Multi-Protocol IoT Network Reconnaissance

Stefan Gvozdenovic*, Johannes K Becker†, John Mikulskis‡ and David Starobinski§
Department of Electrical and Computer Engineering, Boston University
Boston, MA 02215
Email: *tesla@bu.edu, †jkbecker@bu.edu, ‡jkulskis@bu.edu, §staro@bu.edu

***Abstract*—Network reconnaissance is a core security functionality, which can be used to detect hidden unauthorized devices or to identify missing devices. Currently, there is a lack of network reconnaissance tools capable of discovering Internet of Things (IoT) devices across multiple protocols. To bridge this gap, we introduce `IoT-Scan`, an extensible IoT network reconnaissance tool. `IoT-Scan` is based on software-defined radio (SDR) technology, which allows for a flexible implementation of radio protocols. We propose passive, active, multi-channel, and multi-protocol scanning algorithms to speed up the discovery of devices with `IoT-Scan`. We implement the scanning algorithms and compare their performance with four popular IoT protocols: Zigbee, Bluetooth LE, Z-Wave, and LoRa. Through experiments with dozens of IoT devices, we demonstrate that our implementation experiences minimal packet losses, and achieves performance near a theoretical benchmark.**

I. Introduction

The Internet of Things (IoT) device market is currently exhibiting exponential growth (among the 29 billion connected devices forecast this year, 18 billion will be related to IoT [1]). These devices run a variety of low-power wireless communication protocols, such as Bluetooth Low Energy (BLE) [2], Zigbee [3], Z-Wave [4], and LoRa [5]. which support applications in smart homes, assisted living, smart grid, health care, and environmental monitoring.

The heterogeneity of the IoT ecosystem – and in particular the large number of IoT protocols – represents a major challenge from a network security monitoring perspective [6], [7]. This heterogeneity makes it hard for network administrators to run network reconnaissance tasks, which aim at discovering wireless IoT devices and their properties. Since many IoT devices are mobile (e.g., wearables and trackers), network reconnaissance tasks must be run regularly. New laws adopted by regulators, such as the IoT Cybersecurity Improvement Act of 2020 [8] in the US, and the upcoming legislation in EU [9], provide further impetus to the design of effective solutions for IoT network reconnaissance.

The simplest solution for IoT network reconnaissance is to use a monitoring device equipped with a different network card for each protocol. However, even devices operating on the same protocol may be incompatible if they run different versions of the protocol (e.g., normal versus long-range Z-Wave). Using dozens of different USB dongles or network cards for each protocol is prohibitive for practical network security auditing. Furthermore, some protocols, like LoRa, encode the network ID at the PHY layer. In such cases, common network cards cannot detect devices belonging to other networks even though they run the same protocol.

Existing software tools for network reconnaissance, such as Nmap [10], focus on devices with IP addresses. Nmap can scan IP/port ranges for an arbitrary number of local or remote hosts and their services. However, this approach is limited to IP-enabled devices only, and yet many popular IoT protocols, including BLE, Zigbee, Z-Wave, and LoRa do not support IP addressing. While there is currently an effort by several vendors to create a unified, IP-based IoT protocol, called Matter [11], its level of adoption and backward-compatibility with legacy devices remain uncertain.

To address this current gap, we propose `IoT-Scan`, an extensible, multi-protocol IoT network reconnaissance tool for enumerating IoT devices. `IoT-Scan` runs both on the 900 MHz and 2.4 GHz bands and currently supports four popular IoT protocols: Zigbee, BLE, LoRa, and Z-Wave. Remarkably, `IoT-Scan` runs on a single piece of hardware, namely a software-defined radio (SDR) [12].

`IoT-Scan` leverages software-defined implementation of IoT communication protocol stacks, mostly under the GNU Radio ecosystem [13], [14]. This approach reduces the amount of hardware needed to address the growing number of IoT protocols. This further allows for future expansion into new protocol versions, thus eliminating the need for purchasing or upgrading protocol-specific hardware [15].

Our main contributions are thus as follows:

- We introduce `IoT-Scan`, a universal tool for IoT network reconnaissance. `IoT-Scan` consists both of a collection of efficient and practical IoT scanning algorithms and of their implementations using a single commercial off-the-shelf software-defined radio device, namely a USRP B200 SDR [16].
- We validate the performance of the algorithms through extensive experiments on a large collection of devices. We demonstrate multi-protocol, multi-channel scanning both on the 2.4 GHz band for Zigbee and BLE, and on the 900 MHz band for LoRa and Z-Wave. Our implementation allows to promiscuously listen to network traffic, even when the network ID is encoded at the PHY layer.
- We propose new active scanning algorithms and show an implementation for Zigbee, which cuts down the discovery time by 87% (from 365 seconds to 46 seconds) compared to a sequential passive scanning algorithm.
- We evaluate the efficiency of the scanning algorithm implementation on the SDR through a theoretical benchmark based on the non-uniform coupon collector prob-

lem [17], [18]. We show that passive scan algorithms for Zigbee and BLE perform near that benchmark.

Threat model. The purpose of `IoT-Scan` is to enumerate IoT devices and their properties at a given location (e.g., an office, a hospital room, etc.). This can be used to detect hidden unauthorized devices, some of which may have been intentionally planted by an adversary for malicious purposes (e.g., eavesdropping). We assume that these devices transmit packets, such as beacons, and/or respond to queries according to their respective wireless protocols. Note that it is hard to detect devices that do not transmit at all. `IoT-Scan` can also be used to identify missing devices which may have been deactivated or stolen by a malicious party (these devices would appear in scans up to some point, but disappear afterward).

The rest of this paper is structured as follows. Section II discusses related work. Section III presents the scanning methods and algorithms forming the core of `IoT-Scan`. Section IV discusses performance metrics for the algorithms, as well as a theoretical model for benchmarking device discovery. Section V elaborates on how `IoT-Scan` discovers addresses of devices in each case. Section VI presents our experiments, including implementation aspects, experimental setup, and results. Section VII concludes our findings, discusses ethical issues, and presents an outlook on future work.

II. Related Work

This section presents related work. Most existing work focuses on *protocol-specific* techniques. In contrast our work introduces several *cross-protocol* algorithms for IoT scanning, and further benchmarks their performance both theoretically and experimentally.

