Identifying the BLE Advertising Channel for Reliable Distance Estimation on Smartphones

Christian Gentner[*]
German Aerospace Center (DLR)
Wessling, Germany
christian.gentner@dlr.de

Daniel Günther[*]
Technical University of Munich (TUM)
Munich, Germany
d.guenther@tum.de

Philipp H. Kindt[*]
Technical University of Munich (TUM)
Munich, Germany
philipp.kindt@tum.de

Abstract—As a response to the global COVID-19 surge in 2020, many countries have implemented lockdown or stay-at-home policies. If, however, the contact persons of every infected patient could be identified, the number of virus transmissions could be reduced, while the more incisive measures could be softened. For this purpose, contact tracing using smartphones is being considered as a promising technique. Here, smartphones emit and scan for Bluetooth Low Energy (BLE) signals for detecting devices in range. When a device is detected, its distance is estimated by evaluating its received signal strength. The main insight that is exploited for distance estimation is that the attenuation of a signal increases with the distance along which it has traveled. However, besides distance, there are multiple additional factors that impact the attenuation and hence disturb distance estimation. Among them, frequency selective hardware and signal propagation belong to the most significant ones. For example, a BLE device transmits beacons on three different frequencies (channels), while the transmit power and the receiver sensitivity depend on the frequency. As a result, the received signal strength varies for each channel, even when the distance remains constant. However, the information on which wireless channel a beacon has been received is not made available to a smartphone. Hence, this error cannot be compensated, e.g., by calibration. In this paper, we for the first time provide a solution to detect the wireless channel on which a packet has been received on a smartphone. We experimentally evaluate our proposed technique on multiple different smartphone models. Our results help to make contact tracing more robust by improving the accuracy of distance estimation.

Index Terms—Bluetooth, Bluetooth Low Energy, BLE, Channel, Contact Tracing, Neighbor Discovery, Distance Estimation, COVID-19, SARS-CoV-2

I. INTRODUCTION

The global surge of the novel coronavirus SARS-CoV-2 has resulted in many countries implementing lockdown and stay-at-home policies. This has lead to a burdensome situation for the population and causes severe economic problems. However, whenever a person is tested positive, if all of their contacts could be identified, the further spread of the virus could potentially be stopped or slowed down, while other, more incisive measures could be softened.

For this purpose, smartphone-based contact tracing approaches are being considered as an important tool. Here, every device continually transmits Bluetooth Low Energy (BLE) beacons and listens to the channel for incoming transmissions.

As soon as a beacon from another device is received, the distance between both of them is estimated. When the contact duration (which is estimated based on the period of time during which beacons from a particular device are received) is sufficient and/or the estimated distance is small enough, a contact is considered as relevant. Hence, when a person is tested positive for SARS-CoV-2, all relevant contacts can be identified in retrospect by evaluating the smartphone data [1]. Besides contact tracing, distance- or proximity estimation plays a key role in a host of applications, such as indoor navigation or object tracking.

Distance Estimation: Distance estimation works as follows. One device transmits a beacon with a certain transmit power P_t. This value is piggy-backed onto the beacon. The wireless signal then undergoes a certain *path loss*, which depends on the distance along which the signal travels. The opposite device will receive the beacon with a certain power P_r. In free space, a wireless signal traveling along a distance d between a sender and a receiver will be received with a power of

$$P_r = P_t \cdot G_t G_r \cdot \left(\frac{\lambda}{4\pi d} \right)^2 , \qquad (1)$$

see [2]. In Equation (1), λ is the wavelength of the signal and G_t and G_r are the gains of transmitter and receiver, respectively. G_t and G_r can be obtained by calibrating each smartphone model individually. Since P_r and P_t are known by the receiver, the distance d can be estimated.

This estimation is aggravated by multiple effects, of which the most important ones are the following.

- The antennas of both devices might be directional and hence, the orientation between both devices impacts P_t and P_r.
- Human tissue attenuates the signal by a considerably higher degree than free space. For example, the attenuation between the chest and the back of the human body has been reported as $19.2\,\mathrm{dB}$ [3]. As a result, the estimated distance is strongly impacted when human tissue is within the direct signal path, and hence also by how two human bodies are oriented relatively to each other, and where the phones are worn.
- Multipath propagation, e.g. reflections on metal surfaces, can interfere with the direct signal.

- The sensitivity of G_t and G_r depend on the wireless channel and hence frequency on which a beacon is received.

