WiPush: Opportunistic Notifications over WiFi without Association

Utku Günay Acer
Nokia Bell Labs
Antwerp, Belgium
utku_gunay.acer@nokia-bell-labs.com

Otto Waltari
Department of Computer Science, University of Helsinki
Helsinki, Finland
otto.waltari@helsinki.fi

ABSTRACT

Push notifications have become a prevalent way of communication for user devices to receive updates about their online profiles. State-of-the-art mechanisms for push notification delivery do not take users' spatio-temporal context into account as it would require a user device to retrieve the location and deliver it to the app servers, both of which are costly in terms of energy and traffic. In this paper, we provide an opportunistic, local, network-centric mechanism, *WiPush*, to deliver contextual notifications, leveraging the high density of WiFi access points (AP). WiPush does not require association with APs and uses WiFi public action frames to deliver notifications without abusing the underlying standards. Our evaluation shows that while WiPush can deliver short notifications to the users with high reliability without affecting the networking functionalities of the AP adversely, it can not be used to transfer large volumes of data.

CCS CONCEPTS

• **Information systems** → **Location based services**; Mobile information processing systems; • **Human-centered computing** → **Ubiquitous computing**; **Mobile computing**; • **Networks** → *Mobile networks*; *Wireless local area networks*; Link-layer protocols;

KEYWORDS

Spatio-temporal context, push notifications, contextual notifications, WiFi, opportunistic delivery

ACM Reference Format:
Utku Günay Acer and Otto Waltari. 2017. WiPush: Opportunistic Notifications over WiFi without Association. In *Proceedings of Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous).* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3144457.3144492

1 INTRODUCTION

With the proliferation of smart phones and wearables, a barrage of new services is provided to users. These services use the history of users' online activity and the resources provided by the device such as sensors to collect information about them and their environment including their location to provide better service.

Smart devices have shifted the way users interact with online content. Push notifications are small and succinct messages used by mobile applications to inform users of new events and updates allowing the third party content providers to initiate communication with the users even when the application is not actively being used. This type of communication provides a visual or audible cue to inform the mobile users about new unattended short messages or event updates. This way, users no longer need to open apps or browsers to pull new content from the servers.

As mobile devices leverage their ubiquity and allow information to reach them at any time and place, spatio-temporal notifications, that is providing contextual information, has become ever more important. For example, a retail store might offer coupons to customers that are present at a shop at a time, a city government might want to warn citizens in a park when the pollen count is high at that location, or the users might be asked to fill qualitative queries about the services provided to them when they visit a mall, train station, airport, museum, etc.

In order to receive notifications based on the spatio-temporal context, state-of-the-art systems require a device to compute its location and communicate it with the remote application server continuously, even when the application does not run in the foreground. However, OS providers rather limit the number of connections especially for apps running in the background. Besides, some smart devices such as wearables do not typically include a cellular interface and/or a GPS sensor and can not notify its location unless it is paired with another device such a smartphone. When no other device is available, these devices can not compute the location or send it to the server. In addition, computing the location using GPS consumes a high amount of energy. The OS provider may inform the third party app servers about the location of the users through the push notification channels between the app servers and the notification servers; however, to the best of our knowledge, the OS providers do not report the location of the devices to the app servers.

The high density of WiFi Access Points (APs) presents an opportunity for delivering spatio-temporal notifications possible. As of 2014, 42% of IP traffic is terminated at a device connected to a WiFi network and by 2019 it is predicted that WiFi will account for 53% of the IP traffic and 63% of the global Internet traffic [2]. Examples such as ESP[1] and Intel Edison[2] demonstrate that WiFi chips can be embedded inexpensively in small form factors. Besides the popularity and high coverage, WiFi APs intrinsically have access

MobiQuitous, November 8–10, 2017, Melbourne, VIC, Australia

ACM ISBN 978-1-4503-5368-7/17/11…$15.00
https://doi.org/10.1145/3144457.3144492

[1] http://www.esp8266.com
[2] https://software.intel.com/en-us/iot/hardware/edison

to fine-grained user location information. As the user devices with WiFi interfaces roam around, they occasionally scan the wireless medium for available WiFi networks. This way, APs are able to pinpoint the location of the device within the range of wireless signals.

One major drawback of using WiFi for notification delivery is, despite its ubiquity, that it requires a device to *associate* with an AP before it can send or receive any data [1]. At least the first association of a device with an AP requires manual action from the user. This may include selecting the AP from a list of available WiFi networks, entering a password or navigating through a landing page to accept terms of use and press a "connect" button. Public APs in particular typically require recurring input through user interference to remain connected or associated, even though free Internet access is provided. On wearable devices and smart objects, little or no user interface controls are available for entering a password or navigating through a landing page. Hence, these devices typically connect to a network through a master device such as a smartphone.

In this paper, we propose *WiPush*, a best-effort ubiquitous messaging layer built on top of WiFi that allows an AP to deliver contextual notifications to user devices *without the need for being associated*. We argue that only a pure network-driven solution can address efficient spatio-temporal notification delivery without compromising mobile energy usage or requiring dedicated applications with continuous Internet access.

With WiPush, the content providers have no financial barriers for introducing their messages to relevant users. They do not have to pay third party suppliers for the delivery service and rely on the dedicated apps which likely differs for each service. The notification can be created locally and directly delivered through APs in their premises. The users, on the other hand, receive most relevant notifications from local APs even if their device do not have access to the Internet through another connection.

WiPush relies on WiFi public action frames to push small notifications from the AP to the device. Opposed to existing work, we do not abuse the WiFi standard, but adhere to it. WiPush allows notifications to be delivered to specific devices within reach of the AP. It is lightweight and it does not alter the power management mechanism of the WiFi interface to receive notifications and hence costs little energy to receive notifications. With WiPush, a device can receive the notification from an AP even if it is associated with another AP. On the other hand, WiPush is not a full communication stack. It cannot be used for large volumes of data traffic. Instead, it is used to deliver spatio-temporal, contextual, push notifications to the devices.

