BLESS: BLE-aided Swift Wi-Fi Scanning in Multi-protocol IoT Networks

Wonbin Park[†], Dokyun Ryoo[†], Changhee Joo[‡], and Saewoong Bahk[†]
[†]Department of Electrical and Computer Engineering and INMC, Seoul National University, Seoul, South Korea
[‡]Department of Computer Science and Engineering, Korea University, Seoul, South Korea
Email: {wbpark, dkryoo}@netlab.snu.ac.kr, changhee@korea.ac.kr, sbahk@snu.ac.kr

Abstract—**Wi-Fi scanning that searches neighboring access points (APs) is an essential prerequisite for Wi-Fi operations such as initial association and handover. As the traffic demand increases, APs are more densely deployed and the number of operating Wi-Fi channels also increases, which, however, results in additional scanning delay and makes the scanning a burdensome task. In this paper, we note that the co-location of Wi-Fi protocol with BLE protocol is a common practice in IoT networks, and develop a Wi-Fi passive scanning framework that uses BLE to assist scanning. Although the framework has great potential to improve scanning performance without explicit message exchanges, there are technical challenges related to time synchronization and channel switching delay. We address the challenges and develop a practical passive scanning scheme, named *BLESS-Sync*. We verify its performance through testbed experiments and extensive simulations, and show that *BLESS-Sync* significantly outperforms legacy Wi-Fi scanning in terms of scanning delay and energy efficiency.**

Index Terms—**Wi-Fi scanning, bluetooth low energy, IoT, scheduling**

I. Introduction

When a Wi-Fi station (STA) tries to join a Wi-Fi network, it needs to obtain necessary information from access points (APs). Wi-Fi scanning is the process by which the STA collects information from APs within its range, and it is essential for Wi-Fi operations such as initial association and handover. As the number of active channels and neighboring APs increases, Wi-Fi scanning time gets longer, which greatly reduces air-time for data transmission and degrades network performance [1].

There are two types of Wi-Fi scanning: *active scanning* and *passive scanning*. In the active scanning, a STA broadcasts a probe request to each Wi-Fi channel and receives a corresponding probe response from each neighboring AP. The active scanning achieves fair scanning performance by exchanging control frames when the number of APs is small, but it causes air-time depletion when the number of APs is large. In the passive scanning, a STA waits on a channel to receive a beacon transmitted periodically by each neighboring AP. The passive scanning does not consume air-time for control messages, but requires a relatively long waiting time on each channel, resulting in a longer scanning delay when the number of channels is large. In practice, active scanning is used more often than passive scanning [2].

There are other attempts in the Wi-Fi protocol to improve scanning performance. In [3], simultaneous scanning of the 2.4 GHz and 5 GHz bands has been attempted using two transceivers. This can be used to half the scanning delay at the extra hardware cost of multi-transceiver. A passive scanning scheme developed in [4] lets each AP transmit beacons at a fixed time interval such that a STA can expect the next beacon transmission time. A non-broadcasting-probe based active scanning scheme proposed in [5] reduces the scanning delay by avoiding collisions due to multiple responses. In [6], the authors improve the performance of active scanning by prioritizing the channel access of probe responses. In [7], a STA finds channels where active APs exist based on its handover history, and conducts active scanning on those channels. The authors of [8] consider the correlation between probe response arrival time and AP signal quality, and proposes an active scanning scheme, where a STA waits for a short time to selectively receive probe responses with good signal quality.

In this work, we improve the performance of Wi-Fi *passive scanning* by exploiting co-located Bluetooth Low Energy (BLE). Recently, AP and STA manufacturers tend to put multiple protocols together on a single device due to cost concerns [9]. Such APs act not only as Wi-Fi APs but also as gateways to BLE or ZigBee. They are already on the market, e.g., Samsung Connect Home [10], Google WiFi AP [11], and Cisco Catalyst Series [12]. There are several studies to improve Wi-Fi scanning performance using these co-located protocols. Blue-Fi [13] predicts the presence of nearby APs based on Bluetooth contact-patterns and cell-tower information, and activates the Wi-Fi module according to the prediction. In BLEND [14], BLE advertising packets are used to obtain operating channel information of neighboring Wi-Fi APs. The cross radio technology can also help to detect Wi-Fi APs. It has been shown that ZigBee [15]–[17] and Bluetooth [18] radios can detect Wi-Fi beacons, which, however, needs to listen to multiple beacons for correct detection and work only on the 2.4 GHz band to avoid mismatches in the operating spectrum.

We take a different approach from the aforementioned studies. We design a *BLE-aided Swift Wi-Fi Scanning (BLESS)* framework, in which each AP broadcasts beacon transmission information through BLE (including the operating channel and

Saewoong Bahk is the corresponding author.

This work was supported in part by Institute of Information communications Technology Planning Evaluation (IITP) grant funded by the Korea government (MSIT) (No.2018-0-00923), in part by Institute for Information communications Technology Promotion (IITP) grant funded by the Korean government (MSIT) (No. 2018-0-00815), and in part by the MSIT, Korea, under the ICT Creative Consilience program (IITP-2020-0-01819) supervised by the IITP.

beacon transmission time), and each STA optimizes channel scanning sequence based on this information. To this end, we take into account two practical challenges: *Time synchronization* and *Wi-Fi channel switching delay*. The former is needed to determine the accurate beacon transmission time, and the latter increases the complexity of finding the optimal scanning sequence. We address the two challenges by managing the time uncertainty of BLE advertising and developing heuristic algorithms that achieve good empirical performance.

Our main contributions are summarized as follows:

- We design the *BLESS* framework for swift Wi-Fi passive scanning that uses BLE advertising. This has great potential to improve Wi-Fi scanning performance at the expense of small overhead.
- We develop a practical *BLESS* implementation, named *BLESS-Sync*, which considers two major challenges of time synchronization and channel switching delay.
- We implement our scheme on commercial off-the-shelf (COTS) devices by modifying Wi-Fi and BLE drivers in the Linux kernel, and evaluate its performance through testbed experiments.
- Through extensive simulations, we also show the performance of *BLESS-Sync* in large-size networks, in comparison with legacy Wi-Fi scanning.

The rest of this paper is organized as follows. We explain the background in Section II, and describe the *BLESS* framework and challenges in Section III. We develop our solution that addresses the challenges in Section IV. We evaluate its performance through experiments and simulations in Section V. After a brief discussion of compatibility and transmission range in Section VI, we conclude the paper in Section VII.

II. Background

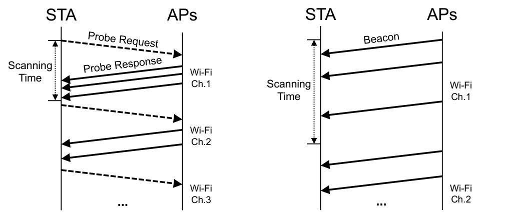
We consider a network of IoT devices where Wi-Fi and BLE protocols are co-located on each AP and STA. We describe the scanning behavior of legacy Wi-Fi protocol, and then explain advertising behavior of BLE protocol, which we will use for swift Wi-Fi scanning.

A. Legacy Wi-Fi Scanning

We first describe Wi-Fi scanning of a legacy device (denoted by legacy Wi-Fi scanning), which is classified into two types of scanning: legacy active scanning and legacy passive scanning. We describe their detailed behaviors.