Heinrich et al. presents BTLEmap [19], a BLE-focused device enumeration and service discovery tool inspired by traditional network scanning tools like Nmap [10]. While BTLEMap supports both Apple's Core Bluetooth protocol stack and external scanner sources, it is limited to Bluetooth LE by design and does not aim to support multiple protocols. In contrast, `IoT-Scan` is not tied to a particular vendor as a host device, and supports multiple protocols simultaneously, with one radio source.

Tournier et al. propose IoTMap [20], which models interconnected IoT networks using various protocols, and deduces network characteristics on multiple layers of the respective protocol stacks. IoTMap requires dedicated radios for each protocol in order to operate, whereas `IoT-Scan` achieves device detection across multiple protocols with a single software-defined radio transceiver.

In prior work, Mikulskis et al. presented Snout [21] and showcased scanning of BLE and Zigbee devices under a common SDR platform. `IoT-Scan` encompasses additional protocols, namely LoRa and Z-Wave. Furthermore, our work introduces novel scanning algorithms and conducts extensive evaluation of these algorithms, both theoretically and empirically with dozens of IoT devices. In contrast, the work in [21] did not present scanning algorithms and had no evaluation contents (either theoretical or empirical).

Bak et al. [22] optimize BLE advertising scan (i.e., device discovery) by using three identical BLE dongles. This approach is not scalable since it requires a new hardware receiver for each new channel, and equally does not scale beyond the BLE protocol. In contrast, our SDR-based approach uses the same SDR hardware to receive multiple protocols.

Kilgour [23] presents a multi-channel BLE capture and analysis tool implemented on a field programmable gate array (FPGA). This multi-channel BLE tool allows receiving data from multiple channels in parallel. However, the focus is on BLE PHY receiver implementation and related signal processing rather than actual scanning and enumeration of devices. In contrast, our work extends beyond Bluetooth LE, and crucially performs practical device enumeration scans to quantify scanning performance.

Park et al. describe a Wi-Fi active scan technique performed using BLE radio using cross-protocol interference [24]. The active scan algorithms in `IoT-Scan` are motivated by similar ideas, but require judicious use of protocol-specific mechanisms (i.e., sending beacon request packets in Zigbee).

Hall et al. [25] describe a tool, called EZ-Wave that can discover Z-Wave devices passively and actively. The EZ-Wave tool actively scans a Z-Wave device by sending a "probe" packet with acknowledgement request flag set. In the older version S0 of the Z-Wave protocol, it was compulsory for a Z-Wave device to reply with acknowledgements to such packets. By getting this acknowledgement back, the EZ-Wave tool learns about a device's presence. However, the EZ-Wave tool only supports older versions of Z-Wave protocol. In the new version (S2) of the Z-Wave protocol, acknowledgements are not compulsory and this is not a reliable active scan mechanism. The old Z-Wave protocol uses only the R1 (9.6 kbps) and R2 (40 kbps) physical layers. Our work adds R3 (100 kbps PHY) as well as multi-protocol capabilities. The R1, R2, and R3 rates are defined in [4, Table 7-2].

Choong [26] implements a multi-channel IEEE 802.15.4 receiver using a USRP2 software-defined radio. Choong describes a channelization method similar to the receive chain used in this work (see Section 2) that extracts multiple channels from a wider raw signal stream. However, Choong's work focuses on the performance impact of the SDR host computer, and is a Zigbee-specific implementation, whereas our work focuses on device enumeration in a multi-channel as well as multi-protocol context.

Our Zigbee, BLE, and Z-Wave GNU Radio receiver implementations are based on scapy-radio [14] flowgraphs. Our LoRa GNU Radio receiver flowgraph is based on a work by Tapparel et al. [5]. A similar multi-channel LoRa receiver was implemented by Robyns in [27]. In order to support multi-radio, multi-channel capabilities, `IoT-Scan` implements several changes to these GNU Radio receiver implementations. In general, these changes pertain to the signal path between the SDR source and the receive chains for individual channels and protocols (i.e., frequency translation, filtering, and resampling, see Section VI-A). Additionally, our LoRa receiver can listen to LoRa packets promiscuously.

Algorithm 1: Passive_Scan(ch_list, $dwell\_time$, $scan\_time$)

▷ Enumerate devices by repeatedly listening for duration *dwell_time* on each channel in *ch_list* and stop after *scan_time*

1 $t_{start} \leftarrow$time() ▷ Store current time
2 $device\_list \leftarrow \{\}$ ▷ Initialize device list
3 $i \leftarrow 0$ ▷ Set channel counter to zero
4 **while** time()$-t_{start} \leq scan\_time$ **do**
  ▷ $ch\_list(i)$ is the i-th element in *ch_list*
5 $new\_dev \leftarrow$ **Listen**$(ch\_list(i), dwell\_time)$
6 $device\_list = device\_list \cup new\_dev$
7 $i \leftarrow (i+1)$ mod $|ch\_list|$
8 **end while**
9 **return** $device\_list$

III. Scanning Algorithms

In this section, we introduce SDR-based scanning algorithms that form the core of `IoT-Scan`. The notion of *channel* in this section refers to a 3-tuple containing the center frequency of the channel, the channel bandwidth (i.e., a range of frequencies delineated by the lower and upper frequencies of the channel), and the protocol type. The concept of *instantaneous bandwidth* refers to the range of frequencies captured by the SDR at any given point of time. The *center frequency* corresponds to the frequency at the middle of the range.

A. Single-channel methods

The key building block to any of the following scanning algorithms is the function **Listen**. This function takes two input parameters, namely a channel ch (defined by a center frequency, bandwidth and protocol) and a time period $dwell\_time$ after which the procedure terminates listening to channel ch. During execution of this procedure, the SDR decodes any packet received on the channel, and extracts address information that identifies a device. Upon the expiration of the channel dwelling time, the procedure returns the list of discovered devices.

Algorithm 1 presents a simple sequential scanning procedure **Passive_Scan** that can be used in conjunction with any IoT protocol. This algorithm represents a baseline against which the performance of more advanced algorithms can be compared. The algorithm invokes the **Listen** procedure in a round-robin fashion on each channel of a given channel list ch_list, which is provided as an input to the procedure. The total scan time is set by the $scan\_time$ input parameter. Note that generally $scan\_time \gg dwell\_time$, and hence each channel is visited several times during the scan. The algorithm returns the list of discovered devices.