We have tested WiPush using an off-the-shelf smartphone and an embedded computer as the user device and OpenWRT router as an AP. Though the actual figures depend on the underlying hardware and driver that could boost the delivery of the physical signals, WiPush achieves high delivery ratio and little latency in both devices where a small number of notifications are delivered and where there is a moderate traffic load on the WiFi medium. The energy consumption of WiPush is negligible in comparison to the case where they do not receive any notifications at all. Our results also show that the addition of WiPush does not hinder the networking functionalities of the AP.

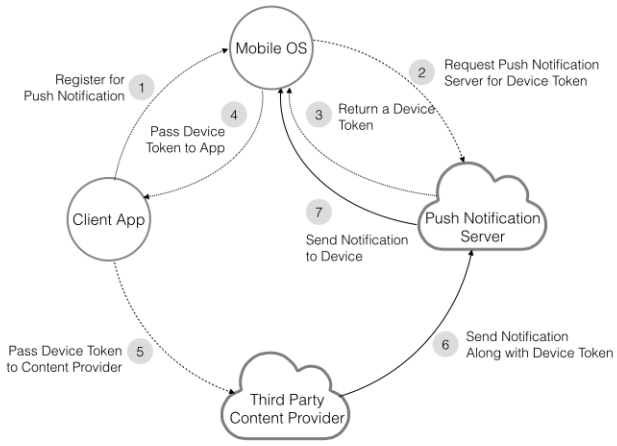

Figure 1: Architecture of Push Notification Service

2 BACKGROUND

2.1 Mobile Push Notification Service

Mobile push notification service provides a simple, lightweight mechanism to tell mobile applications to contact the server directly, to fetch the updated application or user data. Mobile operating system providers facilitate this service through dedicated notification servers, e.g., Apple Push Notification Server (APNs), Google Cloud Messaging Server (GCMs), etc.

Figure 1 illustrates interactions in the push notification ecosystem where the stages are labeled numerically. While stage 1-6 describe an initialization and security handshake between the third content provider and the application, the delivery of notifications occurs in stage 7, through a persistently maintained connection between the notification server and the mobile device. In order to keep this connection alive, the device periodically sends a keep-alive message. The device may report its location in these messages. The size of the typical payload of notification messages is fairly small, i.e. less than 4 Kbytes.

2.2 Applications of Push Notifications

Several mechanisms can be used for delivering contextual push notifications.

2.2.1 Cloud-based notifications. Cloud-based spatio-temporal notifications could be achieved directly by the app servers or through push notification servers such as GCM and APN. Either case, localization has to be performed in the device through, which requires considerable amount of energy. In addition, an active internet connection is necessary. While a WiFi connection is not always possible if the device does not recognize any APs close by, a considerable number of devices, wearables in particular, lack a cellular interface. Even when this is not the case, it has been shown that delivering push notifications over the cellular connection is highly inefficient in terms of power consumption [3]. While it is technically possible for notification servers to share location information with 3rd party app providers, to the best of our knowledge, major OS providers

do not send such information to app providers. Hence, apps need to report the location information themselves through occasional or persistently maintained connections that lead to high energy consumption both for computing the location and reporting it.

2.2.2 Cell-based notifications. In several countries, the cellular network is leveraged by governments as a public safety system to alert citizens about critical emergency situations (e.g., alerts involving imminent threats to safety or amber alerts). These alerts are geographically pushed to cell towers in the location of an emergency and broadcast in text-like messages to connected phones. Given a phone is typically connected to the cellular network, listening for SMS messages does not demand extra energy from a device (as the device does this anyway). However, like cloud-based notifications, this approach excludes a wide range of devices without a cellular interface. In additions, such messages can only be targeted to a coarse area, and not to a fine-grained geographical location.

2.2.3 Beacon-based notifications. The rise of iBeacon technology in recent years has led to several use cases for hyper-local notifications such as flash sales, airport navigation, information provisioning in exhibits, home automation, etc. Leveraging the Bluetooth Low Energy (BLE) standard, iBeacons advertise their presence via broadcasted UUID and major, minor identifier pair. However, the initiative to interpret these identifiers remains with the end-user devices and often involves an online look-up of data that is associated with a particular beacon. Consequently, iBeacons need to be physically embedded in an environment and devices need a dedicated application (with Internet access) to translate a sensed iBeacon's identifiers into meaningful information.

Alternative to Bluetooth is WiFi beacon stuffing where APs could simulate beacon messages to unassociated devices [4]. To do that, the AP abuses the 802.11 standard to overwrite data on SSID, BSSID or vendor-specific fields of beacon frames. Altering these fields lead to negative consequences. For example, devices no longer recognize an AP because SSID is not advertized in the beacon.

3 DESIGN CHALLENGES

With WiPush, we aim to overcome the present limitations of spatio-temporal push notifications discussed earlier, by pushing short messages over ubiquitous WiFi without the need to be associated with the WiFi network or any other type of wireless network. To this end, we put forward three major design challenges.

3.1 DC1: Compliance with the existing WiFi specification

It is critical that WiPush does not break the existing standards nor jeopardize the traditional operation of the WiFi protocol. We transparently upgrade WiFi stack to support push notifications with backward compatibility. For such upgrade to be feasible, both a device and an AP should support WiPush. Hence, a device and an AP should become aware of each other's capabilities to assess whether a notification delivery is possible. In addition, it is essential that no message is used outside scope of its original purpose by embedding the notification in a field that is not supposed to carry such information.

3.2 DC2: Directed notification messages

With WiFi, data can be broadcast to every device within the vicinity of the source AP or to a specific device. Broadcasting notifications to every user does not allow personalization. As a user may be not interested in every notification that is available, the AP should not send every notification to every user device. Therefore, it is necessary to send directed notifications.

3.3 DC3: Minimal energy expenditure

To minimize the energy footprint, mobile device needs to use radio time wisely. Battery operated smart devices turn off their WiFi interface as long as they can to conserve energy. During this time the WiFi interface is asleep, and hence the device cannot send or receive any data. Therefore, directed messaging requires some form of coordination of radio time between a device (listening) and an AP (transmitting) and not alter the duty cycle of the WiFi interface.