On smartphones, the received power of a BLE signal is available in the form of a received signal strength indicator (RSSI), which is provided by the Bluetooth radio. Distance estimation using the RSSI on smartphones has been studied thoroughly throughout the last years, e.g., in [4], [5]. Currently, due to the high relevance for contact tracing, analyzing and increasing the accuracy of distance estimation is receiving considerable attention by the scientific community [6], [7].

Channel-Dependent RSSI: This paper addresses the problem of the frequency dependence of the RSSI, which reduces the accuracy of distance estimation. In the BLE protocol, which is used for contact tracing on smartphones, advertising packets are sent on 3 different channels, which are spread over almost the entire frequency band used. These 3 channels use the center frequencies $2.402\,\text{GHz}$, $2.426\,\text{GHz}$ and $2.480\,\text{GHz}$ [8]. Because of the following three effects, the RSSI depends on the channel on which the packet was received.

1) Almost every device has frequency-dependent values of G_r and G_t. In other words, a packet sent on a certain channel (and hence frequency) will have a larger power than when being sent on a different channel, and the receiver will similarly sense different RSSI values for the same actual received power on different channels. This effect can lead to a RSSI estimation error of multiple dB, which can result in a distance estimation error of several meters. If it is known on which channel each packet is received, then G_r and G_t can be measured separately for each channel, and hence the error can be cancelled easily.

2) The path loss of a signal depends on the channel/frequency on which the signal is transmitted, see Equation (1). This effect can also easily be cancelled when the channel on which a packet was received is known.

3) Packets sent on different channels propagate differently in the environment. BLE signals are reflected, scattered and diffracted by objects in the surrounding. Hence, the signal reaching the receiving antenna consists of multiple replicas of the transmitted signal, which are called multipath components. These replica of the transmitted signal interfere with those transmitted along the direct path, i.e., the line-of-sight path. When interfering constructively, the RSSI increases, whereas it is reduced in the case of destructive interference. Thus, the RSSI accuracy might be drastically reduced due to the distorted received signal by multipath propagation.

These effects are shown in Fig. 1, which depicts multiple RSSI measurements for the 3 different channels used in BLE. Here, we used a Google Pixel 2 smartphone as a transmitter and a Google Pixel 3 as a receiver. Transmitter and receiver were placed in a height of $67\,\text{cm}$ in an indoor environment, and their distance was varied for each measurement. As can be
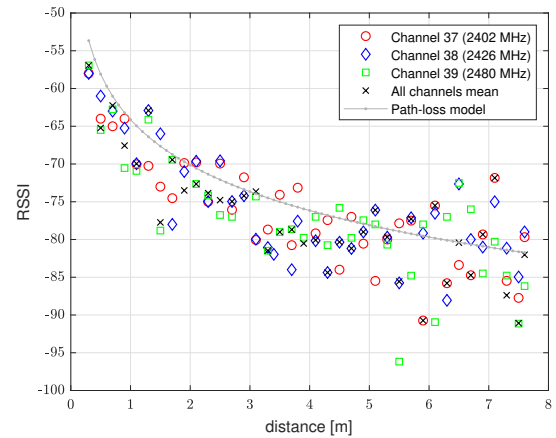

Fig. 1. Measured RSSI at a Google Pixel 3 smartphone for different distances. As a transmitter a Google Pixel 2 was used.

seen, for a given distance, the RSSI differs significantly among the 3 channels. In particular, we could observe differences of up to $15\,\text{dB}$. This occurred even for distances below $2\,\text{m}$, which are the most relevant ones for contact tracing.

In addition to measured RSSI values, Fig. 1 also depicts the path loss model as introduced in Equation (1), where G_t, G_r and P_t have been fitted to minimize the squared error between model and measurements. As already mentioned, the distance is usually estimated based on such a model, which assigns a distance to each RSSI value. When now estimating the distance based on the RSSI, an attenuation of $15\,\text{dB}$ due to the channel-dependent multipath propagation can lead to distance estimation errors in the order of tens of meters. Hence, it is of crucial importance to be aware of the channel on which a packet was received. If the channel was known, the estimation error due to frequency-selective hardware and free-space path loss could be cancelled using calibration. Furthermore, errors due to multipath propagation could be reduced by computing the average of the same number of RSSI values from different channels, or even by more advanced methods, e.g. [9].