Sequential passive scanning can be slow, especially if an IoT protocol supports many channels, but only a few channels are used. To speed up device discovery, Algorithm 2, referred to as **Active_Scan**, implements a two-phase approach. During the first phase (line 4), it invokes a helper function **Probe_Channels**, which sends a probe packet on each channel ch in the provided $channel\_list$ and waits for a response. If one or more devices respond, then channel ch is added to the $active\_channels$ list. During the second phase (lines 5–7), Algorithm 2 performs passive scanning only on channels appearing in the $active\_channels$ list for the remaining scan time. Algorithm 2 is especially useful for protocols such as Zigbee, which defines 16 different channels, not all of which may be in use (see also Section V).

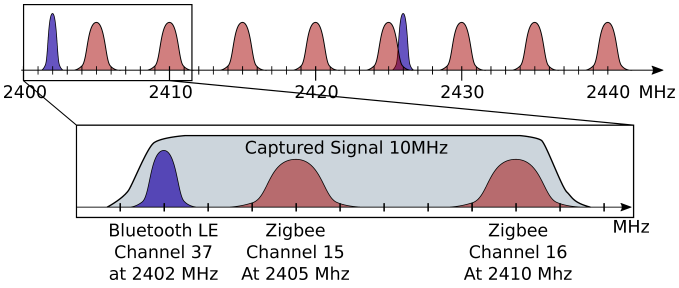

Figure 1: **Find_Channels_In_Range** starts at the lowest channel in the provided list and returns all channels that are in range of the SDR hardware based on the provided instantaneous bandwidth parameter.

B. Multi-channel methods

The subsequent algorithms expand from single channel scanning to handling multiple channels and multiple protocols, at the same time. First, we need a method of grouping channels within the range of the instantaneous bandwidth of the SDR. The function **Find_Channels_In_Range** identifies all channels in an input channel list (ordered by ascending frequency) by selecting all channels that fit the instantaneous bandwidth under consideration of their respective center frequencies and channel bandwidths, see Fig. 1. Note that while some channels overlap, transmissions on these channels do not occur continuously. Hence, it is possible to decode packets if they do not collide, which is usually the case.

We further define a helper function **Listen_In_Parallel** which simultaneously listens to multiple channels by calling **Listen** on all provided channels. Implementing this algorithm requires extracting multiple signal streams by frequency-shifting, filtering, and resampling the incoming signal. This procedure is called *channelization*. The implementation aspects of this procedure are described in Section VI-A.

Multiprotocol_Scan (Algorithm 3) describes a parallel multi-protocol scan that can be used with any number of IoT protocols. Based on a list of channels to consider (ordered by ascending frequencies), the algorithm starts at the lowest frequency and determines all channels within range of the first channel by calling **Find_Channels_In_Range**. It subsequently listens to those channels by invoking **Listen_In_Parallel**.

Active_Multiprotocol_Scan combines the aforementioned active scanning and multi-protocol scanning capabilities. It is useful for scanning multiple protocols, some actively and some passively (such as a combination of Zigbee and BLE).

Algorithm 2: Active_Scan($ch\_list, dwell\_time, scan\_time$)

▷ Enumerate devices by first identifying the list of *active_channels* in *ch_list* and then performing passive scanning only on those *active_channels*

$device\_list \leftarrow \{\}$ ▷ Initialize list of found devices
$active\_channels \leftarrow \{\}$ ▷ Initialize list of busy channels
$t_{start} \leftarrow$time() ▷ store current time
▷ Phase 1: Scan active devices
$active\_channels, device\_list \leftarrow$ **Probe_Channels**$(ch\_list, dwell\_time)$
▷ Phase 2: Passive-scan known active channels for the remaining time
$t_{scan} \leftarrow scan\_time - (\text{time}() - t_{start})$ ▷ Compute remaining scanning time
$new\_dev \leftarrow$ **Passive_Scan**$(active\_channels, dwell\_time, t_{scan})$ ▷ Run passive scan on active channels
$device\_list \leftarrow device\_list \cup new\_dev$ ▷ Add devices found during passive scanning
return $device\_list$

Algorithm 3: Multiprotocol_Scan($ch\_list, dwell\_time, scan\_time, bandwidth$)

▷ This algorithm enumerates devices by scanning as many channels as can fit in the instantaneous bandwidth of *bandwidth* for a duration *dwell_time* in each iteration.

$ch\_unscanned \leftarrow ch\_list$ ▷ All channels in the list are unscanned
$ch\_groups \leftarrow \{\}$ ▷ Initialize list of channel groups
while $ch\_unscanned \neq \{\}$ **do**
  ▷ Find channels that can be scanned simultaneously as they fit the instantaneous *bandwidth*.
  $ch\_range \leftarrow$ **Find_Channels_In_Range**(ch_unscanned, bandwidth)
  ▷ Scan channels that fit in instantaneous bandwidth BW around center freq.
  $ch\_groups \leftarrow ch\_groups \cup \{ch\_range\}$ ▷ Add this group to the list of channel groups
  $ch\_unscanned \leftarrow ch\_unscanned \setminus ch\_range$ ▷ Remove channels from unscanned list
end while
$t_{start} \leftarrow$time() ▷ Store current time
$device\_list \leftarrow \{\}$ ▷ Initialize list of found devices
$i \leftarrow 0$ ▷ Set channel counter to zero
while time()$-t_{start} \leq scan\_time$ **do**
  ▷ Scan all channels of the i'th channel group in parallel
  $new\_dev \leftarrow$ **Listen_In_Parallel**(ch_group(i), dwell_time)
  ▷ Remove scanned channels from unscanned channel list
  $device\_list \leftarrow device\_list \cup new\_dev$ $i \leftarrow (i+1) \bmod |ch\_groups|$
end while
return $device\_list$

IV. Performance Metrics and Analysis

A. Metrics

Our main metric is the discovery time of IoT devices, which we aim to minimize. Assume there are N devices in total, with corresponding discovery times $T_1, T_2, \ldots, T_N$. We are interested in characterizing the *order statistics* of these random variables, i.e., the time elapsing till one device is discovered, which is denoted $X_{1:N}$, then till two devices are discovered which is denoted $X_{2:N}$, and so on till all devices are discovered, which is denoted $X_{N:N}$. We thus have

$$X_{1:N} = \min(T_1, T_2, \ldots, T_N), \quad (1)$$
$$X_{2:N} = \min(\{T_1, T_2, \ldots, T_N\} \setminus X_{1:N}), \quad (2)$$
$$\ldots$$
$$X_{N:N} = \max(T_1, T_2, \ldots, T_N). \quad (3)$$