4 WIPUSH: OPPORTUNISTIC NOTIFICATION DELIVERY

In this section, we provide the service description of WiPush. Using the underlying properties of WiFi networks, we explain how an AP delivers notifications to the user devices. We then explain how the user preferences can be managed for notifications that they do not wish to receive.

4.1 Primer on IEEE 802.11 Standard

WiFi networks or Wireless Local Area Networks (WLAN) are governed by the IEEE 802.11 standard [1]. In a WLAN, each device that transmits or receives information is called a station (STA) and an access point (AP), which is a special kind of STA, provides connectivity to other stations. The set of STAs that an AP serves is called Basic Service Set (BSS).

Each BSS operates on a WiFi channel that is determined by the AP. The set of available channels differs across countries and continents. A STA can communicate with an AP only if they are on the same channel.

WiFi network operation is achieved through three types of Layer 2 frames. *Data frames* transport traffic from higher layers such as application data. *Control frames* police the STAs to access the wireless medium without causing a collusion, i.e. two devices transmitting at the same time. *Management frames* are used to provide and maintain connectivity to the STAs. Management frames and their purposes include authentication frames to securely connect to an AP, association request frames to receive resources from the AP to send and receive data, beacon frames for AP to announce its presence and its service set identifier (SSID), probe requests for STAs to scan surrounding APs, probe responses for APs to announce their presence to the scanning STAs, etc. A STA can send/receive management frames to/from an AP if it is not associated with it and even when it is associated with another AP.

The format of management frames is shown in Fig. 2. The Frame Control sequence yields the type and subtype of the frame. Addr 1 and Addr 2 yield the MAC address of the receiver and the transmitter of the frame, respectively.

The management frames carry *information elements* in their variable length frame body. Information elements are formulated

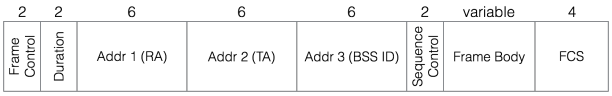

Figure 2: IEEE 802.11 Management Frame Type

using type-length-value (TLV) format. Each information element has a specific one-octet ID that is carried in `type`. For example, SSID element uses type 0. The `length` also uses 1 octet space and denotes the number of bytes that `value` takes. In other words, the value can take as much as 255 octets.

4.2 WiFi Power Management

With power management, devices turn off the power to the system components. When such components are not actively used, these components *sleep*, i.e. they run in a lower power state. WiFi chipsets also feature power management. When the WiFi interface is in sleep state, the STA cannot send or receive any frames.

When a STA is not associated with an AP, the WiFi interface typically remains in this sleep mode. Occasionally, the STA powers the WiFi interface, i.e. *wakes it up*, to scan available WiFi networks, passively or actively. In passive scanning, STAs listen to every channel and wait for beacon frames that are periodically sent by the APs, typically every 100ms. When a device scans actively, it sends a probe request on every channel and listens for probe responses from the APs on those channels. State-of-the-art devices typically use active scanning as it takes long to finish passive scan. Once the scan is complete and if the STA does not recognize any APs to associate with, the interface goes back to power saving sleep state.

Associated devices also perform power management when WiFi interface is inactive for a certain duration. If the device plans to switch to sleep state, it sends a *null* data frame without any payload. The Frame Control sequence in this case sets the power management bit to 1. The AP that receives this message will buffer all data frames to be received by this STA until its WiFi interface wakes up. This is also indicated by a null data frame with with power management bit set to 0. The STA also sends a null data frame with power management bit set to 1 when it scans WiFi networks. During this scan, the AP buffers data frames as well until the scan is complete and the STA is back on the channel of the AP [7].

4.3 Notification Delivery

In our design, we leverage the active scans performed by the devices that also wakes the WiFi interface. When a STA sends a probe request on a channel, the WiFi interface remains on that channel for a short period to allow close by APs to receive the probe request and send a probe response back to STA. Then, the STA *hops* to another channel to discover APs operating on that channel. This short window after receiving a probe request from a STA and discovering it, gives an opportunity to the AP to send a push notification to the device. We take advantage of this property in WiPush design.

The protocol for delivering WiPush notifications is given in Figure 3. Once an AP discovers a STA by receiving a probe request that indicates that the STA is capable of WiPush, it piggybacks the spatio-temporal push notification to the probe response. The probe request emitted by a device features an information element that

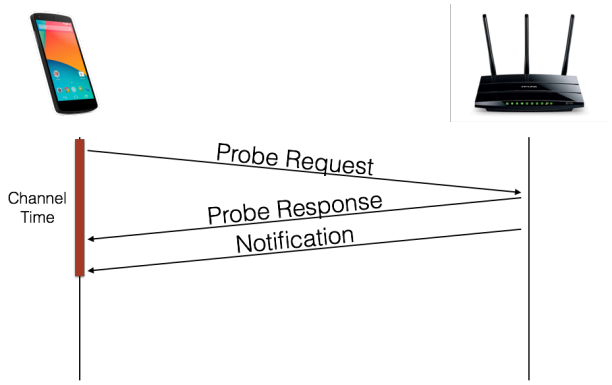

Figure 3: Protocol for Delivering WiPush Notifications

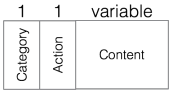

Figure 4: Format of Action Frame Body

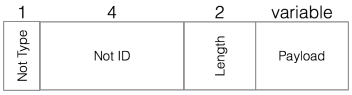

Figure 5: Notification Content

indicates the device's capability to receive WiPush notifications to the AP. We use one of the reserved information element ids to advertise this capability.

Unlike state-of-the-art notification delivery mechanisms such as APNS and GCM that rely on TCP based connections, WiPush is a best-effort mechanism. It does not guarantee the delivery of notifications. No feedback mechanism is employed to make sure a mobile device receives a notification sent by the AP.