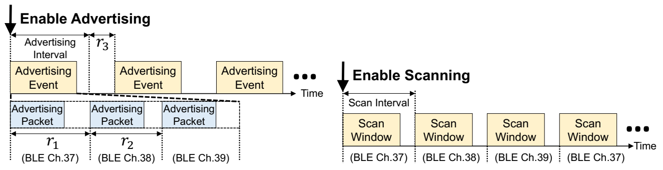
The legacy active scanning is widely used nowadays and its operation is illustrated in Fig. 1(a). A STA broadcasts a probe request frame on each channel to examine each AP. If a nearby AP receives the probe request frame, it informs the STA of its presence by sending a probe response frame. This reactive approach allows the STA to stay on each channel for a relatively short period of time (e.g., 40 ms per channel [19]).[1] However, control frames for probe request/response consume a significant amount of air-time and may lead to a lack of air-time for data transmission. This problem becomes more severe as the deployment of Wi-Fi devices (STAs and APs) gets denser [20], [21].

[1]The scanning time per channel, i.e., the amount of time that a STA stays on a channel is OS and hardware dependent.



(a) Legacy active scanning (b) Legacy passive scanning

Fig. 1. Operation of legacy Wi-Fi scanning.



(a) Advertising process (b) Scanning process

Fig. 2. A snapshot of standard BLE advertising and scanning.

In the legacy passive scanning (Fig. 1(b)), a STA finds nearby APs on each channel by passively receiving beacon frames, which are periodically broadcasted by APs. To ensure successful scanning, a STA stays for at least an AP's beacon interval (typically 102.4 ms [22], [23]) on each channel. This results in a much longer scanning delay compared to legacy active scanning.

B. BLE Advertising

BLE advertising plays two different roles depending on the BLE communication mode: *connection-oriented* and *connection-less*. The connection-oriented mode uses advertising to disseminate the presence of the advertiser. This mode is suitable for bi-directional communication or heavy data transmission. Meanwhile, the connection-less mode exploits advertising for light data transmission. Nearby observers can receive advertising data without any connection overhead. A single advertising packet can contain up to 31 bytes of payload according to the Bluetooth 4.2 specification [24]. In this work, we use the *connection-less* mode for BLE advertising.

Fig. 2 shows a snapshot of standard BLE advertising and scanning. BLE scanning is an operation to receive advertising packets. Meanwhile, BLE advertising enables an AP to invoke an advertising event at a fixed *Advertising Interval*, which can be set in the range of [20, 10240] ms [24]. In each advertising event, the AP broadcasts packets over BLE channels of 37, 38, and 39 in sequence. Each STA also sequentially monitors the three channels at a fixed *Scan Interval* for the duration of *Scan Window*, which can be set in the range of [2.5, 10240] ms [24].

III. BLE-aided Swift Wi-Fi Scanning

In this section, we develop a Wi-Fi scanning framework that leverages BLE in IoT networks, denoted by *BLE-aided Swift Wi-Fi Scanning (BLESS)*. We assume that IoT devices (AP and STA) are with the protocol stacks of Wi-Fi and BLE, which is very common in modern IoT devices. We specify necessary *BLESS* operations at each AP and STA, and discuss several implementation challenges.

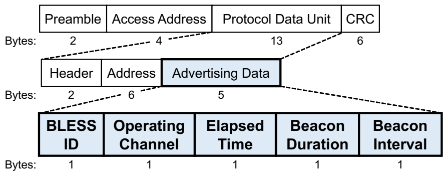


Fig. 3. Proposed advertising packet format.

A. BLESS Framework

The legacy sequential scanning for Wi-Fi channels is time-consuming since an IoT device should scan a channel for a sufficient amount of time, e.g., 102.4 ms (typical Wi-Fi beacon interval [22], [23]), to ensure successful beacon reception from all neighboring APs. However, if we know when beacons will be transmitted *in advance*, the scanning time can be much shorter. This is because the STA can scan each channel in a timely manner and even skip some channels with no AP. We collect Wi-Fi beacon timing information by taking advantage of the co-located protocol, i.e., by using BLE advertising. We describe the basic operation as follows.

1) An AP broadcasts necessary information of its operating channel and beacon transmission time through BLE advertising.
2) A STA collects information from multiple APs and builds a Beacon Transmission time Map (BTM).
3) Using the BTM, the STA optimizes its scanning sequence to minimize overall scanning delay.

The AP sends a BLE advertising packet that contains information about the Wi-Fi beacon. This information includes *BLESS* identifier (ID), operating Wi-Fi channel, elapsed time since the last beacon transmission, beacon duration, and beacon interval, each of which takes 1 byte, as shown in Fig. 3. Note that BLE advertising is mandatory for BLE operation, and our *BLESS* needs to add 5 bytes of overhead to the advertising packet (without requiring any control message exchanges). We denote the BLE advertising packet with additional information for *BLESS* by *BLESS advertising packet*.

In *BLESS*, the STA collects advertising information from multiple APs. When a STA joins a Wi-Fi network, it starts in BLE networks and receives *BLESS* advertising packets from neighboring APs for time T that is sufficiently large to ensure the reception of advertising packets. (We set $T = 200$ ms.[2]) We use the *BLESS* ID field of the *BLESS* advertising packet to identify a *BLESS*-support AP. After the initial time, for each received *BLESS* advertising packet from AP x, the STA computes the expected Beacon Transmission Time (BTT_x) as

$$\begin{aligned} &\text{BTT}_x(t) = \{t_n(t,x),\ n = 0,1,2,\dots\}, \\ &\text{where } t_n(t,x) = -((t - t_r^x + e_x) \mod t_b^x) + n \cdot t_b^x, \end{aligned} \tag{1}$$

t is the current local time of the STA, and t_r^x is the (local) reception time of the *BLESS* advertising packet from AP x. The elapsed time e_x since the last beacon transmission and the beacon interval t_b^x are taken from the received advertising packet (see Fig. 3). After calculating BTT for each neighboring AP, the STA builds the BTM, schedules the scan, and decides *when* and *to which channel* it has to switch to minimize the total time for receiving all the beacons.

[2]The initial period T needs to be set depending on the number of APs and the advertising interval (typically 100 ms [25]). Assuming the advertising interval of 100 ms, we roughly set T to 200 ms (for 20 APs), 330 ms (for 40 APs), 470 ms (for 60 APs), or 630 ms (for 80 APs) [26].

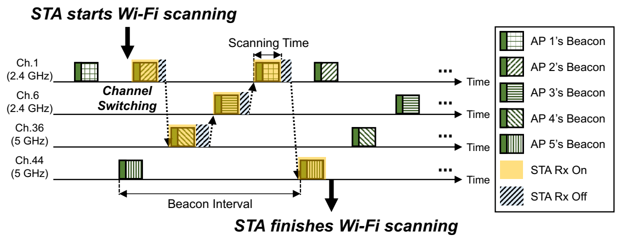


Fig. 4. An example of *BLESS* operation. After collecting the beacon timing information, the STA schedules scanning.

Fig. 4 shows an example of *BLESS* operation. Note that the legacy passive scanning scans all channels in order and stays in each channel for a large amount of time (e.g., for one beacon interval at each channel). The *BLESS* framework helps a STA to scan channels more swiftly and allows it to dwell in the idle mode, saving extra energy when there is no expected beacon (e.g., STA Rx Off in Fig. 4). As a result, it can significantly reduce the time and energy consumed for the Wi-Fi scanning.

B. Implementation Challenges

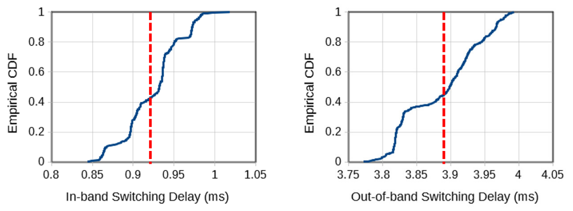
BLESS has the initial overhead T used to collect the beacon timing information. However, even taking into account this overhead, it significantly reduces overall scanning delay. In this section, we discuss several practical issues that need to be addressed to realize *BLESS*.

