed via software, and is immediately available for any other processing task. We believe that with this system developers and security researchers will be able to debug BLE applications more efficiently. Moreover, we explore the capabilities of GPUs in boosting the performance of SDR-based applications.

C. *Structure of the paper*

The rest of the paper is organised as follows: in Section II we present the hardware setup underpinning the system and motivate the main design choices; in Section III we describe in greater detail the signal processing chain and how BLE packets are extracted from a sampled wide-band signal; in Section IV we show preliminary results; finally, in Section V we briefly discuss further implications of our work.

II. System Architecture

The sniffer, sketched in Fig. 1, can be logically divided into three distinct parts:

- a radio front-end composed of multiple SDRs that when operating together—i.e. joining their bandwidth—can capture the entire 2.4 GHz ISM band;
- a processing back-end implemented on the host for 1) separating narrow-band BLE channels from the wide-band signal acquired by each SDR and 2) extracting BLE packets from those channels;
- a synchronisation mechanism used to produce coherent timestamps on channels acquired by different SDRs.

The system can operate in real-time. We assume that a small latency due to buffering in the processing chain is tolerated.

A. *Radio Front-end*

The radio interface of our system is composed of two Ettus USRP B210 boards. Each USRP B210 can sample up to 56 MHz of instantaneous bandwidth from 70 MHz to 6 GHz and can acquire IQ samples with a "double" floating point resolution of 64 bits per component. A single B210 is clearly not sufficient to capture the entire 2.4 GHz ISM band, therefore we need two of them. We tune them respectively onto the 2,420 MHz and 2,462 MHz centre frequencies and set an IQ sampling rate of 40 MHz on both boards. This accounts for a total data rate towards the host machine of approximately 640 MB/s. To provide enough speed, the boards in the SDRs front-end are connected to the host machine using two USB 3 controllers.

B. *Processing Back-end*

We develop our BLE sniffing system on a computer with an Intel Core i7-7700K and 16 GB of memory, which run the Ubuntu 16.04 operating system. The host computer is also equipped with an Nvidia GTX 1080 GPU with 8 GB of memory. The real-time operation of the system is achieved by leveraging the parallel architecture of the GPU.

In general, GPU-based applications exhibit excellent performance in terms of computational power but suffer from repeated data transfers to/from host memory due to high latency and limited bandwidth. To limit this problem, samples are first stored in a temporary buffer (e.g. up to 1 s or 2 s) and then transferred in batches between the host and the GPU. Keeping in RAM samples from the wide-band signals can require a large amount of memory—up to a few GB, depending on the size of the temporary buffer. On the other hand, since everything is kept in RAM, the footprint on the hard drive is minimal and depends only on the number of detected packets that will be added to the capture file. This is a great advantage with respect to the work in [7], which required substantial space to be available on the hard drive.

C. *Time Synchronisation*

The two USRP B210 boards in the radio-front end are required to capture time-aligned IQ samples, in order to produce coherent timestamps for the BLE packets received in the lower and higher portion of the spectrum. To synchronise the operation of the SDRs in the time domain, we use an external clock distribution module, namely the Ettus OctoClock. This module provides both a 10 MHz reference clock signal and a 1 PPS signal, both of which are fed to the two B210 boards and are used to discipline the IQ sampling.

Using a clock distribution module is the best solution in terms of accuracy, but admittedly can prove relatively expensive; therefore we also design a cheaper, yet less accurate, alternative time synchronisation procedure. In this latter case we can increase the bandwidth of the two SDR in the radio front end, so that they overlap at the centre of the 2.4 GHz ISM band. We can then program an inexpensive BLE dongle (such as a Nordic nRF52840) to transmit beacons at (known) regular time intervals on a fixed central frequency that is captured by both radios. Using the timestamp of the beacons, we are able to synchronise the timestamps of all the BLE packets in a post-processing phase.

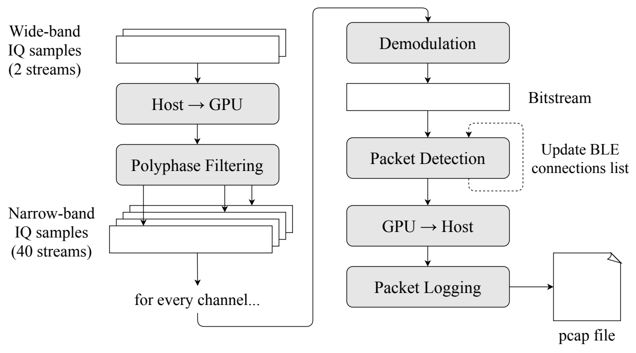

Fig. 2. Diagram of the processing chain. Operations are executed in parallel by assigning a GPU core to every IQ sample or bit in a single channel stream. Each task is then repeated for every channel.

1 B	4 B	2 B	0-255 B	3 B
Preamble	Access Address	Header	Payload	CRC

Fig. 3. Structure of a generic BLE packet. Header, Payload and CRC are whitened using a function that depends only on the channel number, hence dewhitening is performed immediately upon reception.