We use *Public Action frames* to send contextual push notifications from the AP to the STAs. Action frames are used for various purposes including spectrum management, radio measurement, etc. The particular type of action frame is denoted in a 1-octet `Category` field as shown in Figure 4. In a Public action frame, this field is equal to 4 and is immediately followed by a 1-octet `Action` field. There are currently 16 values for `Action` field. In our system, we use one of these reserved values reserved for future use.

The content of the public action frame is given in Figure 5. Forming a notification, the AP first puts the notification type on the content of the public action frame, which is followed by 4-octet ID of the notification maintained by the AP. The MAC address of the AP (retrieved from the MAC header) and this ID distinguishes every notification in the STA. The frame provides the length of the notification payload in 2 octets, which is enough to represent the maximum length of a 802.11 frame, 2.3 KByte. The length field is followed by the payload of the notification.

The notification type indicates whether the AP requests a response from the STA. A crowdsourcing notification may offer the

user a number of options and request a selection. The type occupies one octet value that may hold other values in the future.

If the notification to the STA requires a response, the STA also similarly forms a public action frame once the user provides the input, albeit with a different `Action` value that indicates the message is a response. In the content, the STA includes the notification ID, the ID of the original notification received from the AP.

Note that it is possible to add the capability to send larger notifications that are fragmented to multiple frames using another type value. Moreover, the contents of the notification can be encrypted so that third parties cannot sniff and encode the notification.

4.4 Feasibility of WiPush

Past research suggests that people enable the WiFi interface in their smart devices most of the time [5, 12]. Hence, considering its extensive coverage, WiFi APs are well situated to transfer notifications related to the spatio-temporal context to user devices.

The device discovery in the AP leverages active scans by the devices. These scans indicate when devices allocate radio time to listening to messages originating from nearby APs. The scanning frequency or the time interval between successive scans depend on a number of factors including OS version, device manufacturer, WiFi chipset vendor and user interaction with device [6, 14]. In particular, we are interested in a device's scanning behavior while being unassociated with a WiFi AP and its display is off. The frequency by which a device scans the APs close by gives a first indication whether WiPush is feasible.

Table 1: Active scanning intervals of contemporary mobile devices.

Device	OS Version	Scanning Interval Average (s)	Scanning Interval RSD
iPhone 5	iOS 8.3	47	23.8%
iPhone 4	iOS 7.1.1	45	41.5%
Nexus 5	Android 5.1.1	57	88.4%
Galaxy S5	Android 5.0	58	37.9%
Nexus S	Android 4.1.2	290	13.7%

In Table 1, we provide characteristics of several modern smartphones as reported by Waltari and Kangasharju [14]. The table shows the average time between two scans from a number of devices and the relative standard deviation (RSD) that indicates how much the interval lengths vary over time. In each active scanning round, the device's WiFi chip is shortly tuned to listening mode so that messages from the APs can be received. Hence, for the tested devices, this means that an AP has an opportunity to reach out to a device every 47 (e.g., iPhone 5) or 57 (e.g., Nexus 5) seconds.

With respect to design challenges laid out in Section 3, WiPush copes with DC1 as no frame is used outside the scope of its intended use. Because the push notifications are encapsulated in public action frames that are sent to individual users addressed in the MAC header, WiPush complies with the DC2. Since it does not alter the power management mechanism of the WiFi interface and leverages the small time interval that the device utilizes for receiving probe responses when the device actively scans WiFi networks, DC3 is satisfied as well.

5 MANAGING USER PREFERENCES

Since interest in some types of notifications is highly subjective, it is vital to incorporate spatiotemporal user preferences. These preferences specify per location in which notifications a user is interested and how frequently. The user should be notified only by the information that matches her preferences. We consider three options for managing these preferences and filtering out unwanted notifications from the user's attention. We now outweigh their pros and cons:

On the device: This is the most straightforward case where the management of preferences of notification messages happens on the device itself. The APs send all notifications to a discovered device. Upon receiving a notification, the device checks whether its type matches the user's interests and preferences. The notification is displayed to the user only if the user is interested with the type of the notification.

On the cloud: Here, the user preferences are maintained in a centralized repository. Once an AP discovers a device by receiving probe request, it contacts the cloud server to retrieve the preferences of the user. Once it finds out what the user is interested in, the AP can check every notification to see if it is relevant to the user. This procedure takes time preventing the AP from sending the push notification right after receiving the probe request. Therefore, the AP either needs to wait until the next time the device performs a scan or schedule the device to make a transmission to indicate that AP can send the push notification.

On the AP: In this case, the user preferences are stored on the WiFi APs in a distributed way. Once the user updates its preferences, all or some APs are notified about this change through a central server. Note that, the centralized entity does not store the preferences itself but only forwards them to a subset of APs that participate in the service.

Filtering notifications locally in the device is the most privacy-aware mechanism as no other entity knows about the users' preferences. However, it comes at the cost of increased processing at the user device, which depletes the battery of the device faster. On the other hand, storing user preferences in the cloud means that an AP queries a centralized location every time it discovers a user device, which raises privacy concerns as it is possible to track drown the user movement by following such queries.

We argue that the filtering notifications on the AP offers a better alternative. The user location is already known to the AP due to reception of probe requests and the discovery information is not disseminated elsewhere. Hence, the risk of being tracked is not changed. With users empowered on the selection of the APs that maintain their preferences, malicious APs can be prevented from accessing user information. In addition, since the preferences are maintained locally the AP can promptly filter the notifications according to user interests and send it to the device. On the other hand, this solution requires a local database that stores the user preferences.

6 IMPLEMENTATION

We implement WiPush in *wpa_supplicant* daemon in user devices and in the *hostapd* daemon in APs, both working in the background and on the user space[3]. They handle the management of WiFi network, i.e. discovery, authentication, association and as such.

On AP, we also implement a queue that contains messages for the stations and a rule database that indicates the type of notifications that the user is interested in. Notifications are entered locally by the system administrators with a time-to-live value that indicates for how long the notification is valid. Once the notification expires, it is removed from the queue. Once a user is discovered, the AP checks whether it maintains any rules for the particular user. If so, it checks every notification whether the user is interested in such context. The AP immediately sends every relevant notification to user.