1) Time synchronization: For accurate timing information, AP and STA times should be synchronized. Time synchronization can be achieved by using GPS, but it is costly and consumes a substantial amount of power. Thus, it is not suitable for most IoT devices that are cheap and battery-limited. An alternative method is to use relative timing information. In *BLESS*, the AP advertises the time since the last beacon transmission as well as its beacon interval, which allows a STA to compute the next beacon transmission time using (1). This approach does not require any special equipment and can be used for low-price IoT devices. However, a couple of issues remain for accurate timing information.

First, there is some randomness in BLE advertising transmission, which makes it hard for AP to set exact time information in the advertising packet. The standard BLE protocol stack sequentially transmits advertising packets over three channels of 37, 38, and 39 during an advertising event, as shown in Fig. 2(a). In this procedure, the BLE protocol suffers from three random delays[3]: two within the advertising event (r_1 and r_2), and one between two consecutive events (r_3). These delays are determined on-the-fly by the BLE controller (hardware or firmware) and are not accessible from the outside.[4] Since the AP cannot precisely estimate when the advertising packet will be transmitted, it is hard to preset the

[3]The random delays are in the range of [0, 10] ms.

[4]At least, for the device firmware that we have tried.



(a) In-band switching delay (b) Out-of-band switching delay

Fig. 5. Empirical CDFs of in-band and out-of-band Wi-Fi channel switching delays. Their averages (marked as vertical dotted lines) are 0.92 ms and 3.89 ms, respectively.

elapsed time e, which is the time between the recent Wi-Fi beacon transmission and the BLE advertising.

Second, uncertainty in transmission timing can also arise from BLE Host Controller Interface (HCI) transport, which we denote by *host delay*. It includes the delay for the BLE host to send an HCI command to the BLE controller (at AP), the delay for the BLE controller to send an HCI report to the BLE host (at STA), and the processing delay (e.g., modulation, coding, etc.) for both transmission and reception. The propagation delay is relatively small and can be ignored.

2) Channel switching delay: Given the beacon timing information, *BLESS* schedules scanning in advance and minimizes the total scanning delay (e.g., see Fig. 4). However, switching between channels takes a non-negligible amount of time, and thus we need to schedule the scanning taking into account the switching delays. For commercial Wi-Fi chipsets, switching to a nearby channel (*within the band* of 2.4 GHz or 5 GHz) takes about 1-8 ms [27]–[29], and switching to a far-away channel *across the band* takes a much longer time [30], [31]. Since the switching delay is of great importance in scheduling, we experimentally measure it to obtain an accurate value.

We use a laptop (Ubuntu 14.04 LTS) equipped with AR9380 Wi-Fi Network Interface Card (NIC). We consider a total of 22 channels: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13 in the 2.4 GHz band and 36, 40, 44, 48, 149, 153, 157, 161, 165 in the 5 GHz band. Let $< a, b >$ denote channel switching from channel a to channel b, and let C denote the set of channels that we consider. For switching cases of $< 1, b >$ for $b \in C \setminus \{1\}$ and $< 36, b >$ for $b \in C \setminus \{36\}$, we measure the switching delays, each for 20 times.

Based on our measurement results,[5] we characterize two different switching types: *in-band switching* and *out-of-band switching*. In-band switching occurs between channels within the 2.4 GHz or 5 GHz band. Out-of-band switching occurs between channels across the bands. On average, the out-of-band switching takes about 4 times longer delay than the in-band switching, as shown in Fig. 5. Further, within each type, the delay variance is small. The largest delay variance is 0.17 ms for the in-band switching and 0.22 ms for the out-of-band switching. We present our measured delays in Fig. 5.

Owing to the large average delay gap between the two switching types and the small delay variance within each type, we are able to precisely schedule the scanning. On the other hand, in terms of optimizing overall scanning delay, the two different switching delays make the problem even more difficult since the minimum time required to receive a beacon depends on the scanning sequence.

IV. Implementation of BLESS

We describe how to address the two implementation challenges aforementioned in Section III-B, and then present the overall structure of our *BLESS* implementation.

A. Achieving Time Synchronization

There are two random delays that make it hard to achieve time synchronization between AP and STA. One is unpredictable random delays in the standard BLE advertising, and the other is a BLE host delay. We eliminate delay uncertainty by manipulating the BLE advertising and compensating for the host delay.

Manipulating the BLE advertising: By default, the BLE controller, which is often inaccessible and implemented in the hardware or firmware, decides when to transmit a BLE advertising packet including the random delays of r_1, r_2, and r_3. These unpredictable delays are the main difficulties in estimating the timing of BLE advertising packet transmission. We address the problem by changing the decision structure such that the BLE host, which is modifiable, makes the transmission decision on behalf of the BLE controller.

To avoid random delays generated by the BLE controller, we pay attention to the first packet when the BLE host enables advertising. Note that the first advertising packet is transmitted immediately after the BLE host enables the advertising, which implies that the BLE host controls the first packet transmission timing. Hence, we can imitate the standard advertising behavior by repeatedly enabling and disabling the advertising on the BLE host (using a preset advertising channel). For example, we reproduce the advertising process in Fig. 2(a) as follows: after initially enabling the advertising, we disable it soon. We change the advertising channel and wait for r_1 random delay (from the first enabling time), after which we re-enable the advertising to immediately transmit the second advertising packet as the original BLE controller does. Afterward, we repeat this enabling/disabling with random delays of r_2 and r_3 to transmit advertising packets in the same time sequence on the corresponding channel. This manipulation allows us to mimic the standard advertising behavior with the knowledge of r_1, r_2, and r_3.

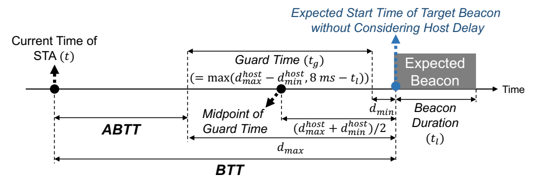
To this end, we make use of HCI commands in the Bluetooth specification that provide the BLE host with an effective means to control various BLE operations. Specifically, we use *LE Set Advertise Enable Command* to enable/disable the advertising, and *LE Set Advertising Parameters Command* to configure the advertising channel before enabling the advertising.

Compensating for the host delay: We address the random host delay in the BLE protocol stacks. If this value has a relatively small variance, we can regard it as a constant and predict the accurate transmission time through compensation.

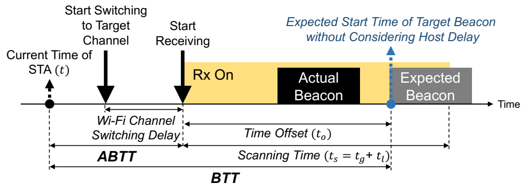
[5]We measure the difference between the time when the kernel (mac80211) triggers a switching to a target channel and the time when it starts receiving by enabling Rx register in ath9k driver. It also corresponds to the interval between Start Switching to Target Channel' and 'Start Receiving' in Fig. 6(b).

TABLE I
MEASUREMENT SUMMARY OF THE SUM OF BLE HOST DELAY AND TRANSMISSION DELAY

Vendor		Broadcom	TI	Nordic
NIC model		BCM20704	CC2540	nRF52840
HCI transport layer type		USB	USB	UART
Sum of host delay and transmission delay (ms)	Min.	3.73	3.01	3.91
	Max.	7.36	5.35	7.41
	Avg.	5.44	4.06	5.84


Expected Start Time of Target Beacon without Considering Host Delay
Current Time of STA (t)
Guard Time (t_g)
$(= \max(d_{max}^{host} - d_{min}^{host}, 8\,ms - t_l))$
Expected Beacon
Time
d_{min}
Beacon Duration (t_l)
Midpoint of Guard Time
$(d_{max}^{host} + d_{min}^{host})/2$
ABTT
d_{max}
BTT

(a) Adjusting BTT considering host delay


Start Switching to Target Channel
Start Receiving
Expected Start Time of Target Beacon without Considering Host Delay
Current Time of STA (t)
Rx On
Actual Beacon
Expected Beacon
Time
Wi-Fi Channel Switching Delay
Time Offset (t_o)
ABTT
Scanning Time $(t_s = t_g + t_l)$
BTT

(b) Receiving a target beacon using Adjusted BTT

Fig. 6. An example of receiving a target beacon after adjusting beacon transmission time taking into consideration the host delay and the switching delay.