Unavailability of Channel Information: The BLE radio does not relay the information on which channel a beacon was received to the smartphone operating system. Indeed, the BLE *host control interface*, which is used for data exchange between radio and smartphone, requires that incoming advertising packets are reported to the host without any channel information [8]. As a result, the frequency-dependent error cannot be mitigated. A potential method for reducing the error is averaging over multiple packets from different channels. However, without knowing on which channel a packet was received, it cannot be ensured that the same number of packets from each channel are used for averaging.

Proposed Solution: In this paper, we for the first time propose a technique to detect on which channel an advertising packet was received on an Android smartphone. We thereby exploit an undocumented behavior of many Bluetooth SoCs found in recent smartphones. In particular, consecutive reception windows are separated from each other by a period of time that is

significantly larger than the distance between two consecutive packets. After the scanning for incoming packets is activated by an application (app), we observe that on most smartphones we have tested, scanning starts on the same channel. Hence, we can exploit this behavior for obtaining the first channel on which the smartphone scans. We can then classify the channel of later received packets based on associating their reception time with the time windows during which the receiver listens for incoming packets and the corresponding channels.

Paper Organization: The rest of this paper is organized as follows. In Section II, we describe how the procedure used for contact tracing in BLE, called *advertising and scanning*, works. Next, in Section III, we describe our proposed technique for channel detection. We experimentally evaluate this technique in Section IV and conclude our findings in Section V.

II. Advertising and Scanning in BLE

A contact tracing app will use the following procedure for detecting devices in its surrounding. It is provided by the BLE protocol [8] and referred to as *advertising and scanning*.

Every device periodically schedules an *advertising event* once per T_a time-units. T_a is called the *advertising interval* and is composed of a static part $T_{a,0}$ plus a random delay $\rho \in [0, 10\,\mathrm{ms}]$. In each such event, three beacons in a row are sent. The first of them is sent on channel 37 (which corresponds to a center frequency of $2.402\,\mathrm{GHz}$), the second one on channel 38 ($2.426\,\mathrm{GHz}$) and the third one on channel 39 ($2.480\,\mathrm{GHz}$) [8]. This is depicted in Fig. 2. Here, every arrow stands for an advertising event. Since the 3 individual beacons in each event are sent within short amounts of time, we do not show them in the figure.

For being able to receive incoming beacons, every device also listens to the channel by using so-called *scan windows*. Every scan window has a duration of d_s time-units, and there is one such window every *scan interval* T_s. After every instance of the scan window, the channel for the succeeding window is toggled between channel 37, 38 and 39 in a round-robin fashion. This is also depicted in Fig. 2.

A device can detect the presence of another device, once a beacon from the remote device coincides with one of its scan windows [10]. Most values for (T_a, T_s, d_s) supported by Android fulfill $T_a < d_s$ (cf. Fig. 2) and hence, the reception of at least one beacon is guaranteed in each scan window. Note that the number of coinciding beacons might vary for every scan window. For example, in Fig. 2, the scan window on channel 38 has 3 coinciding beacons, whereas the window on channel 39 has 4.

The Android operating system does not allow an app to select these parameter values directly. Instead, an app can chose between three different settings that determine T_a and three different ones that determine T_s and d_s. These settings are listed in Table I.

It needs to be mentioned that there is no transparent mapping between these settings and the corresponding values. First, the values that correspond to a certain setting (e.g.,

SCAN_MODE_BALANCED) are not officially specified by Google. We have therefore obtained them from the source-code of the latest version of Android[1]. Second, the values that are actually used could differ from those given in Table I due to scheduling conflicts. In particular, the radio might maintain other Bluetooth connections, and the points in time at which other packets are exchanged might overlap with those needed for advertising and scanning. In addition, the Bluetooth radio is also used for WiFi on many devices, which could lead to additional scheduling conflicts. However, we found in our experiments that the values from Table I are actually used during normal operation, i.e., when no scheduling conflicts are present.

Android Setting	T_a [s]	T_s [s]	d_s [s]
SCAN_MODE_LOW_POWER	-	5.120	0.512
SCAN_MODE_BALANCED	-	4.096	1.024
SCAN_MODE_LOW_LATENCY	-	4.096	4.096
ADVERTISE_MODE_LOW_POWER	1.000	-	-
ADVERTISE_MODE_BALANCED	0.250	-	-
ADVERTISE_MODE_LOW_LATENCY	0.100	-	-

TABLE I
BLE PARAMETERIZATIONS IN ANDROID.