III. BLE Processing Chain

The sniffer is capable of processing all BLE sessions in real time by offloading the most demanding operations of the signal processing chain to the GPU. The processing chain is illustrated in Fig. 2.

A. IQ Samples Processing

IQ samples captured by each SDR are temporarily stored in a buffer on the host and then moved in batches to the GPU memory. By using a double buffer, we ensure a continuous flow of samples from the SDRs to the processing back-end. A polyphase filter is used to channelise the wide-band signal of each SDR into 20 narrow-band signals (this is done on both the receiving chains). The resulting narrow-band signals correspond to the 40 BLE channels. The structure of polyphase filtering is well-suited for the highly-parallel architecture of the GPU and the implementation is fairly efficient [9].

All BLE transmissions use a binary Gaussian Frequency-Shift Keying (GFSK) modulation. This is a very simple modulation that encodes transmitted bits in the frequency deviation of the signal from the carrier. For every channel we convert the sequence of IQ samples into the corresponding bitstream by discriminating the phase difference between successive samples. In this case, all the operations can be performed in parallel and again the GPU excels in boosting the overall decoding speed.

B. Bitstream Processing and Packet Logging

At this point the system has to process the 40 bitstreams—one for each channel—looking for new BLE packets. However, not all the 40 channels are 'equal' and in particular they are divided into two categories: 3 advertising channels that are reserved for discovering new devices and broadcasting information, and the remaining 37 data channels that are used when two or more devices are connected according to a master/slave paradigm. We will show how to take this difference into account when processing advertising and data channels, but the procedure for detecting new BLE packets is the same and is based on correlating the bitstreams with some known fields of the packets.

The structure of a generic BLE packet is reported in Fig. 3. A short preamble of alternating 1's and 0's precedes the Access Address (AA), a pseudo-random sequence of 32 bits used to identify a data session. The length of the payload is encoded in the Header field. The header and payload are protected by a 24-bit Cyclic Redundancy Check (CRC).

Given that the packet preamble is really short, we cannot rely only on it to detect BLE packets because the number of false positives is very high. We must correlate the bitstream both with the preamble and the AA, i.e. with a 40-bit long sequence, in order to be sufficiently confident that we have detected a packet. Once a packet is detected, we can then validate its reception by checking that the CRC received and the one computed from the packet content match. This is straightforward for Advertising Packets, where the AA and the CRC initialisation value are known quantities, fixed by the standard. However, when a master and a slave devices form a piconet, these parameters are pseudo-random values that are not known a priori.

Our sniffer can discover and track other BLE connections in two distinct ways. In the first case, the sniffer is active and intercepts a `CONN_IND` packet on one Advertising channel. This packet, which signals the beginning of a new connection, encodes in clear in its payload the AA and the CRC that will be used in the connection. These parameters are added to a list of known values and the discovery of subsequent data packets works in the same way as for advertising packets. In the second case, a target BLE connection has been established before the startup of the sniffer or outside its range; in such scenarios, the discovery of the same parameters requires a little more effort. Here we exploit the fact that when devices are connected, they shall continuously transmit packets—even empty ones—in order to stay synchronised while hopping. Packets from the master are often followed by a response from the slave on the same channel after a short inter-frame space. This implies that we will see the same 40-bit sequence (Preamble and AA) repeating twice during a short time window on the same data channel (it is worth noting that the AA is never encrypted nor whitened). Even if the payload is encrypted, we are still able to validate packet reception because the header and the CRC are applied after encryption. To recover the CRC initialisation value, one can simply check which one of the possible 2^{24} values matches the CRC received. Obviously, with this method we can get the correct initialisation value only if BLE packets are received without errors; however, one can become more and more confident about the result if multiple packets agree on the same CRC initialisation value.

When we have identified all the BLE packets in a trace, we still have to write them to a capture (pcap) file. At this point it is not possible to save any data to file, because all the information is still kept in the GPU memory. When a GPU core identifies a valid BLE packet on one single-channel bitstream, it marks the first bit of the packet with a flag. Note that every bit in the bitstream is encoded as `uint8_t` and we only use the least significant bit to represent its value. This means that we have room for up to 7 flags if we use for example a one-hot encoding for the control information. With this method, when the 40 bitstreams are moved back to the host memory, the host Central Processing Unit (CPU)

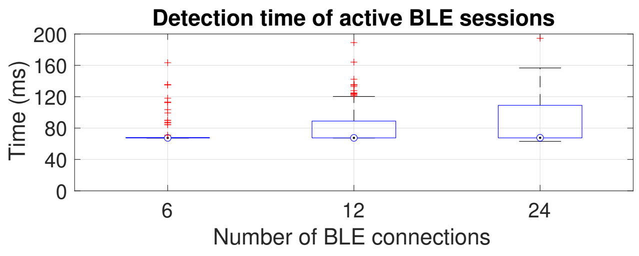

Fig. 4. Boxplot of the time required to identify all the BLE connections within the range of the sniffer in the considered testbed.

can rapidly skim through them looking for specific flags only, without double-checking if BLE packets are present and correctly received. The only task for the host—quite easy at this point—is to recover the payloads and produce a capture file with all the packets properly ordered in time.