We validate our approach by measuring the host delay on COTS BLE devices.

For the sake of measurement, we measure the sum of the host delay and the transmission delay, where the latter can be easily computed by dividing the packet size by the transmission rate. We use two laptops that are time-synchronized using the Network Time Protocol (NTP) [32] with an error of less than 50 μs: one is set as the transmitter and the other as the receiver. We measure the difference from the time when the BLE host at the transmitter enables the advertising to the time when the BLE host at the receiver successfully receives the reception signal (i.e., denoted by the advertising report). We measure the delays of 1000 transmissions for each device of BCM20704 [33], CC2540 [34], and nRF52840 [35]. Table I shows the results. The average delays (i.e., the sum of the host delay and the transmission delay) are 5.44 ms, 4.06 ms, and 5.84 ms for BCM20704, CC2540, and nRF52840, respectively.

Based on the results, we adjust two time parameters for the successful reception of a target beacon: 1) the start time of receiving the beacon and 2) the time duration until the end of the reception. The latter is denoted as *Scanning Time*.

- *Setting the Scanning Time:* Let t_l denote the largest beacon duration among those observed up to now.[6] In our measurements, the minimum host delay is d_{min}^{host}=3.01 ms and the maximum host delay is d_{max}^{host} = 7.41 ms. We set the Scanning Time t_s to the sum of the beacon duration t_l and the guard time[7] $t_g = d_{max}^{host} - d_{min}^{host}$. (See Fig. 6(a).)
- *Adjusting the expected Beacon Transmission Time*: We adjust the beacon start time by adding the midpoint of d_{max}^{host} and d_{min}^{host} to half the guard time. To elaborate, we set *Adjusted BTT (ABTT)* of AP x as

$$\text{ABTT}_x(t) = \{t'_n(t,x),\ n = 0, 1, 2, \dots\}, \tag{2}$$

$$\text{where } t'_n(t,x) = -((t - t_r^x + e_x + t_o) \bmod t_b^x) + n \cdot t_b^x$$

$$\text{and } t_o = \frac{d_{max}^{host} + d_{min}^{host}}{2} + \frac{t_g}{2}.$$

Fig. 6 shows the detailed time structure of beacon duration, guard time, scanning time, time offset, and receiving a target beacon after adjusting BTT.

B. Scan Scheduling with Switching Delays

Based on BTM, a STA determines the scanning sequence to minimize overall scanning delay. In this subsection, we address the problem of finding the optimal scanning sequence, which is challenging since the channel switching delay is dependent on the switching type, i.e., in-band switching or out-of-band switching. This implies that a switching delay depends on the channel scanning sequence. We formulate the problem using a graph, show that it is an NP-hard problem, and develop heuristic solutions.

Problem formulation: We consider a complete directed graph $G(V, A)$ with the set of nodes $V = \{1, ..., n\}$ and the set of arcs $A = \{(i,j) \mid i, j \in V\}$. Let c_{ij} be the cost associated with arc (i,j). We assume no self-loop and set $c_{ii} = \infty$ for all $i \in V$. Each node corresponds to an AP associated with its beacon transmission time (i.e., ABTT$_x$ for AP x), except one node, denoted by node k, which is a starting point of the scanning. For ease of explanation, we say that STA initially sets its transceiver to the channel of virtual AP$_k$. We assume that all APs have the same beacon interval t_b, which is common in practice [18], [23].

Suppose that, after the initial BLE reception, node k starts the Wi-Fi scanning at time t_k. Then for each AP$_i$ known from received BLE advertisement, we compute ABTT$_i$ by setting $t = t_k$ in (2). It is the waiting time of the STA if it receives the beacon from AP$_i$ for the first time, and can be computed as

$$c_{ki} = (t_i \ominus d_{ki}) + d_{ki},$$

where $\ominus$ is the modulus minus with respect to t_b, d_{ki} denotes the switching delay from the current channel to the channel of AP$_i$, and t_i is an element in ABTT$_i$. We note that due to the modulus operation, any $t_i \in$ ABTT$_i$ leads to the same c_{ki}. Also, d_{ki} depends on whether the two APs k and i are on the same channel, on a different channel in the same band (in-band switching), or on a different channel in the different band (out-of-band switching). Similarly, assuming that the STA scans AP$_j$'s channel following AP$_i$'s channel, we set c_{ij} for arc (i,j) with $i, j \neq k, i \neq j$ as

$$c_{ij} = (t_j \ominus (t_i + t_s + d_{ij})) + t_s + d_{ij},$$

where t_j is an element in ABTT$_j$, t_s denotes the scanning time per target beacon (i.e., beacon from AP$_j$), and d_{ij}

[6]We use the largest beacon duration instead of the beacon duration specified in the BLE advertising packet since the value of Scanning Time cannot change during the scanning once a scanning procedure starts.

[7]In our implementation, we observe several unstable system behaviors when the Scanning Time is less than 8 ms. Hence, we set the guard time to $\max\{d_{max}^{host} - d_{min}^{host}, 8\text{ ms} - t_l\}$ such that the Scanning Time is at least 8 ms.

denotes the corresponding switching delay. For c_{ik}, we set $c_{ik} = \max\{c_{ij} \mid (i,j) \in A, j \neq k, i\}$ for all i.

Note that we set the cost c_{ij} (for $i,j \neq k, i \neq j$) to satisfy the minimum time difference on receiving a beacon from AP$_j$ following a beacon from AP$_i$. Thus, finding the sequence of scans to receive all the beacons starting from node k is equivalent to finding a cycle from node k visiting every node exactly once in the graph, which is known as finding a Hamiltonian cycle [36]. Furthermore, finding a cycle with the minimum cost[8] is equivalent to finding a sequence to minimize the scanning delay. We can formulate this as an integer programming problem as follows.

$$\begin{aligned}
\text{minimize} \quad & \textstyle\sum_{i\in V}\sum_{j\in V} c_{ij}x_{ij} && \text{(3a)}\\
\text{subject to} \quad & \textstyle\sum_{i\in V} x_{ij} = 1, \quad j \in V, && \text{(3b)}\\
& \textstyle\sum_{j\in V} x_{ij} = 1, \quad i \in V, && \text{(3c)}\\
& \textstyle\sum_{i\in S}\sum_{j\in S} x_{ij} \leq |S| - 1, && \text{(3d)}\\
& x_{ij} \in \{0,1\}, S \subset V \text{ with } |S| \geq 2. && \text{(3e)}
\end{aligned}$$

Constraints (3b) and (3c) imply that the in-degree and out-degree of each node should be one, and constraint (3d) prevents creating sub-cycles in the graph.

Heuristic solutions: Finding a Hamiltonian cycle with the minimum cost in a given directed graph is known as asymmetric traveling salesman problem (ATSP), which is an NP-hard problem [37]. There have been several heuristics with good empirical performance [38]. In this paper, we consider three ATSP heuristic algorithms to schedule scanning: 1) *First Come First Served (FCFS) ($O(n)$)*, 2) *Nearest Neighbor (NN) ($O(n^2)$)* and 3) *3-opt ($O(n^3)$)*.

We describe how the heuristics construct a scanning sequence. Note that we should order $n-1$ nodes starting from the starting node (i.e., node k). Let V_u be the set of unscheduled nodes and x be the latest selected node. We start with $V_u = V \backslash \{k\}$ and $x = k$. FCFS selects node y where $y = \arg\min_{z\in V_u} c_{xz}$ and then removes node y from V_u. It repeats this greedy selection until V_u is empty. For NN, it acts similarly to FCFS except the first AP selection. It fixes the first node j and obtains a path starting $\{k, j, \dots\}$ using the greedy selection as FCFS. Then it repeats this path selection changing the first node for all $j \in V$, resulting in $n-1$ paths. Among these paths, NN uses the minimum cost path.