In our experiments, we estimate the expectation of the n-th order statistics $E[X_{n:N}]$, for $n = 1, 2, \ldots, N$. To obtain these estimates, we run each scanning algorithm M times and denote by $x_{n:N}^{(m)}$ the time till n devices are discovered at the m-th iteration, where $m = 1, 2, \ldots, M$. We then compute the *sample mean* for the n-th order statistics as follows:

$$\bar{x}_{n:N} = \frac{\sum_{m=1}^{M} x_{n:N}^{(m)}}{M}. \quad (4)$$

We also provide $(1-\alpha)100\%$ *confidence intervals* for our estimates

$$[\bar{x}_{n:N} - e_{n:N}, \bar{x}_{n:N} + e_{n:N}], \quad (5)$$

based on computing the sample standard deviation $s_{n:N}$ and

the confidence interval parameter $e_{n:N}$ as follows:

$$s_{n:N} = \sqrt{\frac{1}{M-1}\sum_{m=1}^{M}(x_{n:N}^{(m)} - \bar{x}_{n:N})^2}, \quad (6)$$

$$e_{n:N} = t_{\alpha/2,M-1} \times \frac{s_{n:N}}{\sqrt{M}}, \quad (7)$$

with $t_{\alpha/2,M-1}$ denoting the $1-\alpha/2$ quantile of the t-distribution with $M-1$ degrees of freedom [28]. In our experiments, described in Section VI, we run $M = 10$ independent iterations for each algorithm and consider 95% confidence intervals (i.e., $\alpha = 0.05$), hence $t_{\alpha/2,M-1} = 2.262$.

B. Theoretical Model

We next propose a theoretical model to estimate the expectations of order statistics of the discovery time, under appropriate statistical assumption. The analysis further assumes an idealized channel environment where no packet loss occurs (in practice such losses could occur due to imperfect receiver implementation or interference). In Section VI-C, we show that the performance of the scanning algorithms approaches that predicted by the theoretical model, which demonstrates the efficiency of the algorithms.

1) Statistical assumptions: To model device enumeration, we need statistics of the inter-arrival times of packets generated by each device. For the sake of *analytical tractability*, we assume that devices transmit in a memoryless fashion, i.e., the inter-arrival times of their packets follow an exponential distribution. Note that the mean and standard deviation of an exponential random variable are equal. Hence, we expect that this model can provide a reasonable approximation, if for each device i, its mean inter-arrival time μ_i and standard deviation of inter-arrival times σ_i are roughly equal. We stress that this assumption is not needed for the implementation of the scanning algorithms, only for their analysis.

To check this assumption, we collected statistics of the inter-arrival times of packets of the Bluetooth and Zigbee devices listed in Table I below. Table I indicates that indeed for all tested BLE devices and most Zigbee devices $\mu_i \approx \sigma_i$.

2) Analysis of order statistics: Enumerating devices shares similarities with the non-uniform coupon collector's problem [17], albeit with certain modifications. The coupon collector's problem assumes a probability distribution in which each draw results in a coupon (i.e., a discovered device). This cannot be applied directly to a scenario in which devices' transmission characteristics may result in *null coupons*, i.e., a scan iteration in which no new device is discovered. Anceaume et al. [18] provide a method of calculating the expectation of the non-uniform coupon collector problem which accounts for a null coupon. Define the probability vector $\boldsymbol{p}$ in which p_0 is the probability of no device transmitting, and p_i is the probability of device i transmitting, $i = 1, 2, \ldots, N$. The expectation for the n-th order statistics $X_{n,N}$ (i.e., the time to to discover n out of N devices) is then given by

$$\mathbb{E}[X_{n:N}(\boldsymbol{p})] = \sum_{h=0}^{n-1} R_{N,n,h} \sum_{J \in S_{h,N}} \frac{1}{1-p_0-P_J} \quad (8)$$

where

$$R_{N,n,h} = (-1)^{n-1-h}\binom{N-h-1}{N-n}. \quad (9)$$

Here, $S_{h,N}$ denotes all $\binom{N}{h}$ subsets containing exactly h devices. Denote by J any subset of $S_{h,N}$ that contains exactly h devices. Then, $P_J = \sum_{j \in J} p_j$ is the summation of the transmission probabilities of all devices belonging to J. Note that the second summation term in Eq. (8) works out to a summation over all possible subsets J of cardinality h.

Assuming all N devices send packets in an independent and identically distributed memoryless fashion as discussed above, the device traffic can be modeled as N independent Poisson processes with rate $\lambda_i = 1/\mu_i$. The combined influx of packets from all the devices then follows a Poisson process with rate $\lambda = \sum_{i=1}^{N} \lambda_i$. By selecting a small interval Δt such that either zero or one packet arrives during any interval Δt, we can use Eq. (8) to compute the expectation of the order statistics of the discovery time of devices. If all devices transmit on one channel that is continuously monitored, the probability p_i that device i transmits during an interval Δt is then

$$p_i = (\lambda_i \Delta t)e^{-\lambda_i \Delta t} \approx \lambda_i \Delta t. \quad (10)$$

Note that if all devices are randomly distributed on any of C available channels, a randomly channel-hopping radio scanner would receive a transmission from device i with probability p_i/C. This can also be used as an approximation when the scanner visits channels in a round-robin rather than in a random fashion.

V. Protocol Device Enumeration

In the previous sections, the concepts of "listening to a channel" and "extracting device addresses" were presented in a generic way. We now discuss these aspects for all the IoT protocols implemented in `IoT-Scan`.

A. Zigbee

`IoT-Scan` implements both passive and active scans for Zigbee. A passive scan listens on each channel for a certain amount of time (i.e., the *channel dwell time*) repeatedly until the total scan time expires. With active scanning, channels with network activity are discovered by sending beacon requests on each channel. Receiving a beacon frame in response to a beacon request indicates that there is a network on the current channel (typically, a Zigbee network has a router or a coordinator that responds to beacon requests). Subsequent passive scanning rounds can then be limited to these active channels (line 6 in Algorithm 2), in order to detect any further devices that did not respond to active scanning.