For the sake of completeness, we here also mention three other configuration options. First, the SCAN_MODE_OPPORTUNISTIC setting can be used by an app to obtain scan results when scanning has been triggered by a different app, without triggering the scanning itself. Second, a different set of values for T_s and d_s is available when using *batch scanning*, where multiple received packets are reported to the app jointly after some time instead of immediately after discovery. However, we could not find any documentation of this feature and hence did not study it in detail in this paper. Third, Google and Apple are currently drafting their *Exposure Notification* service [11]. Here, an advertising interval of $200\,\mathrm{ms}$ to $270\,\mathrm{ms}$ is specified, while no values for T_s and d_s are given. None of these additional options will supersede the need for channel detection, as we propose in this paper.

III. Channel Detection

In this section, we describe how the channel on which an incoming beacon is received can be detected on a smartphone. As already mentioned, the BLE radio does not relay this information to the smartphone's operating system, since the Bluetooth standard does not specify an interface for this.

According to the BLE specification [8], the channel on which the radio scans is toggled after *every* scan window. Thereby, the same order of channels 37, 38, 39, 37,... is always pursued.

Though the Bluetooth Low Energy (BLE) specification does not specify on which channel the radio has to scan first after its activation, we could observe on different smartphone models

---

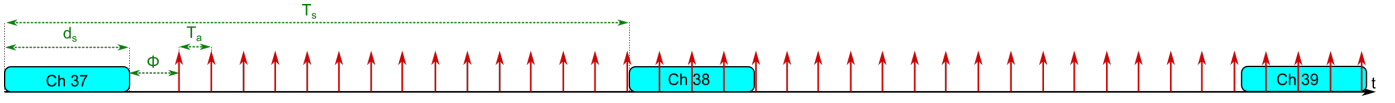[1]In older Android versions, the parameter values of the scan modes are different.

Fig. 2. Advertising and scanning in BLE. Arrows depict advertising events, which consist of a sequence of 3 packets on 3 different channels. The rectangles depict scan windows.

(see Section IV and Table III) that when scanning is activated, the device will always begin with channel 37.

Let the point in time at which scanning was activated be given by t. Then, incoming packets will only be received on channel $c \in \{37, 38, 39\}$, if their reception time falls within a time-interval $I_c(k) = [t_{l,c}(k), t_{r,c}(k)], k = 1, 2, 3, 4, ...$, with

$$
\begin{aligned}
t_{l,c}(k) &= t + 3 \cdot (k-1) \cdot T_s + (c-37) \cdot T_s \\
t_{r,c}(k) &= t + 3 \cdot (k-1) \cdot T_s + (c-37) \cdot T_s + d_s .
\end{aligned}
\tag{2}
$$

Equation (2) directly follows from Fig. 2. Therefore, we can detect the channel on which a beacon was received by classifying the time of each beacon reception into I_{37}, I_{38} or I_{39}.

The values for d_s and T_s can be obtained from Table I. However, recall that especially in case of scheduling conflicts, the phone might potentially deviate from this periodic scheme. Though we could observe that the values from Table I appear to be used in *most* cases (i.e., when no scheduling conflicts are present), there might potentially be (slight) changes of these parameter values, or even dropped scan windows or beacons. In our experiments, the scan windows always occurred at the expected points in time given by Equation 2, even when WiFi was activated. However, some of the transmitted beacons were not received, indicating that the scan windows were interrupted on a short-term basis for carrying out WiFi communication. This does not negatively impact channel detection, since the classification of reception times according to Equation (2) remains unaffected. The same holds true when beacon transmissions are omitted or their transmission times change due to scheduling conflicts.

Because the clocks of the smartphone and Bluetooth radio are not synchronized, they could drift against each other. This might disturb the channel detection based on Equation (2), since the classification is carried out within an app that relies on the clock of the smartphone, whereas the scan windows are scheduled using the clock of the radio. To mitigate the effects of this, we slightly modify the interval borders of $I_c(k)$ from Equation 2 to $\hat{I}_c(k) = [\hat{t}_{l,c}(k), \hat{t}_{r,c}(k)], k = 1, 2, 3, 4, ...$, with

$$
\begin{aligned}
\hat{t}_{l,c}(k) &= 3 \cdot (k-1) \cdot T_s + (c-37) \cdot T_s + t_g/2 \\
\hat{t}_{r,c}(k) &= t + 3 \cdot (k-1) \cdot T_s + (c-36) \cdot T_s - t_g/2 .
\end{aligned}
\tag{3}
$$

Hence, for realizing a more robust detection, we here classify each received packet by into which instance of T_s it falls, even when being received outside of the (estimated) scan window. Further, t_g is a guard time to compensate for the clock drift, for which we propose a concrete value in Section IV. In addition, for limiting this drift, we propose to regularly re-start the scanning after a certain period of time. We evaluate

after which amount of time such a re-start should occur in Section IV.