While FCFS and NN are path construction algorithms, 3-opt is a path improvement algorithm. Given a path, 3-opt picks three arcs on the path and removes them, which divides the original path into 3 parts, say $A{\to}B{\to}C{\to}A$. It generates new path by rearranging the parts in the order of $A{\to}C{\to}B{\to}A$ and by adding the corresponding arcs between the parts. If the new path has a smaller cost, then the original path is replaced with the new one. 3-opt repeats this procedure. To generate the initial path, we use NN and denote the resultant algorithm by *NN+3-opt*.

[8]Note that every cycle needs the last hop to node k to become a cycle, and we set all the last hop cost c_{ik} to the same value. Thus, the last hop cost does not impact on finding the minimum cost.

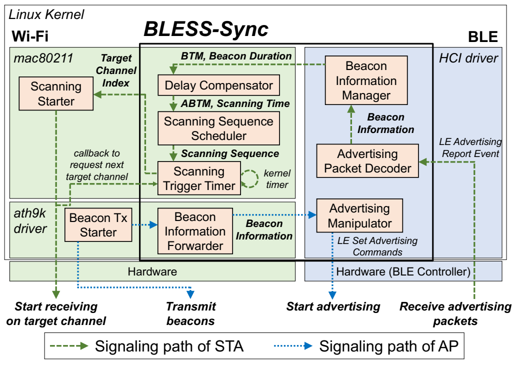


Fig. 7. Implementation structure of *BLESS-Sync*.

C. Implementation Structure

We implement our *BLESS* scheme, named *BLESS-Sync*, on COTS laptops (Ubuntu 14.04 LTS with the Linux kernel 3.13). We use AR9380 Wi-Fi NIC and BCM20704 BLE NIC. *BLESS-Sync* is located in mac80211 and ath9k driver at Wi-Fi side, and HCI driver at BLE side. Fig. 7 describes the implementation structure of *BLESS-Sync*. The details of function flow are as follows.

We first describe the flow of AP. *Beacon Tx Starter* hands over the beacon information (Operating Channel, Beacon Duration, and Beacon Interval) to the BLE side. Also, *Beacon Tx Starter* constantly notifies the timing of the last beacon transmission whenever new beacon transmission occurs,[9] which is essential to compute the Elapsed Time afterward. Then *Advertising Manipulator* configures advertising data with the delivered beacon information. Note that with our advertising manipulation, the Elapsed time is precisely set. Finally, it starts to transmit *BLESS* advertising packets with the corresponding information (Fig. 3).

For STA, it starts with receiving advertising packets. Upon a reception, *Advertising Packet Decoder* differentiates *BLESS* advertising packets from legacy (non-*BLESS*) advertising packets by comparing *BLESS* ID. If the packet is a *BLESS* advertising packet, *Advertising Packet Decoder* hands over the received *BLESS* information to *Beacon Information Manager*. After obtaining the BTM and (longest) beacon duration from the information, the STA forwards them to the Wi-Fi side. *Delay Compensator* at the Wi-Fi side adjusts the BTM and sets the relaxed scanning time considering the host delay and the transmission delay of BLE. With the adjusted BTM (ABTM in short) and scanning time, *Scanning Sequence Scheduler* constructs a scanning sequence by one of the scheduling algorithms (FCFS, NN, or NN+3-opt). Finally, *Scanning Trigger Timer* starts to scan following the sequence. It recursively notifies *Scanning Starter* of the target channel index until all APs are found. Also, it timely calls *Scanning Starter* at the Start Switching to Target Channel (Fig. 6(b)), considering the Wi-Fi channel switching delay.[10]

[9]We use the time that a beacon frame is put into the hardware queue when *ath9k_hw_puttxbuf()* is called. Although actual beacon transmission can be slightly delayed by channel contention, it is negligible considering the sufficiently long guard time (see Fig. 6).

[10]We use the Linux kernel timer to call *Scanning Starter* timely.

V. Performance Evaluation

In this section, we evaluate the performance of *BLESS-Sync* through both testbed experiments and simulations.

A. Testbed Experiments

Experimental settings: In each experiment, we use total N+1 COTS laptops equipped with AR9380 Wi-Fi NIC and BCM20704 BLE NIC. Ubuntu 14.04 LTS is used. One laptop works as a STA and the other N laptops are set as APs using *hostapd* 2.8 [39]. The same set of channels in Section III-B is used. We allocate $N/2$ APs on each of 2.4 GHz and 5 GHz bands, and let each AP operate on a different channel. The transmission power is set to 20 dBm for Wi-Fi and 8 dBm for BLE. We locate the APs within the STA's range for both Wi-Fi and BLE, and set the Wi-Fi beacon size to 225 bytes that corresponds to a beacon duration of 1.8 ms in 2.4 GHz band and 0.3 ms in 5 GHz band,[11] respectively. Also, we use the typical beacon interval of 102.4 ms [22], [23], and the typical BLE advertising interval of 100 ms [25].

For the STA, we set the initial advertising reception time T to 200 ms, and both the BLE scanning window and BLE scanning interval to 80 ms. In our settings, the longest beacon duration t_l is 1.8 ms, and thus the guard time t_g is set to 6.2 ms to ensure the scanning time of at least 8 ms (See Section IV-A). Based on our measurements in Section III-B, we set the in-band switching delay and the out-of-band switching delay to 1.1 ms and 4.1 ms, respectively.

We evaluate the performance of *BLESS-Sync* along with other five scanning methods: 1) *BLESS-Sync* with Exhaustive search (BSE), 2) Selective Active Scanning (SAS), 3) BlueScan Passive Scanning (BPS) [18], 4) Legacy Active Scanning (LAS), and 5) Legacy Passive Scanning (LPS). BSE is the same as *BLESS-Sync* except that it finds the optimal scanning sequence through exhaustive search. This is impractical due to the high computational complexity, so we use it as a performance reference. SAS finds the channels on which an AP operates, and conducts active scanning on only those channels. It shows the same behaviors as BLEND [14]. BPS conducts passive scanning skipping over no-AP channels, and reduces scanning time per beacon using the beacon transmission time. However, it does not consider scanning sequence scheduling (i.e., it scans channels in ascending order). Note that as *BLESS-Sync*, BSE, SAS, and BPS also require some information such as operating channel and beacon transmission time. We assume that they obtain such information through *BLESS* advertising packets. For SAS and LAS, we set the scanning time per channel to 40 ms as in [19], and for LPS, we set the scanning time per channel to 111 ms (default value in the Linux kernel). For *BLESS-Sync*, BSE, and BPS, we set the scanning time of one beacon to 8 ms.

For simplicity, we denote *BLESS-Sync with FCFS* by *BSF*, *BLESS-Sync with NN* by *BSN*, and *BLESS-Sync with NN+3-opt* by *BSN3*, respectively.

[11]The rate of beacon is 1 Mbps and 6 Mbps in 2.4 GHz band and 5 GHz band, respectively.

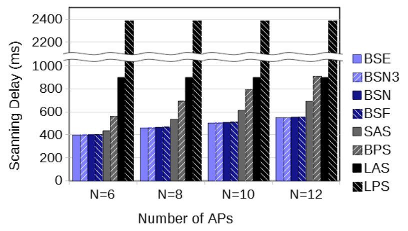


Fig. 8. Experimental results of scanning delay. The delays in *BLESS-Sync*, SAS, and BPS include the initial BLE overhead T.