On the user device, we implement a simple application that displays the notifications received through the wpa_supplicant daemon to the user.

Development for both daemons is based on version v2.5-devel. Our implementation featured the same version on Raspberry Pi as well. In total, it took 3598 lines of C code to add WiPush functionality both daemons, whereas the entire code base consists of roughly 300K lines of code.

7 EVALUATION

In this section, we evaluate WiPush. Due to space constraints, our evaluation is limited to networking and device aspects. We plan to evaluate the end-to-end system including maintaining user preferences in multiple APs in future work.

We use a TP-Link TL-WDR4310 router that runs OpenWRT 15.05 Chaos Calmer[4] as AP. We used a Nexus 5x smartphone running Android 6.0.2 via Android Open Source Project[5] and a Raspberry Pi[6] running Raspbian Jessie as user devices to receive WiPush notifications, referred to as Nexus and RPi, respectively. Nexus has a driver and antenna designed by the vendor whereas RPi uses an off-the-shelf WiFi dongle to send/receive data that uses Ralink RT2800USB Wireless LAN driver[7].

The AP works on two bands, channel 1 on the 2.4 GHz band and channel 36 on the 5 GHz band, each runs a separate instance of hostapd daemon. We attach WiPush service to the hostapd managing the WiFi network on channel 1.

7.1 Delivery and Latency

Here, we look at WiPush performance in terms of message delivery. In order to do this, we automatically generate a number of notifications in the AP that are to be delivered to each device. To do these tests, we place both the Nexus and RPi about 10 meters away from the AP.

The delivery ratio yields the rate of messages received by a device to the number of notification generated in the AP. Since the notifications can be received only when the device scans WiFi networks, the latency gives the time between the scan command is issued to the device driver, rather than the notification is introduced to the AP, and the time that the notification is received.

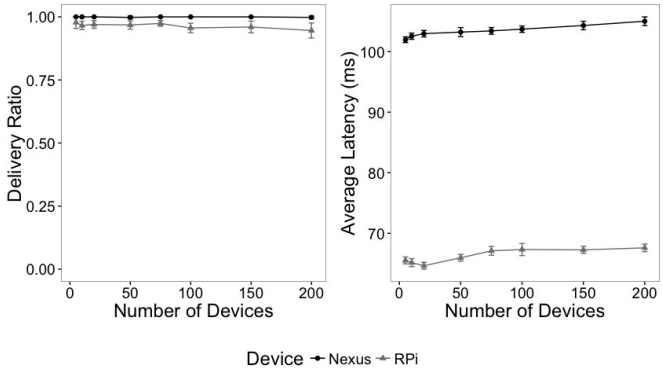

Figure 6: Performance with respect to the number of WiPush user devices. Regardless of user density, WiPush delivers a high ratio of notifications with small latency to both devices.

For convenience in terms of time to finish a test run, we force both devices to scan WiFi networks every 10 seconds since unmodified scanning interval of two devices are different. For example, RPi continuously sends probe requests until it is associated. We kept the interval to a low value for convenience to finish runs faster.

In each run, we let the devices scan WiFi networks 100 times that are used by the AP to send notifications. For every scenario, results are averaged over 5 runs. The vertical bars in the graphs correspond to the 95% confidence intervals.

7.1.1 Number of Devices. We first evaluate how the performance changes with the number of devices capable of WiPush to see how efficient AP is serving a large number of users with WiPush notifications. In order to mimic other devices that stem the AP to send notifications, we use another Raspberry Pi that artificially injects probe requests into channel 1 with randomly generated MAC addresses using libtins[8] library. For every MAC address, this device transmits a probe request every 10 seconds to make the AP send a notification to the address. The probes injected to the medium are separated by uniformly distributed intervals based on the number of MAC addresses. Since the AP has no way of knowing whether probe requests come from actual devices or fake addresses, it will send notifications to every MAC address in the Addr 2 field (transmitter address) of the received probe request.

Figure 6 shows how the performance changes in both Nexus and RPi with the increasing number of other WiPush devices that receive notifications from the AP. Though the delivery ratio for RPi slightly decreases with the density of users, it is consistently over 90%. For Nexus, the ratio is consistently around 100%.

Note that latency values take the instant a scan command is issued to the driver instead of the instant the device sends a probe request because user space programs do not have access to the

[3] http://w1.fi
[4] https://openwrt.org
[5] https://source.android.com
[6] https://www.raspberrypi.org
[7] https://wiki.debian.org/rt2800usb
[8] http://libtins.github.io

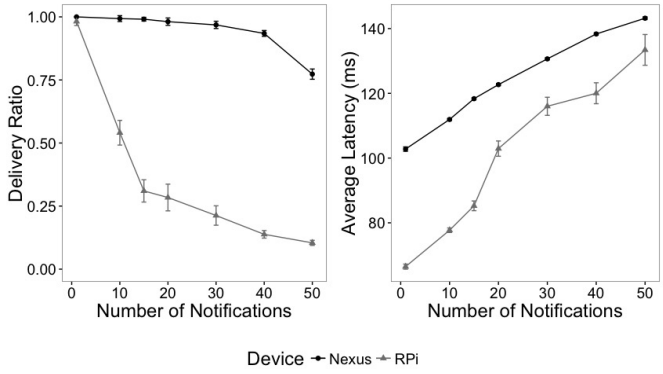

Figure 7: Performance with respect to the number of notifications. Regardless of user density, WiPush delivers a high ratio of notifications with small latency to the both devices.