Algorithm 1 shows the pseudo-code of our proposed algorithm for detecting the channel on which a packet was received. In order to limit the power consumption of the app, the algorithm starts the BLE scanner using the SCAN_MODE_LOW_POWER setting (c.f. Line 4 in Algorithm 1). As soon as BLE signals are detected, the main part of the algorithm is executed and the BLE scanner is re-started using the SCAN_MODE_LOW_LATENCY setting (c.f. Line 9-12 in Algorithm 1). We remember the timestamp t when the scanning was re-started. Note that this timestamp does not perfectly coincide with the actual re-start of the BLE scanning, since there might be delays and/or jitter. The effect of such misalignments are mitigated through the guard time t_g in Equation (3). All subsequent packets are handled in the inner while loop starting in Line 13 of Algorithm 1. For each such packet, the reception time is computed and t (i.e., the time when scanning was started) is subtracted. The resulting time difference is used by the *ClassifyChannel()*-function, which infers the channel of reception by evaluating Equation (3). After *Max-Scan-Time* (cf. Line 16 in Algorithm 1) has passed, the scanning is re-started to limit the clock drift between smartphone and radio, as already explained. The guard time t_g compensates for any clock drift before this re-start.

In the next section, we evaluate the accuracy of channel detection using this algorithm on different smartphone models and for different re-start intervals of the BLE scanning procedure.

IV. EVALUATION

In order to validate our approach and Algorithm 1 for different smartphone models, we have set up the following test environment.

A. Experimental Setup

Our experimental setup was as follows. Four Raspberry Pis[2], denoted as R(i), $i = 1 \ldots 4$ continuously transmitted BLE advertising packets. Each Raspberry Pi was configured to transmit on a different set of channels, viz., R(1) on channel 37, R(2) on channel 38, R(3) on channel 39 and R(4) on all three channels, as also shown in Table II. The advertising interval was set to $100\,\text{ms}$, which is way below d_s (cf. Table I). Hence, multiple advertising packets will fall into every scan window. Note that $T_a = 100\,\text{ms}$ is used by the ADVERTISE_MODE_LOW_LATENCY setting on Android

---

[2]In our experiments with Samsung Galaxy S5 and Xiaomi MI-A2 smartphones, these Raspberry Pis were replaced by Bluegiga BLE112 radios, which carried out the same tasks as the Raspberry Pis.

Algorithm 1: Android BLE Channel-Detector

```
1  doInit = true;
2  while 1 do
3  │   if doInit then
4  │   │   Scan-Mode = SCAN_MODE_LOW_POWER;
5  │   │   Re-Start-Scan();
6  │   │   doInit = false;
7  │   │   Channel-Detection = false;
8  │   if BLE signals detected then
9  │   │   Channel-Detection = true;
10 │   │   Scan-Mode =
   │   │    SCAN_MODE_LOW_LATENCY;
11 │   │   Re-Start-Scan();
12 │   │   t = GetSystemTime();      // see Eq. (3)
13 │   while Channel-Detection do
14 │   │   if BLE signal received then
   │   │   │   /* classify BLE signal into
   │   │   │      Î₃₇, Î₃₈ or Î₃₉ using Eq. (3)
   │   │   │   */
15 │   │   │   ClassifyChannel(t, GetSystemTime());
16 │   │   if GetSystemTime() - t > Max-Scan-Time then
17 │   │   │   Re-Start-Scan();
18 │   │   │   t = GetSystemTime() ;  // see Eq. (3)
19 │   │   if No signals detected then
20 │   │   │   doInit = true;
21 │   │   │   break;
```

Raspberry Pi	Channel	Center Frequency in [GHz]
R(1)	37	2.402
R(2)	38	2.426
R(3)	39	2.480
R(4)	37,38,39	2.402, 2.426, 2.480

TABLE II
EXPERIMENTAL SETUP WITH 4 RASPBERRY PIS.

smartphones. Therefore, though having been obtained using Raspberry Pis as senders, these results remain valid when packets are sent using smartphones. It is worth mentioning that we have tested our proposed methodology also with all other advertising intervals supported by Android (viz., $100\,\mathrm{ms}$ and $1000\,\mathrm{ms}$), and found it to be working successfully irrespective of the value of T_a used.