Experimental results: Fig. 8 shows the scanning delay of the schemes according to the number of APs. The results are averaged over 100 times where each has a different BTM. All *BLESS-Sync* schemes, by which we denote the three schemes of *BSF*, *BSN* and *BSN3*, achieve similar performance. The figure also shows that *BLESS-Sync*, SAS, and BPS achieve significantly shorter scanning delay than LPS. This is because, unlike LPS that uses a long scanning time for each channel, *BLESS-Sync* and BPS use a short scanning time based on the expected beacon transmission time, and even skip over no-AP channels. *BLESS-Sync* further reduces the scanning delay by optimizing the scanning sequence, outperforming LAS. As the number of APs increases, the performance gap between *BLESS-Sync* and the others widens.

We further investigate the performance of our heuristic *BLESS-Sync* schemes. Fig. 9 shows the scanning delay gap from BSE. Note that if the gap is zero, the heuristic finds the optimal scanning sequence. The figure provides us with 6 indicators: 1) the minimum excluding outliers (lower black line), 2) the maximum excluding outliers (upper black line), 3) 25th percentile (lower boundary of the blue box), 4) 75th percentile (upper boundary of the blue box), 5) median (red line), and 6) outliers (red cross).[12] We focus on the maximum excluding outliers (MEO in short) and the outliers. *BSF* shows fair performance in terms of MEO, but sometimes shows extreme outliers (e.g., 70.2 ms when N=10 and 90.9 ms when N=12), which implies that a poor scanning sequence is constructed. In contrast, *BSN* removes the problem of extreme outliers in *BSF*, and also achieves slightly smaller MEO than *BSF*. *BSN3* even further improves performance, in particular, achieving zero MEO when N=6,8,10.

Next, we consider energy consumption of *BLESS-Sync* based on the power profile. We focus on STA, which is reasonable since APs are often wall-powered. We estimate the energy consumption from our experimental measurements and the power profile described in BCM4354 datasheet [40]. To elaborate, we model the expected energy consumption for scanning as

$$\begin{aligned} E = & P_{w,i} \cdot t_{w,i} + P_{w,t} \cdot t_{w,t} + P_{w,r} \cdot t_{w,r} \\ & + P_{w,c} \cdot t_{w,c} + P_{b,r} \cdot t_{b,r}, \end{aligned} \tag{4}$$

where $P_{w,i}$, $P_{w,t}$, $P_{w,r}$, $P_{w,c}$, and $P_{b,r}$ are the expected power consumption in the states of Wi-Fi_Idle, Wi-Fi_Tx, Wi-Fi_Rx, Wi-Fi_CCA, and BLE_Rx, respectively. Also $t_{w,i}$, $t_{w,t}$, $t_{w,r}$,

[12]Outliers are the points that are located out of the range of [Q1 $-$ 1.5 $\cdot$ IQR, Q3 + 1.5 $\cdot$ IQR], where Q1 is 25th percentile, Q3 is 75th percentile and the interquartile range (IQR) is (Q3 $-$ Q1).

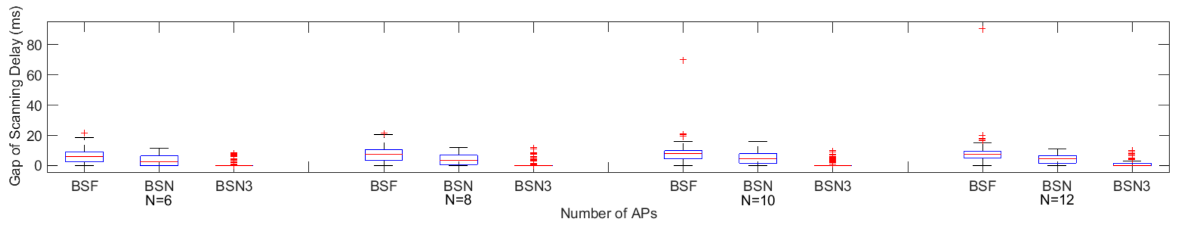


Fig. 9. Experiment results of scanning delay with respect to BSE.

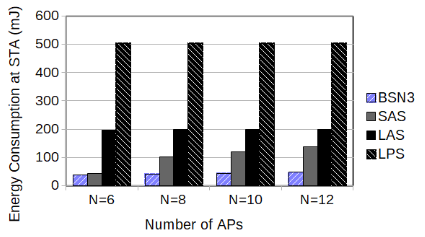
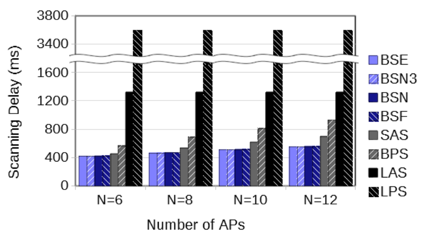
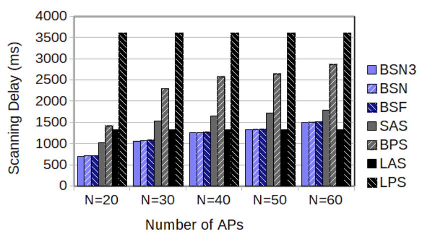


Fig. 10. Estimated energy consumption at STA for one-shot scanning.



(a) Small-scale network scenarios



(b) Large-scale network scenarios

Fig. 11. Simulation results of scanning delay according to the number of APs. The delays in *BLESS-Sync*, SAS, and BPS include the initial BLE overhead T.

$t_{w,c}$, and $t_{b,r}$ are the spending times in the states of Wi-Fi_Idle, Wi-Fi_Tx, Wi-Fi_Rx, Wi-Fi_CCA, and BLE_Rx, respectively. We use the datasheet for the expected power consumption and experimental results for the time spent in each state. Note that *BLESS-Sync* dwells in Wi-Fi_Idle when there is room between two consecutive scans, saving extra energy.

Fig. 10 shows the estimated energy consumption. Even taking into account the additional overhead of using BLE for T=200 ms, *BLESS-Sync* and SAS consume much less energy than the legacy scanning methods. This is due to shorter total scanning delay by skipping the scanning on channels with no AP. Also, *BLESS-Sync* performs better than SAS with the number of APs. This is due to two reasons. One is that *BLESS-Sync* saves energy by dwelling in Wi-Fi_Idle during the scanning, which is missing in SAS. The other is that, SAS actively transmits probe requests/ACKs[13] and receives probe responses, while *BLESS-Sync* passively receives beacons.

B. Simulation

We conduct extensive simulations to investigate our schemes in large-scale networks and the impact of the guard time on scanning delay performance.

Simulation settings: We consider a total of 32 channels across the 2.4 GHz and 5 GHz bands,[14] and deploy APs uniformly over the channels in a round robin manner. We change the initial advertising reception time T according to the number of APs as 200 ms for up to 20 APs, 260 ms for 30 APs, 330 ms for 40 APs, 400 ms for 50 APs and 470 ms for 60 APs [26]. Otherwise specified, we set the guard time t_g to 6.2 ms as in the experiments. We assume that a STA successfully receives a target beacon within the (relaxed) scanning time. The other parameters are also set as in the experiments.

Simulation results: We first evaluate the scanning delay in *BLESS-Sync* according to the number of APs. We denote the case of less-than-20 APs as a small-scale network, and the case of no-less-than-20 APs as a large-scale network. Fig. 11 shows the simulation results in terms of scanning delay. The results are averaged over 1000 times where we generate a different BTM for each run.

In small-scale network scenarios (Fig. 11(a)), the results are similar to those of our experiments. *BLESS-Sync*, BPS, and SAS substantially reduce scanning delay compared to the legacy scanning methods by skipping the scan on channels with no AP. Moreover, *BLESS-Sync* and BPS show much shorter scanning delay than LPS owing to its short scanning time per target beacon. We also observe that the *BLESS-Sync* schemes outperform the others by accelerating the scanning procedure through optimizing the scanning sequence.