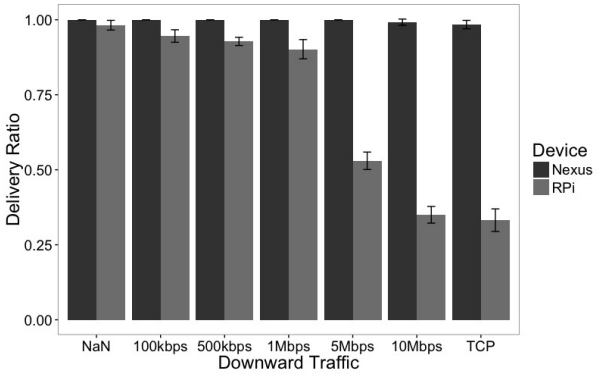

Figure 8: Delivery ratio with respect to background traffic. RPi performs badly with high load due to shortcomings of the WiFi driver whereas Nexus receives most of the notifications.

actual time that probes are sent during a scan. Notification delivery latency is larger for Nexus than that of RPi. This is because Nexus employs power management for its WiFi interface while RPi does not since it is not battery operated. So, for each scan, Nexus spends time to power on its WiFi interface. After the scan, being not associated with an AP, the device goes back to sleep mode. The latency increases with the number of devices for both devices since it is more likely for the AP to process and transmit notifications for other devices and the AP cannot send two notifications at same time.

7.1.2 Number of Notifications. Figure 7 shows how the performance of WiPush changes with the number of notifications sent by the AP at every scan by a WiPush capable user device. While it is possible to aggregate several notifications into one frame, we use a single frame for every notification in this scenario to assess the effect of number transmitted frames.

As 802.11 standard allows only one frame to be transmitted at a time in order to prevent collisions, notifications can only be sent in sequence. Hence, it takes longer to transmit and receive a larger number of notifications for both devices. As in the previous case, the latency for Nexus is larger than RPi since the device also spends time for awaking the WiFi interface.

Because the devices have a limited time window for receiving probe responses during the scans before hopping to another channel, a window that is exploited for receiving WiPush notifications, any notification that is transmitted outside this window cannot be received by the device. Hence, delivery ratio decreases with the increasing number of frames that carry the notifications at a scan. The decrease in the delivery ratio is more dramatic for RPi as Nexus's drivers and antenna outperform those of RPi.

The lack of ability to receive a large number of messages at a time also prevents WiPush from being used for general purpose data traffic.

7.1.3 Background Traffic. WiPush leverages the existing WiFi infrastructure where APs provide Internet connectivity to user devices. Traffic flows from/to such devices through APs. In this section, we capture how the background traffic influences WiPush performance. In order to emulate the traffic that flows through the AP, we use two other Raspberry Pi devices. One of them, RPi1, is connected to the AP through a wired, Ethernet connection and the other one, RPi2, is connected to the WiFi network served by the AP. Two other devices, Nexus and RPi, receive WiPush notifications from the AP. We generate artificial TCP or UDP traffic flows between RPi1 and RPi2 via iperf3[9] tool. With UDP traffic, both RPi1 and RPi2 generate traffic; however, the rate of the downward traffic (from RPi1 to RPi2) is 5 times as high as the upward traffic from RPi2 to RPi1. For TCP traffic, we initiate 4 parallel streams from RPi1 to RPi2 with a rate as high as possible.

The UDP datagrams and TCP packets are encapsulated in about 1600 byte data frames. The time interval between two successive data frames decreases as the data rate gets higher. This time interval is critical for the performance of WiPush. In transmissions of 802.11 frames, stations follow the Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) scheme in which STAs sense the wireless medium and transmit a frame only if the channel is idle. Before sending data frames, stations also exchange RTS/CTS frames to make sure no other device sends any frames during the data transfer. No RTS/CTS handshake is performed for the management frames. When a device initiates a scan, it should first wait for the end of ongoing data frame transfer to send a probe request. Therefore, a WiPush device can only receive a notification within the time interval between two data frames.

In Figure 8 and Figure 9, we show how the delivery ratio and latency for WiPush notifications change with increasing load on the WiFi channel. With the TCP runs, mean and standard deviation of the achieved data rates were 11.35 and 1.06 Mbps, respectively.

The figures demonstrate that the performance discrepancy between Nexus and RPi is even larger in high network load. This is because the WiFi chipset and driver can not sense the wireless medium with as high efficiency as Nexus. An increased portion of the probe requests sent by the RPi collide with the ongoing data

[9] https://iperf.fr

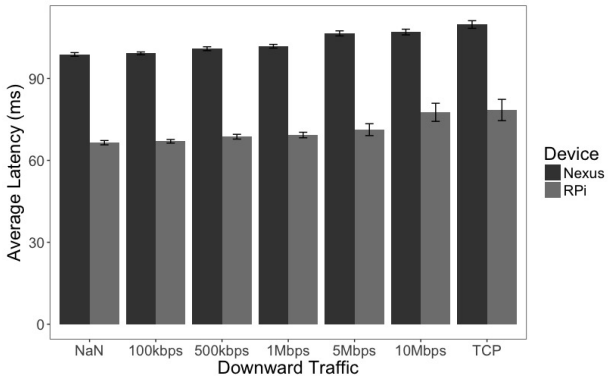

Figure 9: Latency with respect to the background traffic. With the higher load, the devices need to wait longer before transmitting probe requests when they scan WiFi networks.

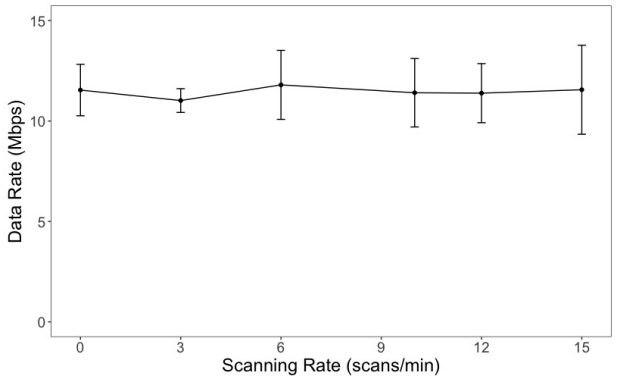

Figure 10: Network throughput with respect to WiPush activity. The data rate does not change significantly as the AP pushes more notifications to the devices that are capable of WiPush.

frames that corrupts the probe requests making it impossible for the AP to process them correctly. As a result, the delivery ratio drops with the increasing load on the WiFi channel. Nexus on the other hand does a very good job of sensing the wireless medium and delay the transmission of the probe requests when there is data traffic. Hence, it consistently receives a large portion of the notifications even with the increased load. Still, delivery ratio slightly decreases with increased load.