In BLE, the services a device offers are advertised with the payload of its packets. Such services are identified by unique identifiers, so-called *Universally Unique Identifiers (UUIDs)*. We have assigned a different UUID to the packets of each Raspberry Pi.

The packets of $R(4)$, which are sent on all three channels, are received in every scan window. This allowed us to detect the individual scan windows of the smartphone. In addition, we could directly see on which channel a beacon was received, since we have encoded the channel information via the UUIDs of each device. In other words, since e.g., R(1) only advertises on channel 37 and we can detect that a packet was sent by R(1) using its UUID, we could easily infer on which channel the scanner was scanning.

B. Results

1) Behavior of the BLE Radio: Fig. 3 shows the reception times of all packets in an experiment using the

SCAN_MODE_LOW_LATENCY (top) and another experiment using the SCAN_MODE_BALANCED (bottom) setting. The individual measurements lie in such a close proximity that they appear as lines rather than as individual points. We have sorted the packets by their UUIDs, which identify their channel. The individual instances of each scan interval, starting from the point in time the scanning was started, are annotated using different background colors. Assuming that scanning started on channel 37, red indicates that in this interval, the scan window was listening on channel 37. Similarly, green corresponds to channel 38 and blue to channel 39.

In both experiments, the scanning was activated at time $t = 0\,\mathrm{s}$. In the SCAN_MODE_LOW_LATENCY setting, as can be seen, the received packets follow the pattern predicted by Equation (3), which is also exploited by Algorithm 1. In particular, starting from time $t = 0\,\mathrm{s}$, packets are received on channel 37 for T_s time units. Next, a sequence of packets is received on channel $38, 39, 37, 38, 39, \ldots$. In contrast, when using the SCAN_MODE_BALANCED setting, we found that the actual scanning procedure starts with a certain offset t_o from the starting time. Our measurements with different smartphone models showed that this offset varies significantly every time the scanning is started, and in addition depends on the smartphone model. This justifies that our algorithm is built upon the SCAN_MODE_LOW_LATENCY setting for detecting the channel of an incoming BLE signal, despite this setting being the most power-hungry one on Android devices. Since we only switch to this mode for short amounts of time after an initial reception (i.e., until the channel has been estimated and a sufficient number of packets for computing the distance have been received), the energy overhead of this will be acceptable. Recent results [12] show that - depending on the smartphone model - the battery of the smartphone is drained by between $5\,\%$ and $20\,\%$ earlier, when the SCAN_MODE_LOW_LATENCY setting is used during all times. Since we only use this mode for small fractions of the time, the reduction of the battery runtime will be way below this.

2) Detection Probability: While Fig. 3 shows that the behavior of the BLE radio on multiple smartphone models appears to be suitable for channel detection, which detection accuracy is achieved when using Algorithm 1? In order to evaluate the success rate of the channel detection algorithm, Fig. 4 shows the fraction of packets for which the channel was detected correctly as a function of the time since the last re-start of the scanning procedure for the Google Pixel 2 and Google Pixel 3 smartphones.

validated	Device Name	model	Android Version	comments
yes	Google Pixel 2	Pixel 2	10	-
yes	Google Pixel 3	Pixel 3	10	-
yes	OnePlus 5	ONEPLUS A5000	9	-
yes	Samsung Galaxy S5	SM-G900F	8.1.0	older API: $T_s = 5\,$s, Lineage OS.
yes	Xiaomi MI-A2	M1804D2SG	9	-
yes	Xiaomi Mi 9T Pro	M1903F11G	10	-
no	Huawei P20 lite	ANE-LX1	9	different scan intervals