In large-scale network scenarios (Fig. 11(b)), owing to the scan scheduling, *BLESS-Sync* shows the shortest scanning delay among the other scanning methods up to N=40. Even when the number of APs is very large (e.g., N=50 or 60), *BLESS-Sync* still shows competitive performance compared to LAS. On the other hand, SAS and BPS lag behind LAS from when N=30 and 20, respectively. This is mainly because they lose their merit, skipping channels where no AP exists, if most of the Wi-Fi channels are occupied by the APs while having the initial time overhead from BLE.

To further investigate scanning delay performance of the heuristic algorithms, we depict the performance gap from BSE for small-scale network scenarios in Fig. 12. For *BSF*, as the number of APs increases, extreme outliers (e.g., gap of over 50 ms) that indicate poor scanning sequence construction are frequently observed. *BSN* resolves most of the extreme outliers in *BSF*, using better scanning sequences. *BSN3* shows the closest performance to the optimal scheduling algorithm in terms of scanning delay. It further reduces the frequency of extreme outliers and shows almost zero MEO even for N=12 while the others have the MEO of around 15 ms.

In large-scale network scenarios, we depict the scanning delay gap from *BSN3* which is the most evolved algorithm

[13]A STA transmits an ACK replying to a probe response from an AP.

[14]According to North America FCC regulatory domain, there exists 11 channels and 21 channels in the 2.4 GHz and 5 GHz bands, respectively.

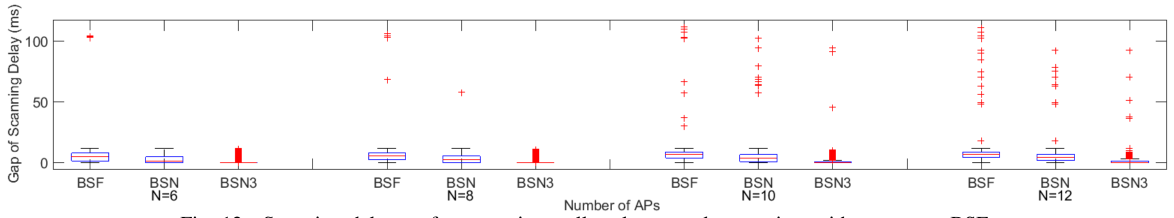


Fig. 12. Scanning delay performance in small-scale network scenarios, with respect to BSE.

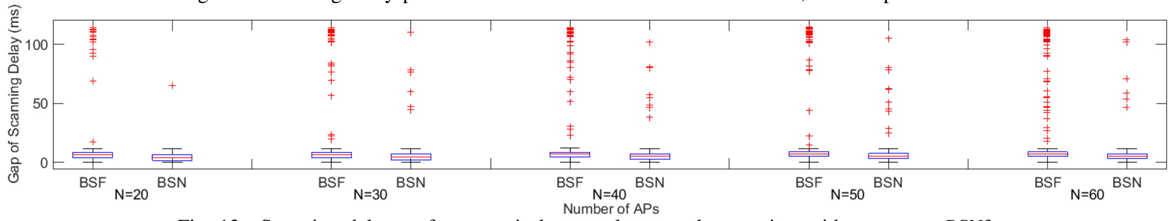


Fig. 13. Scanning delay performance in large-scale network scenarios, with respect to *BSN3*.

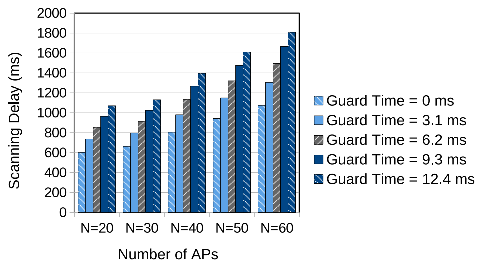


Fig. 14. Scanning delays with different guard times including the initial BLE overhead T.

among our heuristic algorithms in Fig. 13. If the gap is large, it implies that *BSN3* significantly improves the scanning sequence constructed by *BSF* or *BSN*. For *BSF*, even when $N = 20$, it shows 16 extreme outliers (e.g., gap of over 50 ms). As the number of APs increases, the number of extreme outliers gradually increases. In *BSN*, the number of extreme outliers is much smaller than that in *BSF*, but still it has a few.

Next, we investigate the impact of guard time on scanning delay performance. Note that there is room for further improvement of *BLESS-Sync* if we can reduce the host delay variation (i.e., reduce the guard time) in *BLESS* by implementing *BLESS* on the BLE controller level (firmware or hardware). We depict the results of scanning delay under various guard times in Fig. 14. We use *BSN3* in the simulations. We observe that the scanning delay increases as the guard time increases. Note that the shorter scanning time the STA uses, the more beacons it can receive within the same time. The delay gap between different guard time settings increases with the number of APs.

VI. Discussion

A. Coexistence with Non-BLESS APs

If an AP is not compliant with *BLESS*, e.g., due to the absence of BLE protocol stack, it cannot notify a STA of necessary beacon information, and the STA may miss beacons from non-*BLESS* APs.

A simple remedy to the problem is to let *BLESS* APs help out non-*BLESS* APs on the same Wi-Fi channel, and generally deliver the beacon information of non-*BLESS* APs to the STA. To this end, a *BLESS* AP figures out the APs in the same channel by listening to Wi-Fi beacons. Since the *BLESS* AP can also listen to BLE advertising packets, it can differentiate *BLESS* APs from non-*BLESS* APs on the same channel, and the beacon information of non-*BLESS* APs can piggyback on BLE advertising packets. For Wi-Fi channels with no *BLESS* AP, the STA can selectively conduct a legacy scanning method (e.g., legacy active scanning) on those channels.

B. Different Transmission Range of Wi-Fi and BLE

It is well-known that Wi-Fi and BLE have different transmission power and so different transmission range [41], which may cause an inefficient operation of *BLESS*. Let r_w and r_b denote the transmission range of Wi-Fi and BLE, respectively. If $r_w > r_b$, a STA may not receive advertising packets from neighboring APs, and if $r_w < r_b$, a STA may scan channels with no AP.

A quick solution to this problem is to control the transmission power such that $r_w \approx r_b$. The Bluetooth 5.0 specification [42] provides a wide range of transmission power in [-20, 20] dBm which implies r_b can be even longer than r_w [43], [44]. We can exploit this feature to set r_b as close as possible to r_w. This high-power transmission is applied to only BLE advertisements of APs (wall-powered), and thus it does not incur the energy issue at STAs. In certain cases (e.g., indoor environments), it may be hard to set $r_w \approx r_b$ in all directions due to different signal characteristics of Wi-Fi and BLE. In such circumstances, it would better set r_b to a value of slightly longer than r_w to prevent STAs from missing APs.

VII. Conclusion

In this paper, we proposed a framework of swift Wi-Fi scanning aided by the co-located BLE protocol in IoT networks, named *BLESS*. *BLESS*-enabled APs periodically broadcast their Wi-Fi information through BLE advertising, and *BLESS*-enabled STAs use the information to achieve swift Wi-Fi scanning. Taking into account implementation challenges such as time synchronization and channel switching delay, we developed a practical scanning scheme of *BLESS-Sync* and implemented it on COTS devices. Through testbed experiments and large-scale simulations, we showed that our scheme significantly reduces the Wi-Fi scanning delay and energy consumption for scanning.

References

[1] C. Pei, Z. Wang, Y. Zhao, Z. Wang, Y. Meng, D. Pei, Y. Peng, W. Tang, and X. Qu, "Why it takes so long to connect to a WiFi access point," in *Proc. IEEE INFOCOM*, 2017, pp. 1–9.

[2] T. Choi, Y. Chon, and H. Cha, "Energy-efficient WiFi scanning for localization," *Pervasive and Mobile Computing*, vol. 37, pp. 124–138, 2017.