Figure 9 shows that the load increases the latency. Since the devices need to wait until the end of the transmission of the data frame before sending a probe request, it takes longer for both devices to receive the push notification.

7.2 Influence on the Network

In Section 7.1.3, we have analyzed how the ongoing network activity, i.e. the application traffic going through the AP influences the WiPush performance. In this section, we investigate whether the WiPush functionality and the frequency to push notifications to user devices diminishes AP's original purpose to provide connectivity. We control the frequency the AP pushes notifications through devices' scanning intervals. Every time a device scans the AP networks, the AP sends 2 WiPush notifications to the device. Here, we do not use frame injection as we did in Section 7.1.1 because injected frames do not follow CSMA/CA protocol and it is a fundamental aspect of IEEE 802.11 network.

To capture the effect of WiPush on the networking responsibilities of the AP, we use a test setup similar to the previous case with four parallel TCP streams initiated by iperf3 between RPi1 and RPi2 that run for five minutes. The traffic of these streams pass through the AP that also pushes WiPush notifications to Nexus and RPi. Even though there in an abundance of metrics that define the performance of the network, we limit our analysis with the total throughput between the two devices.

Figure 10 shows how the data rate from RPi1 to RPi2 varies with the scanning rate of WiPush devices that receive notifications from the AP. As before, the results are averaged over 5 runs and the vertical bars in the figure indicate 95% confidence intervals. We see no significant change in the data rate even though the AP sends

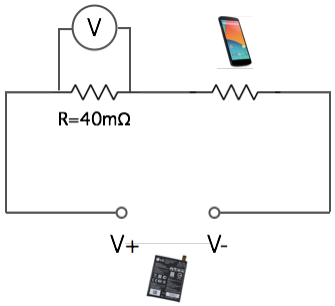

Figure 11: Setup for measuring the power in Nexus 5x with WiPush

more notifications. The data frames are handled in the kernel space so that they can be quickly forwarded to receiving station. On the other hand, WiPush notifications pushed by hostapd daemon in the AP that runs in the user space. Hence, the effect of computation related to WiPush in hostapd has little to no impact on data frames. In addition, WiPush is a lightweight mechanism that uses only one public action frame per a notification. This is insignificant with respect to the number of data frames that flow through the AP.

The delivery ratio and latency values obtained in these tests are consistent with those in Section 7.1.3.

7.3 Power Consumption

In this section, we evaluate the power consumption associated with WiPush. To do that, we disassembled the Nexus 5x smartphone and prepared the setup shown in Figure 11. We disconnected the battery from the rest of the device and placed a small resistor $R = 40\text{m}\Omega$ between them. The current that goes through the resister also passes through the device. The power the device uses for its operation directly depends on this current that it consumes. By Ohm's Law, the current I that goes through the resistor R can be derived from the voltage across R by $I = \frac{V}{R}$. We measured this voltage using a Tektronix oscilloscope.

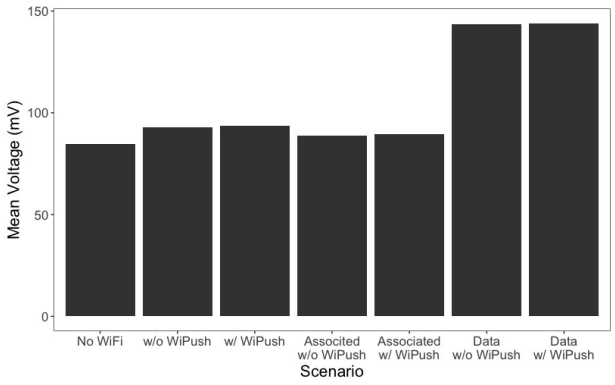

Figure 12: Mean voltage across the resistor R that indicates the current consumed by Nexus. WiPush only slightly increases the power consumption.

Figure 12 shows the mean voltage measurements across resistor R in a number of scenarios that is computed over the course of 2 minutes. As a baseline, we first measured the voltage when the WiFi interface turned off. Then, we evaluate the additional energy required by WiPush when (i) the device is not associated, (ii) it is associated with an AP that can send WiPush notifications but is not involved with online traffic and (iii) when it receives heavy traffic that is initiated with iperf3 tool with device initiates four parallel TCP sessions with a local server over WiFi, similar to the cases in Sections 7.1.3 and 7.2.

In all scenarios, we measured the voltage and hence the current flowing through the device with and without WiPush. In cases without WiPush, we compiled the wpa_supplicant daemon without the capability to receive notifications. In all these scenarios, the device is programmed to scan WiFi APs every 5 seconds. When WiPush is enabled, the AP sends 50 notifications to the device every time it receives a probe request. We use such a large number to make sure we capture the effect of receiving notifications.

In either case, we see that receiving WiPush notifications only marginally increases the consumed current, which is below 1%. This increase is due to both receiving and processing the action frames that carry the notifications. Note that the difference is typically even smaller since the intervals between the scans are much larger and devices would not receive such a high number of notifications at every scan.

We note that the power consumption is higher in unassociated case than the associated case in Figure 12. This observation is caused by artificial scans that are apart by a very short duration. After each scan, if the device does not recognize any APs, the WiFI interface goes to sleep. In other words, the device needs to awake its WiFi interface for each scan. When it is associated, the device sleeps its WiFi interface only after a duration of inactivity; however, the frequent scans prevent this in this test. Therefore, the WiFi interface is awake but it is on a low power state where it does not send or receive any data. Since, the energy that requires to awake and promote the interface is larger than the power in this low power state [10, 16], the power consumption turns out to be higher when this device is not associated. Note that; however, this is not representative of WiFi power consumption in general.