B. Bluetooth Low Energy (BLE)

In BLE, data channels are used for communication after a connection has been established, whereas advertising channels

are used between devices that are in range to discover one another and exchange metadata. Therefore, IoT-Scan only scans the three advertising channels (i.e., there is no need to monitor data channels). Typically, advertising packets are sent on all three advertising channels for any given advertising event. This redundancy makes device discovery more resilient in cases where some of the channels experience interference. This means that scanning for BLE devices on any one of the three advertising channels is as good as a multi-channel scan (sequentially scanning each advertising channel), a fact that we also verified experimentally.

C. LoRa

In our implementation, we scan Yolink devices listed in Table I. The major challenge in receiving any Yolink traffic is in determining the PHY-layer network sync word, because the existing SDR LoRa receiver implementation [5] accepts only sync words with a value of zero. Yet, sync word values containing 0x00 are forbidden in deployed networks and can only be used for testing. We overcome this challenge by modifying the LoRa receiver of [5]. Our implementation allows one to promiscuously listen for all sync words, as well as configure the bandwidth, the center frequency, the bitrate, and other parameters. A key advantage of scanning LoRa using an SDR implementation is that all sync words can be monitored simultaneously, whereas certified LoRa transceiver chips are programmed to receive a specific sync word.

D. Z-Wave

IoT-Scan uses the Source ID [29] in the MAC header to enumerate Z-Wave devices. R1 and R2 Z-Wave PHY implementations are based on scapy-radio [14]. We built the R3 Z-Wave PHY receiver flowgraph based on the existing R2 PHY implementation, the main difference being the bitrate/sampling which is 2.5 times larger.

VI. Experimental Evaluation

In this section, we perform an experimental evaluation of the scanning algorithms of IoT-Scan. We detail SDR implementation aspects, the experimental set-up (including the list of tested devices), and the experimental results.

A. Algorithm Implementation

The main software components of our implementation consist of GNU Radio 3.8 [13] and Scapy-radio 2.4.5 [14]. Scapy-radio is a pentest tool with RF fuzzing capabilities. Note that Scapy-radio is based on GNU Radio version 3.7. We ported receiver flowgraphs to GNU Radio 3.8 gaining great reception improvements due to the automatic gain control (AGC) option inside the USRP Source block.

1) Flowgraph control: We implement the scanning algorithms described in Section III in Python. Signal processing parameters, such as the SDR center frequency, the channel frequency offsets, and the channel bandwidths, are managed by a GNU Radio flowgraph. The GNU Radio flowgraph is imported from the main application as a Python module and is controlled with its native Python API. This allows for dynamic control of flowgraph parameters during the runtime of the flowgraph. Controlling the flowgraph in this way is crucial for correct time-keeping of the experiments, as it allows to compensate for seconds of startup delays due to the initialization of the USRP hardware driver library.

2) Signal processing: The process of converting an unfiltered full-bandwidth signal from an SDR source into the receive chain (i.e., the sequence of DSP blocks connected serially starting with radio source, demodulator, filter, and clock recovery) of a particular protocol is referred to as *channelization* [30]. Channelization is particularly important in multi-protocol scanning, since it selects (filters out) a few narrow band signals (receive chains) from the raw wide band signal. Multi-protocol scans require parallel decoding of two or more receive chains which can overwhelm the capabilities of a typical host computer if the processing chain in the flowgraph is not correctly optimized.

Channelization in IoT-Scan comprises three signal processing steps: frequency translation (from the center frequency of the raw radio signal to the center frequency of the desired channel), channel filtering (filtering out other protocols and potential interference), and re-sampling (down conversion) to reduce the computational load. Reducing the sample rate relies on Nyquist's theorem, which dictates that the sample rate of a signal be at least twice the signal's bandwidth, in order to not lose any information.

B. Experimental Setup

We implemented all the scanning algorithms described in Section III on a single SDR device, namely a USRP B200 device [16], with a PC capable enough to handle data processing in real-time without dropping samples (i.e., overflowing buffers). Thus, all our experiments were run on a ThinkCentre 8 Core Intel i7 running Ubuntu 20.04.

The devices used in the experiments are listed in Table I. All scanning experiments were based on IoT devices under our control, which were placed in the same office room as the SDR. Any foreign device from the environment was filtered

Table I: Tested IoT devices.

(a) Zigbee Device	μ_i [s]	σ_i [s]
#1 Wink hub 2	6.0	7.7
#2 Amazon Echo 4.0	7.5	6.1
#3 Ring Base Station	7.9	7.5
#4 Philips Hue Lamp1	8.4	7.1
#5 Philips Hue Lamp2	8.3	7.0
#6 Philips Hue Lamp3	8.4	7.0
#7 Philips Hue Lamp4	8.4	7.1
#8 Quirky PLINK-HUB	8.5	4.7
#9 Osram Lightify 73674	10.2	7.8
#10 IKEA Gateway	14.0	14.6
#11 CREE Lightbulb A19	14.8	1.2
#12 GE LAMP1 4VE8	14.9	1.4
#13 GE LAMP2 4VE8	14.9	1.4
#14 IKEA LED1732G11	15.9	1.2

(b) BLE Device	μ_i [s]	σ_i [s]
#15 Fit2 fitness tracker	4.1	4.0
#4 Philips Hue Lamp1	4.6	5.6
#5 Philips Hue Lamp2	4.1	4.0
#6 Philips Hue Lamp3	3.9	3.8
#7 Philips Hue Lamp4	4.1	4.1
#16 Mi Smart Band 5	11.0	10.4
#17 AMIR Thermometer	19.5	18.6
#18 Tile tracker	25.7	25.2
#19 Tile tracker	47.3	42.0
#20 Tile tracker	23.1	22.7
#21 Tile tracker	20.7	18.5
#22 WIT Motion sensor	119.5	96.1

(c) LoRa (Yolink)
#26 Water Leak Sensor
#27 Door Sensor
#38 Smart Plug

(d) Z-Wave (Ring)
#3 Base Station
#23 Keypad
#24 Contact Sensor
#25 Motion Detector

out. In order to only account for our devices, we initially enumerated them with a passive scan inside an RF shielded box (Ramsey box STE3500) to determine their addresses. Traffic of the BLE and Zigbee devices were statistically analyzed to derive the parameters of the theoretical traffic model introduced in Section IV-B. Note that we did not analyze transmission statistics of low-power Z-Wave and LoRa devices due to their periodic transmission patterns (devices transmit once every hour or so).