[3] G. Hegde, H. Vaidya, R. Raman, and S. Neelakandan, "Parallel scanning of wireless channels," Feb. 21 2017, uS Patent 9,578,595.

[4] I. Ramani and S. Savage, "SyncScan: practical fast handoff for 802.11 infrastructure networks," in *Proc. IEEE INFOCOM*, vol. 1, 2005, pp. 675–684.

[5] M. Jaakkola, A. Suomi, and J. Poyhonen, "Usage of multiple SSIDs for doing fast WLAN network discovery," Jan. 4 2007, uS Patent App. 11/372,037.

[6] Z. Zhun, "Method to achieve fast active scan in 802.11 WLAN," Feb. 22 2011, uS Patent 7,894,405.

[7] M. Shin, A. Mishra, and W. A. Arbaugh, "Improving the latency of 802.11 hand-offs using neighbor graphs," in *Proc. ACM MobiSys*, 2004, pp. 70–83.

[8] J. Teng, C. Xu, W. Jia, and D. Xuan, "D-scan: Enabling fast and smooth handoffs in ap-dense 802.11 wireless networks," in *Proc. IEEE INFOCOM*, 2009, pp. 2616–2620.

[9] J. Han, C. Joo, and S. Bahk, "Resource sharing in dual-stack devices: Opportunistic Bluetooth transmissions in WLAN busy periods," *IEEE Transactions on Mobile Computing*, vol. 17, no. 10, pp. 2396–2407, 2018.

[10] "Samsung connect home," https://www.samsung.com/sg/smarthome/.

[11] "Google wifi ap," https://madeby.google.com/wifi/.

[12] "Cisco catalyst 9100 ap," https://www.cisco.com/c/en/us/products/wireless/catalyst-9100ax-access-points/.

[13] G. Ananthanarayanan and I. Stoica, "Blue-Fi: enhancing Wi-Fi performance using Bluetooth signals," in *Proc. ACM Mobisys*, 2009, pp. 249–262.

[14] J. Choi, G. Lee, Y. Shin, J. Koo, M. Jang, and S. Choi, "Blend: BLE beacon-aided fast wifi handoff for smartphones," in *Proc. IEEE SECON*, 2018, pp. 1–9.

[15] R. Zhou, Y. Xiong, G. Xing, L. Sun, and J. Ma, "ZiFi: wireless LAN discovery via ZigBee interference signatures," in *Proc. ACM MobiCom*, 2010, pp. 49–60.

[16] Y. Gao, J. Niu, R. Zhou, and G. Xing, "ZiFind: Exploiting cross-technology interference signatures for energy-efficient indoor localization," in *Proc. IEEE INFOCOM*, 2013, pp. 2940–2948.

[17] J. Ansari, T. Ang, and P. Mähönen, "WiSpot: fast and reliable detection of Wi-Fi networks using IEEE 802.15. 4 radios," in *Proc. ACM MobiWac*, 2011, pp. 35–44.

[18] J. Yi, W. Sun, J. Koo, S. Byeon, J. Choi, and S. Choi, "BlueScan: Boosting Wi-Fi scanning efficiency using Bluetooth radio," in *Proc. IEEE SECON*, 2018, pp. 1–9.

[19] X. Chen and D. Qiao, "HaND: Fast handoff with null dwell time for IEEE 802.11 networks," in *Proc. IEEE INFOCOM*, 2010, pp. 1–9.

[20] X. Hu, L. Song, D. Van Bruggen, and A. Striegel, "Is there wifi yet? how aggressive probe requests deteriorate energy and throughput," in *Proc. ACM IMC*, 2015, pp. 317–323.

[21] J. Heo, Y. Yoo, J. Suh, W. Park, J. Paek, and S. Bahk, "FMS-AMS: Secure proximity-based authentication for wireless access in Internet of Things," *Journal of Communications and Networks*, vol. 22, no. 4, pp. 338–347, 2020.

[22] IEEE 802.11, *Part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications*, IEEE Std., Mar. 2012.

[23] H. Lee, J. Kim, C. Joo, and S. Bahk, "BeaconRider: Opportunistic sharing of beacon air-time in densely deployed WLANs," in *Proc. IEEE ICNP*, 2019, pp. 1–11.

[24] B. SIG, "Bluetooth core specification 4.2," *Bluetooth SIG*, vol. 9, 2013.

[25] M. Koühne and J. Sieck, "Location-based services with iBeacon technology," in *Proc. IEEE AIMS*, 2014, pp. 315–321.

[26] G. Shan and B.-H. Roh, "Advertisement interval to minimize discovery time of whole BLE advertisers," *IEEE Access*, vol. 6, pp. 17 817–17 825, 2018.

[27] D. Murray, M. Dixon, and T. Koziniec, "Scanning delays in 802.11 networks," in *Proc. IEEE NGMAST*, 2007, pp. 255–260.

[28] A. Sharma and E. M. Belding, "Freemac: framework for multi-channel mac development on 802.11 hardware," in *Proc. ACM PRESTO*, 2008, pp. 69–74.

[29] V. Navda, A. Bohra, S. Ganguly, and D. Rubenstein, "Using channel hopping to increase 802.11 resilience to jamming attacks," in *Proc. IEEE INFOCOM*, 2007, pp. 2526–2530.

[30] S. Sur, I. Pefkianakis, X. Zhang, and K.-H. Kim, "Wifi-assisted 60 ghz wireless networks," in *Proc. ACM MobiCom*, 2017, pp. 28–41.

[31] M. Yun, Y. Zhou, A. Arora, and H.-A. Choi, "Channel-assignment and scheduling in wireless mesh networks considering switching overhead," in *Proc. IEEE ICC*, 2009, pp. 1–6.

[32] D. L. Mills *et al.*, "Network time protocol (NTP)," 1985.

[33] "BCM20704 datasheet," https://www.cypress.com/file/298026/download.

[34] "CC2540 datasheet," https://www.ti.com/lit/ds/symlink/cc2540.pdf.

[35] "nRF52840 datasheet," https://infocenter.nordicsemi.com/pdf/nRF52840_PS_v1.0.pdf.

[36] T. Akiyama, T. Nishizeki, and N. Saito, "NP-completeness of the Hamiltonian cycle problem for bipartite graphs," *Journal of Information processing*, vol. 3, no. 2, pp. 73–76, 1980.

[37] D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook, *The traveling salesman problem: a computational study*. Princeton university press, 2006.

[38] D. S. Johnson, G. Gutin, L. A. McGeoch, A. Yeo, W. Zhang, and A. Zverovitch, "Experimental analysis of heuristics for the ATSP," in *The traveling salesman problem and its variations*. Springer, 2007, pp. 445–487.

[39] "hostapd: IEEE 802.11 AP, IEEE 802.1X/WPA/WPA2/EAP/RADIUS Authenticator," https://w1.fi/hostapd/.

[40] "BCM4354 datasheet," https://www.cypress.com/file/298141/download.

[41] C.-H. Kao, R.-S. Hsiao, T.-X. Chen, P.-S. Chen, and M.-J. Pan, "A hybrid indoor positioning for asset tracking using Bluetooth low energy and Wi-Fi," in *Proc. IEEE ICCE-TW*, 2017, pp. 63–64.

[42] S. Bluetooth, "Bluetooth 5.0 Core Specification," *Retrieved April*, vol. 12, p. 2019, 2016.

[43] G. Anastasi, E. Borgia, M. Conti, and E. Gregori, "IEEE 802.11b ad hoc networks: Performance measurements," *Cluster Computing*, vol. 8, no. 2-3, pp. 135–145, 2005.

[44] M. Collotta, G. Pau, T. Talty, and O. K. Tonguz, "Bluetooth 5: A concrete step forward toward the IoT," *IEEE Communications Magazine*, vol. 56, no. 7, pp. 125–131, 2018.