TABLE III
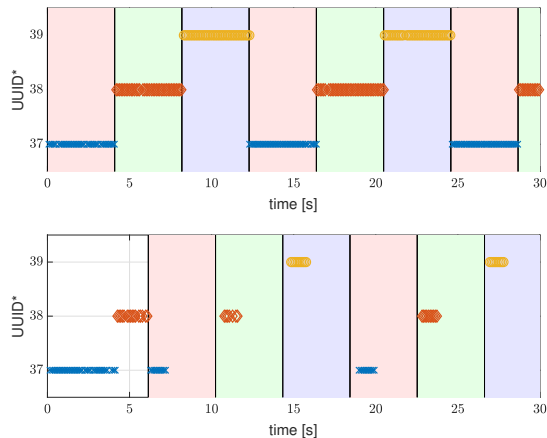TESTED ANDROID DEVICES, A FULL LIST OF DEVICES CAN BE FOUND IN [13]

Fig. 3. Recorded UUIDs* of the Raspberry Pis R(1), R(2) and R(3) versus time in seconds for the SCAN_MODE_LOW_LATENCY and SCAN_MODE_BALANCED settings. The different background colors indicate the channel of the scan intervals given by Equation (3).
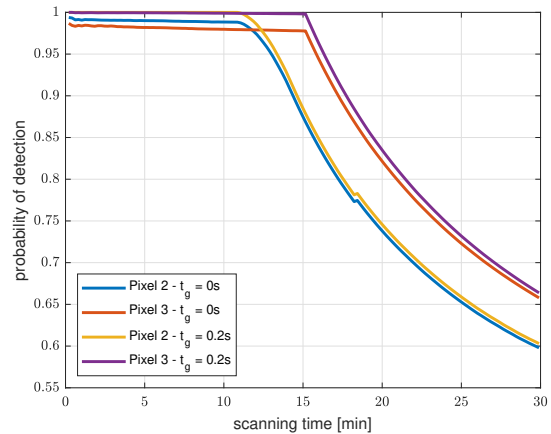
Fig. 4. Probability of classifying the channel correctly as a function of the scanning time without re-starting for the Pixel 2 and Pixel 3 smartphones. Two guard times $t_g = 0\,$s and $0.2\,$s were applied. The BLE scanner was re-started at $t = 0\,$s.

Both smartphones continuously recorded the received BLE signals of the Raspberry Pis for $24\,$h. Scanning was re-started every $30\,$min. This re-starting was necessary because the Android operating system automatically switches from the SCAN_MODE_LOW_LATENCY to the SCAN_MODE_OPPORTUNISTIC setting after $30\,$min of continuous scanning. In the SCAN_MODE_OPPORTUNISTIC setting, the device only schedules scan windows when a different app explicitly triggers the scanning (i.e., by using a different mode than SCAN_MODE_OPPORTUNISTIC), which would have interrupted our measurements.

We used a guard time of $t_g = 0\,$s and $t_g = 0.2\,$s, respectively. As can be seen in the figure, by using a guard time of $t_g = 0.2\,$s, we obtain a probability of detection of $100\,\%$ for more than $10\,$min of contiguous scanning without re-starting for the Pixel 3 smartphone, and $15\,$min for the Pixel 2 smartphone. For a guard time of $t_g = 0\,$s, the initial probability that we classify the channel correctly is slightly reduced to around $97\,\%$. When scanning is carried out for more than $10\,$min without reset, the detection probability gradually becomes smaller for both values of t_g. This is caused by clock drift between the smartphone and radio, as already described. We therefore propose to set t_g to $0.2\,$s and *Max-Scan-Time* in Algorithm 1 to $10\,$min.

3) Different Smartphone Models: We have shown that our proposed algorithm works in principle, but will it work for *all*

smartphone models from different manufacturers? For answering this question, we have tested different smartphone models for their compatibility with our proposed methodology. In particular, we have tested whether they always start scanning on channel 37 and then switch to the next channel after every instance of T_s. Table III summarizes the results of this experiment. Out of 7 smartphones from different manufacturers, Algorithm 1 is compatible with 6. The Samsung Galaxy S5 we have tested was using an older version of Android, in which the scan intervals that were actually used differ from the ones in the most recent Android version. Nevertheless, our proposed algorithm works successfully when adjusting T_s.

The Huawei P20 lite smartphone was the only one on which our proposed methodology did not work. In our experiments, we found that the Huawei P20 lite used a scan window between $100\,$ms and $200\,$ms. After every instance of the scan window, the channel for the succeeding window was toggled, however, the radio did not reliably start the scanning procedure on Channel 37. Hence, the algorithm was not able to classify the BLE signals into $\hat{I}_{37}$, $\hat{I}_{38}$ or $\hat{I}_{39}$ using Equation (3).