We conducted all scanning experiments using the default network configuration of the respective devices and protocols. In all experiments, the tested devices were in an idle state, i.e., not actively used by an operator. Manually operating devices in a way that generates network communication, e.g., actuating Zigbee lights via the Amazon Alexa smartphone app, would impact scanning performance. We expect the results presented in this section to be conservative estimates of the scanning time, since generating additional traffic from the devices should speed up the discovery of the devices.

Regarding the parameters of the algorithms, the channel dwell time (i.e., the scanning time of each channel in each round) was set to 1 second. We also tried channel dwell times of 0.1 second and 3 seconds, and found that the scanning times did not differ significantly. The channel dwell time during active scan of Zigbee was set to 0.2 second. When scanning each individual protocol, we set the instantaneous bandwidth parameter according to the protocol's bitrate. Specifically, BLE's channel bandwidth was set to 1 MHz, Zigbee to 2 MHz, LoRa to 125 KHz, and Z-Wave to 40/100 KHz. When implementing multiprotocol scanning algorithms, we used wider bandwidth to fit the bandwidth of each protocol and channel spacing in between. Both the Zigbee/BLE and Z-Wave/LoRa and multi-protocols experiment used 8 MHz of bandwidth.

C. Results

In this section, we discuss experimental results of the scanning algorithms. The figures show the sample means and 95% confidence intervals of the order statistics of the discovery time of the n-th device (see Eqs. (4) and (5)). Each point represents an average over 10 experiments with identical parameters.

1) Passive Zigbee and BLE Scans and Comparison with Theoretical Model: We first evaluate the performance of the passive scanning algorithms (Algorithm 1) for Zigbee and BLE devices, and compare those with the expected discovery times based on the theoretical model described in Section IV-B.

To build the theoretical traffic model (see Section IV-B), we measured device characteristics of our tested devices by running one long continuous scan of 100 minutes on every Zigbee channel and on every BLE advertising channel, in order to collect a baseline of traffic for each device. The traffic statistics are shown in Table I. We set $\Delta t = 0.1$s in Eq. (10) to compute p_i for each device. We then use Eq. (8) to compute the expectation of the order statistics of the discovery time of devices. Note that for Zigbee, we replace p_i by $p_i/16$, since with Algorithm 1, the SDR listens to only one out of the 16 Zigbee channels at a time.

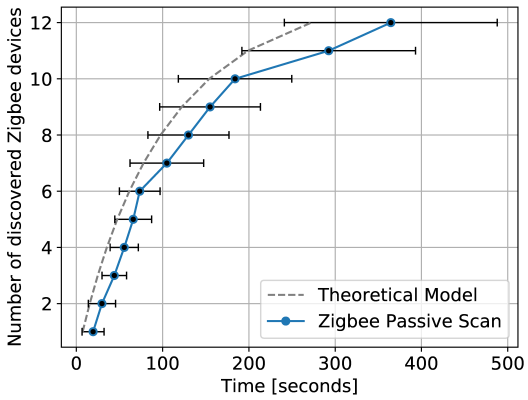

Figure 2: Zigbee theoretical model and experimental passive scan results. The 95% confidence intervals indicate a good fit.

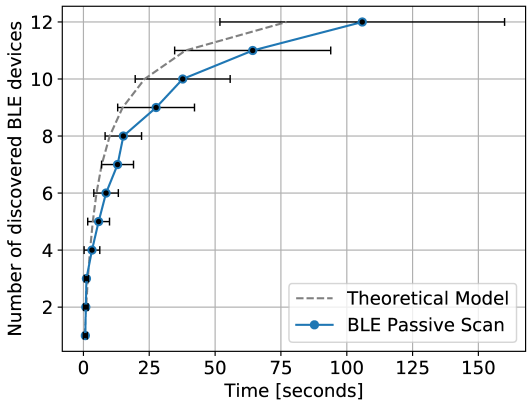

Figure 3: BLE passive scan results align closely with the theoretical model.

Fig. 2 shows curves for the experimental results of Zigbee passive scanning and the theoretical model. The model fits inside most of the 95% confidence intervals. This shows that our passive scan implementation is close to the best performance possible, and our testbed has minimal packet losses. The deviation from the model could be attributed to interference (e.g., from Wi-Fi) and the fact that transmissions of some Zigbee devices are not memoryless.

Fig. 3 shows experimental results for BLE passive scanning and the theoretical benchmark. The measured discovery times again fit the model well. Since all BLE advertising channels are equivalent, scanning is performed on channel 37 only. Note that BLE device discovery can only be performed as a passive scan, since BLE does not allow for broadcast-type scan requests as performed in Zigbee. While BLE scan requests could be a useful active scanning technique for gathering additional device data, they are always directed scans, i.e., they require knowledge of the target device's address.

2) Active Zigbee Scan: We next evaluate the performance of active scanning (Algorithm 2) and compare it to passive scanning in the context of Zigbee. Fig. 4 shows that the passive discovery of 12 Zigbee devices takes 365 seconds on average while active Zigbee discovery takes only 46 seconds, i.e., a reduction of 87% in the scan time. While active scanning

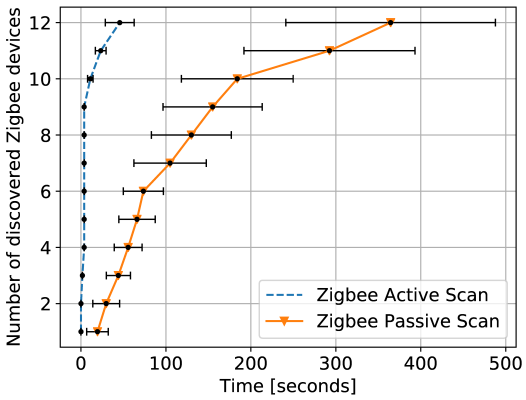

Figure 4: Zigbee active versus passive scan.

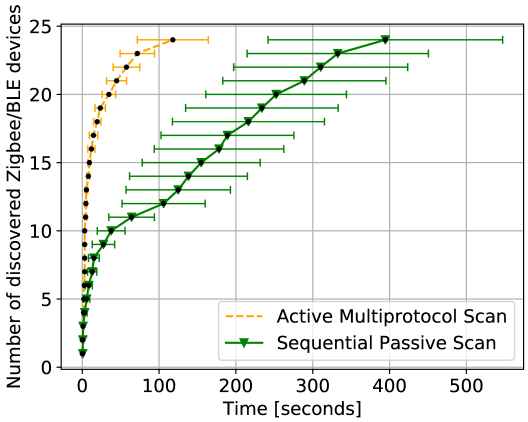

Figure 5: Zigbee/BLE multiprotocol active scan vs. sequential passive scan.

discovers the 12 devices within one minute, passive scanning discovers only 4 devices within one minute. Note that Zigbee supports up to 64,000 nodes per network. It is conceivable that the improvement of active scan over passive scan would be even more significant with a larger number of nodes.