IV. Performance Evaluation

We set up two testbeds to evaluate the performance of our system. In the first one, we implement a BLE transmitter on a Nordic nRF51 development kit. The transmitter emulates a target BLE connection hopping on all the channels and sending 1,000 packets on each. The packets have the same arbitrary AA and embed a sequence number in the payload. We count how many packets our sniffing system detects on every channel in order to extract statistical information about the detection rate. Every packet needs to pass the CRC verification in order to be counted as correctly received. On each channel the system detects on average 99.65% of the transmitted packets. The detection rate is 100% on 23 channels while the worst detection rate we measure on one channel is 97%. It is important to notice that interference due to Wi-Fi and other Bluetooth transmissions is present in our testbed, hence some packets might be discarded during the CRC verification stage.

The second testbed is dedicated to evaluating the ability of our system to detect already existing BLE connections that are less prescribed. Specifically, we establish multiple BLE connections in a controlled environment, using 24 Raspberry Pi 3B and 24 BLE dongles. We repeat the experiment three times with 6, 12, and 24 connections in total. In each experiment all the sessions are created before turning on the sniffing system, which thus has no knowledge of the AAs used. We infer the properties of each connection—AA and CRC—and we count all the sessions detected. Even if we do not show this, it is easy to recover also the hopping sequences thanks to the multi-channel capture.

In Fig. 4 we report statistics about the time needed to discover all the target connections. We see that while this depends on the number of existing connections, even in the most challenging scenario with 24 active sessions our system detects all the connections in less than 200 ms. Another thing to notice is that the distribution gets wider as the number of connections increases. This can be due to the fact that more active sessions have higher chances to collide, therefore generating spurious packets that can be seen as noise and are rejected by our system. Finally, it is interesting to observe that the median of the detection time is almost constant in all scenarios; this means that—irrespective of the actual number of BLE devices—our system can discover all the connections in less than 80 ms more than half of the times.

V. Conclusions

In this paper we presented a novel "software-defined" BLE sniffer that achieves full-band packet capture capability, using SDRs as the radio front-end and harnessing the computation power of a GPU for the processing back-end. We discussed the functionality of the system and showed that it can discover and track multiple connections at the same time on all BLE channels. Our work empirically proves that it is possible to develop applications with high bandwidth in which radio and decoding functionalities are fully implemented in software.

We believe that the benefits of our implementation are twofold. On the one hand, our system lowers considerably the complexity barrier for mounting efficient passive attacks against wireless protocols that use frequency-hopping techniques; on the other hand, we give a practical example on how SDRs can overcome limitations of traditional radio systems in debugging wireless standards.

References

[1] A. Nordrum, "Popular Internet of Things Forecast of 50 Billion Devices by 2020 Is Outdated," Accessed on: Mar. 3, 2020. [Online]. Available: https://spectrum.ieee.org/tech-talk/telecom/internet/popular-internet-of-things-forecast-of-50-billion-devices-by-2020-is-outdated/.

[2] Bluetooth SIG, "2019 Bluetooth Market Update," Accessed on: Mar. 3, 2020. [Online]. Available: https://www.bluetooth.com/bluetooth-resources/2019-bluetooth-market-update/.

[3] M. Cominelli, F. Gringoli, M. Lind, P. Patras, and G. Noubir, "Even black cats cannot stay hidden in the dark: Full-band de-anonymization of bluetooth classic devices," in *IEEE Symposium on Security and Privacy (S&P)*, May 2020, pp. 1631–1645.

[4] S. Sarkar, J. Liu, and E. Jovanov, "A robust algorithm for sniffing ble long-lived connections in real-time," in *2019 IEEE Global Communications Conference (GLOBECOM)*, Dec 2019, pp. 1–6.

[5] "BtleJack: a new Bluetooth Low Energy swiss-army knife," https://github.com/virtualabs/btlejack.

[6] "Bluefruit LE Sniffer - Bluetooth Low Energy (BLE 4.0) - nRF51822," https://www.adafruit.com/product/2269.

[7] F. Gringoli, N. Ali, F. Guerrini, and P. Patras, "A Flexible Framework for Debugging IoT Wireless Applications," in *2018 Workshop on Metrology for Industry 4.0 and IoT*, April 2018, pp. 230–235.

[8] "Ellisys bluetootoh analyzers comparison chart," https://www.ellisys.com/products/btcompare.php.

[9] S. C. Kim, W. L. Plishker, and S. S. Bhattacharyya, "An efficient GPU implementation of an arbitrary resampling polyphase channelizer," in *2013 Conference on Design and Architectures for Signal and Image Processing*, Oct 2013, pp. 231–238.