In addition to the evaluation data contained in this paper, we plan to test additional smartphone models for their scanning behavior in the future. For this reason, we have launched a website [13] on which we will list the outcomes of additional experiments. The website is available under https://www.dlr.de/kn/covid.

V. Conclusion

As required by the BLE specification, the information on which wireless channel a beacon has been received is not made available to a smartphone. Fortunately, as we have shown in this paper, the behavior of BLE radios used on recent smartphone models allows for reliably detecting the channel indirectly, since the channel after a re-start is always channel 37. Hence, we in this paper for the first time proposed a solution to reliably detect the wireless channel on which a packet has been received on a smartphone. We also showed that the channel on which the smartphone scans can be tracked over time by an Android application after re-starting the BLE scanning procedure. The proposed technique was experimentally evaluated on multiple different smartphone models. In particular, we showed that a probability of detecting the correct channel of 100% can be obtained for the Google Pixel 2 and Pixel 3 smartphones.

The information on which channel a packet has been received is highly relevant for distance estimation. e.g., in the context of contact tracing. In particular, the error induced by the frequency-dependent gains G_r and G_t, as well as the frequency dependent signal propagation in free space can be cancelled easily. Also the issue of multi-path propagation, which can occur especially in indoor environments, can be mitigated when the channel on which a packet was received is available. Therefore, we believe that our proposed technique will be helpful for distance estimation in contact tracing apps against COVID-19 and beyond.

References

[1] C. Günther, M. Günther, and D. Günther, "Tracing contacts to control the covid-19 pandemic," *arXiv preprint arXiv:2004.00517*, 2020.

[2] H. Friis, "A note on a simple transmission formula," *Proceedings of the IRE*, vol. 34, no. 5, pp. 254–256, may 1946.

[3] A. Alomainy, Y. Hao, A. Owadally, C. G. Parini, Y. Nechayev, C. C. Constantinou, and P. S. Hall, "Statistical analysis and performance evaluation for on-body radio propagation with microstrip patch antennas," *IEEE Transactions on Antennas and Propagation*, vol. 55, no. 1, pp. 245–248, 2007.

[4] T. Chowdhury, M. Rahman, A.-A. Parvez, A. Alam, A. Basher, A. Alam, and S. Rizwan, "A multi-step approach for RSSI-based distance estimation using smartphones," in *IEEE International Conference on Networking Systems and Security (NSysS)*, 2015, pp. 1–5.

[5] S. Liu and A. Striegel, "Accurate extraction of face-to-face proximity using smartphones and Bluetooth," in *IEEE International Conference on Computer Communications and Networks (ICCCN)*, 2011, pp. 1–5.

[6] F. Sattler, J. Ma, P. Wagner, D. Neumann, M. Wenzel, R. Schäfer, W. Samek, K.-R. Müller, and T. Wiegand, "Risk estimation of sars-cov-2 transmission from bluetooth low energy measurements," *arXiv preprint arXiv:2004.11841*, 2020.

[7] D. J. Leith and S. Farrell, "Coronavirus contact tracing: Evaluating the potential of using Bluetooth received signal strength for proximity detection," 2020.

[8] Bluetooth SIG, "Specification of the Bluetooth system 5.2," December 2019, volume 0, available via bluetooth.org.

[9] C. Gentner, T. Jost, W. Wang, S. Zhang, A. Dammann, and U.-C. Fiebig, "Multipath assisted positioning with simultaneous localization and mapping," *IEEE Trans. Wireless Commun.*, vol. 15, no. 9, pp. 6104–6117, Sep. 2016.

[10] P. Kindt, M. Saur, M. Balszun, and S. Chakraborty, "Neighbor discovery latency in ble-like protocols," *IEEE Transactions on Mobile Computing (TMC)*, vol. 17, no. 3, pp. 617–631, 2018.

[11] Apple, Google, "Exposure notification bluetooth specification," April 2020, available via https://blog.google/documents/70/Exposure_Notification_-_Bluetooth_Specification_v1.2.2.pdf.

[12] P. Kindt, T. Chakraborty, and S. Chakraborty, "How reliable is smartphone-based electronic contact tracing for covid-19?" 2020.

[13] "Android Received Channel Detection Website," https://www.dlr.de/kn/covid.