3) Zigbee and BLE Multiprotocol Scan: We next evaluate the performance of active multiprotocol Zigbee and BLE scan and compare it to sequential passive scan. Sequential passive scan consists of passive BLE scan followed by passive Zigbee scan. Sequential passive scan enumerates the 24 considered devices in 395 seconds on average, while active multiprotocol Zigbee and BLE scan takes 118 seconds on average, which corresponds to a 70% improvement (Fig. 5). Within 1 minute active multiprotocol scan discovers 22 devices while sequential scan discovers only 10. Breaking down sequential passive scan into two: the first 106 seconds corresponds to a BLE passive scan, followed by 289 seconds of Zigbee scan, which is consistent with the results shown in Figs. 2 and 3. The speed-up is achieved because of two aspects: active scan and multiprotocol scan. Zigbee active scan narrows the search down from 16 to only 3 channels. Multiprotocol scan supports reception of one Zigbee and one BLE channel in parallel. Note that parallel reception is possible only if the two channels fit within the instantaneous bandwidth. As mentioned earlier, the instantaneous bandwidth for multiprotocol scan was set to

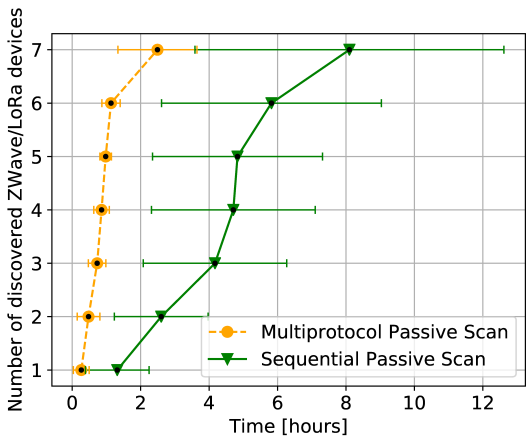

Figure 6: Multiprotocol passive scan (Z-Wave, LoRa)

8 MHz. Three Zigbee active channels were identified, namely channel 11, 15, and 20. BLE has three well-known advertising channels, namely 37, 38, and 39. BLE channel 37 and Zigbee channel 11 can be received in parallel as well as BLE channel 38 and Zigbee channel 15. However, Zigbee channel 20 and BLE channel 39 are scanned separately since they do not fit within the same instantaneous bandwidth.

4) Z-Wave and LoRa Multiprotocol Scan: We next evaluate the performance of passive multiprotocol LoRa and Z-Wave scan on 900 MHz band (Algorithm 3) and compare it to sequential passive scan (Algorithm 1). Passive multiprotocol scan consists of scanning each of 3 frequency channels (2 Z-Wave and 1 LoRa) in a round robin fashion. The passive scanning operation visits the LoRa and Z-Wave channel in a round-robin fashion, one at a time. Due to having 2 Z-Wave channels (908.4 and 916 MHz) and only 1 LoRa channel (910.29 MHz), Z-Wave has an advantage in passive scanning.

Fig. 6 shows that sequential LoRa and Z-Wave scan takes about 8.1 hours on average while multiprotocol Z-Wave and LoRa scan takes 2.5 hours, which represents a reduction of about 70% in the discovery time. Within a single hour passive scan discovers less than 1 device on average while multiprotocol scan discovers 5 out of the 7 devices. This significant speed-up is achieved because multiprotocol scan receives all three channels (from the two protocols) in parallel, namely 908.4 MHz (Z-Wave R2 PHY), 910.23 MHz (LoRa uplink), and 916 MHz (Z-Wave R3 PHY).

VII. Conclusion

We presented `IoT-Scan`, an extensible multi-protocol network reconnaissance tool for the Internet of Things that can be employed for security auditing and network monitoring. `IoT-Scan` leverages the capabilities of SDRs to process multiple streams in parallel. Accordingly, we introduced several scanning algorithms and evaluated them both theoretically and experimentally. Using the theoretical model, we showed that our implementation is efficient and achieves minimal packet loss in reception. We implemented multi-protocol, multi-channel scanning both on the 2.4GHz band for Zigbee and BLE, and on the 900 MHz band for LoRa and Z-Wave, and

demonstrated significant improvement over sequential passive scanning.

Our SDR implementations should prove especially useful in overcoming the incompatibility of different protocols based on the same PHY layer. For instance, besides Zigbee, there exist several IoT protocols based on the IEEE 802.15.4 standard, such as Thread [31] and WirelessHART [32]. We expect that these protocols could readily be integrated into `IoT-Scan`.

The design of `IoT-Scan` does not raise ethical issues in itself. However, like other penetration testing tools, usage of this tool does require explicit consent from the owners of the devices under test. Specifically, active scanning, while brief, may interfere with existing network traffic and delay time-sensitive communication. A major advantage of `IoT-Scan` versus a tool like Nmap is that it also supports a passive scanning mode, which does not generate traffic.

This paper opens several avenues for future work. First, one could explore FPGA implementations of `IoT-Scan` to increase the number of channels and protocols that can be decoded in parallel and further speed up the discovery of IoT devices. While this should yield useful performance improvements, we expect that such implementations would still rely on the algorithms introduced in Section III. Another interesting research avenue lies in the design of active scanning methods for LoRa and Z-Wave, as devices in these protocols transmit sparingly. We have publicly released data traces obtained with `IoT-Scan` in [33]. We envision that these traces should be useful for the design and evaluation of scanning algorithms and other IoT-related research.

Acknowledgements

This research was supported in part by the US National Science Foundation under grants CNS-1717858, CNS-1908087, CCF-2006628, EECS-2128517, and by an Ignition Award from Boston University.

References

[1] Ericsson. (2020) Internet of Things Forecast. [Online]. Available: https://www.ericsson.com/en/mobility-report/internet-of-things-forecast

[2] Bluetooth Special Interest Group (SIG). (2016) Bluetooth Core Specification. v5.0. [Online]. Available: https://www.bluetooth.com/specifications/specs/core-specification/

[3] IEEE Standards Association, *802.15.4-2015 - IEEE Standard for Low-Rate Wireless Networks*, IEEE, Ed., New York, New York, USA, 2015.

[4] International Telecommunication Union. (2015) G.9959: Short range narrow-band digital radiocommunication transceivers - PHY, MAC, SAR and LLC layer specifications. [Online]. Available: https://www.itu.int/rec/T-REC-G.9959

[5] J. Tapparel, O. Afisiadis, P. Mayoraz, A. Balatsoukas-Stimming, and A. Burg, "An Open-Source LoRa Physical Layer Prototype on GNU Radio," in *SPAWC 2020*, 2020, pp. 1–5.

[6] J. Ortiz, C. Crawford, and F. Le, "DeviceMien: network device behavior modeling for identifying unknown IoT devices," in *Proceedings of the International Conference on Internet of Things Design and Implementation*. New York, NY, USA: ACM, Apr, pp. 106–117.

[7] D. Y. Huang, N. Apthorpe, F. Li, G. Acar, and N. Feamster, "IoT Inspector," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 4, no. 2, pp. 1–21, Jun 2020. [Online]. Available: https://dl.acm.org/doi/10.1145/3397333

[8] US Congress, "H.R.1668 - IoT Cybersecurity Improvement Act of 2020," 2020. [Online]. Available: https://www.congress.gov/bill/116th-congress/house-bill/1668

[9] European Commission. (2022) Secure solutions for the Internet of Things. [Online]. Available: https://digital-strategy.ec.europa.eu/en/policies/secure-internet-things

[10] G. F. Lyon, *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*, 2nd ed. Sunnyvale, CA, USA: Insecure.Com LLC, 2008.

[11] CSA. (2021, Dec) Matter: Smart Home Device Solution. [Online]. Available: https://csa-iot.org/all-solutions/matter/

[12] T. Ulversoy, "Software Defined Radio: Challenges and opportunities," *IEEE Communications Surveys & Tutorials*, vol. 12, no. 4, pp. 531–550, 2010.

[13] GNU Radio Project. (2022) GNU Radio. [Online]. Available: https://www.gnuradio.org

[14] Bastille Research. (2015) scapy-radio. [Online]. Available: https://github.com/BastilleResearch/scapy-radio

[15] Y. He, J. Fang, J. Zhang, H. Shen, K. Tan, and Y. Zhang, "MPAP: Virtualization Architecture for Heterogenous Wireless APs," *ACM SIGCOMM Computer Communication Review*, no. 4, pp. 475–476, Aug.

[16] Ettus Research. (2022) USRP B200. [Online]. Available: https://www.ettus.com/all-products/ub200-kit/

[17] P. Flajolet, D. Gardy, and L. Thimonier, "Birthday paradox, coupon collectors, caching algorithms and self-organizing search," *Discrete Applied Mathematics*, vol. 39, no. 3, pp. 207–229, Nov 1992.

[18] E. Anceaume, Y. Busnel, and B. Sericola, "New Results on a Generalized Coupon Collector Problem Using Markov Chains," *Journal of Applied Probability*, vol. 52, no. 2, p. 405–418, 2015.

[19] A. Heinrich, M. Stute, and M. Hollick, "BTLEmap: Nmap for Bluetooth Low Energy," in *Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, ser. WiSec '20. Association for Computing Machinery, 2020, p. 331–333. [Online]. Available: https://doi.org/10.1145/3395351.3401796

[20] J. Tournier, F. Lesueur, F. Le Mouël, L. Guyon, and H. Ben-Hassine, "IoTMap: A protocol-agnostic multi-layer system to detect application patterns in IoT networks," in *10th International Conference on the Internet of Things (IoT 2020)*, Malmö, Sweden, Oct. 2020.

[21] J. Mikulskis, J. K. Becker, S. Gvozdenovic, and D. Starobinski, "Poster: Snout - An Extensible IoT Pen-Testing Tool," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 2529–2531.

[22] S. Bak and Y.-J. Suh, "Designing and Implementing an Enhanced Bluetooth Low Energy Scanner with User-Level Channel Awareness and Simultaneous Channel Scanning," vol. 102, no. 3. The Institute of Electronics, Information and Communication Engineers, 2019, pp. 640–644.

[23] C. D. Kilgour, "A Bluetooth low-energy capture and analysis tool using software-defined radio," Master's Thesis, Simon Fraser University, 2013. [Online]. Available: http://summit.sfu.ca/item/12931

[24] W. Park, D. Ryoo, C. Joo, and S. Bahk, "BLESS: BLE-aided Swift Wi-Fi Scanning in Multi-protocol IoT Networks," in *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*, 2021, pp. 1–10.

[25] J. Hall, B. Ramsey, M. Rice, and T. Lacey, "Z-wave Network Reconnaissance and Transceiver Fingerprinting Using Software-Defined Radios," in *ICCWS 2016*.

[26] L. Choong, "Multi-Channel IEEE 802.15.4 Packet Capture Using Software Defined Radio," *UCLA Networked & Embedded Sensing Lab*, vol. 3, pp. 1–20, 2009.

[27] P. Robyns, P. Quax, W. Lamotte, and W. Thenaers, "A Multi-Channel Software Decoder for the LoRa Modulation Scheme," in *IoTBDS 2018*, 2018, pp. 41–51.

[28] B. Schmeiser, "Batch Size Effects in the Analysis of Simulation Output," *Operations Research*, vol. 30, no. 3, pp. 556–568, 1982.

[29] C. W. Badenhop, S. R. Graham, B. W. Ramsey, B. E. Mullins, and L. O. Mailloux, "The Z-Wave routing protocol and its security implications," *Computers & Security*, vol. 68, pp. 112–129, 2017.

[30] Marija Dimitrijevic. (2018) Replacing many RF receivers with only ONE using Channelization. [Online]. Available: ettus.com/wp-content/uploads/2018/12/Channelization_-_Article_.pdf

[31] Thread Group. (2022) What is Thread? [Online]. Available: https://www.threadgroup.org/What-is-Thread/Overview

[32] FieldComm Group, "WirelessHART: HART Without The Wires," 2021. [Online]. Available: https://www.fieldcommgroup.org/technologies/wirelesshart

[33] S. Gvozdenovic, J. K. Becker, J. Mikulskis, and D. Starobinski. (2022) IoT-Scan Traces. [Online]. Available: https://github.com/nislab/iot-scan