8 RELATED WORK

Beacon stuffing has been proposed in the past as a mechanism to convey information by a WiFi AP to unassociated devices [4]. The authors suggest three techniques to encode information in WiFi beacon frames: SSID concatenation, BSSID concatenation, and embedding data in vendor-specific Beacon Information Elements. Using either of these techniques, data fragments are broadcasted in the corresponding information elements of successive beacons to listening devices which can receive and combine the fragments to obtain data from an AP without being associated with it. Beacon stuffing differs from WiPush in terms of who instigates the communication (the AP vs. the device), the dependency on passive vs. active probing and support for directed messages vs. broadcast-only. Beacon stuffing abuses the IEEE 802.11 standard by using beacon frames and its information elements outside the scope of their purposes. In addition, the amount of data that can be transferred through Beacon Stuffing is extremely limited since SSID field can be at most 32 octets. Therefore, longer messages need to be fragmented. Given that an AP typically transmits a beacon frame every 100 ms, it takes more than 6 seconds to deliver a message of length 2 KByte through Beacon Stuffing with SSID concatenation. When the maximum information element size of 255 octets is used, the length of the data is still very limited considering the maximum payload of push notifications is 4KBytes. When there are multiple messages to be sent to users, the latency associated with Beacon Stuffing becomes even more dramatic. While beacon stuffing is implemented in device drivers, WiPush functionality is added via wpa_supplicant and hostapd daemons making it possible to work with any driver. Finally, since Beacon Stuffing requires devices to continually listen to beacon frames from APs, it prevents WiFi devices from sleeping to save energy. On the other hand, WiPush does not alter the duty cycle of the WiFi interface.

In [8, 9], the authors propose embedding information in the length fields of information elements. The length field takes one octet whereas the payload of most information elements is less than 255 octets. The unused bits in the length field are then used for encoding information.

CA-Fi [15] augments 802.11 with an association-less broadcast mechanism of up to 30 kB/s. It enables supported devices to communicate with each other using standardized frame types without compromising the internal workings of traditional WiFi or depending on specific WiFi drivers. Likewise, WiPush complements WiFi and does not require driver modifications. Whereas CA-Fi provides bi-directional communication between the device and the AP at an unavoidable extra energy cost, WiPush serves push notifications from an AP to a device at a very small energy cost by conveniently leveraging devices' active probing mechanism.

Several works have also targeted Bluetooth Low Energy (BLE) for contextual messaging, in particular using iBeacons. For instance, iBeacons were used to track items and send notifications about these items to user devices [11]. However, iBeacon-based use cases heavily depend on a dedicated application and often require an active network connection to lookup dynamic content associated with advertised beacon identifiers. Studies have also shown that users rarely have Bluetooth switched on by default, as compared to

WiFi [5]. With WiPush, we want to enable application developers to offer beacon-based use cases over existing WiFi infrastructure.

Turkes et al. use a combination of the advertisement mode of BLE and beacon stuffing to send contextual messages [13].

The IEEE 802.11 standard [1] specifies *General Advertisement Service* (GAS) protocol, which is also a communication protocol that operates through exchange of public action frames. The key difference between GAS and WiPush is that GAS is a query based protocol. In GAS, a STA advertises its GAS capability through a designated bit in its beacon or probe response. In order for other STAs to find out more about available services, they have to query for them explicitly with a *GAS Query Request* frame. In WiPush, probe requests are used to trigger communication when required.

9 CONCLUSION AND LIMITATIONS

Push notifications have become increasingly prevalent to allow third party content providers to send information to mobile devices. When such information is related to the user's spatio-temporal context, i.e. their location at a given time, current delivery mechanisms are costly in terms of energy and communication overhead as the device needs to compute its location and report it to remote servers.

In this paper, we present WiPush, a novel opportunistic push notification delivery mechanism for WiFi devices. WiPush delivers notifications even when the devices are not associated with an access point without abusing the underlying 802.11 standard.

It leverages the active scans devices perform by sending out broadcast probe requests and waiting for probe responses to find near by access points. This yields a short window to the access point that the user device is listening to the channel that it is operating on. Upon receiving a probe request from a WiPush enabled device, an access point encapsulates the notifications in a public action frame and push it to the user.

We implement the functionality in widely used hostapd and wpa_supplicant daemons. Even though WiPush is an opportunistic, best effort delivery mechanism, our evaluation shows that WiPush can achieve a large delivery ratio. On certain conditions such a heavy application traffic on the AP or increased number of notifications in each delivery opportunity, the actual performance heavily depends on the underlying driver and hardware. A commercial smart device such as Nexus 5X phone achieves an acceptable performance for notification delivery. However, such performance is still limited for general purpose data traffic transfer.

WiPush is not a communication mechanism for general purpose data traffic. The short window that the access point has after receiving the probe request before the device hops to another channel or turns off the WiFi interface prevents the transfer of large volumes of traffic. Instead, WiPush is only used for delivering short spatio-temporal notifications.

We take user preferences into consideration before sending out notifications so that WiPush is not abused as a spamming tool. The user preferences are managed in the access points and they decide on whether a notification should be sent to a user device. Other approaches, such as a local mechanism in the user device to filter out these notifications and not display them to the user, may be employed as well. We plan to study such approaches in depth in future work.

WiPush functionality relies on using IEEE 80211 management frames. These frames can be picked by a sniffer that is situated close by. On the other hand, since it leverages already emitted probe requests, it does not increase the device's risk to be geo-tracked. With users empowered to select the set of APs that can access their preferences, it is possible to let only trustworthy APs access personal information. On the other hand, a malicious party can sniff on notifications. However, the contents of the notifications can be encrypted by the AP so that only the non-intended devices or sniffers cannot infer the contents addressing security concerns.

ACKNOWLEDGMENTS

This work is part of SerGIo. SeRGIo is an icon project realized in collaboration with imec, with project support from VLAIO (Flanders Innovation & Entrepreneurship).

REFERENCES

[1] Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. *IEEE Std 802.11-2012 (Revision of IEEE Std 802.11-2007)*, pages 1–2793, March 2012.

[2] Cisco Visual Networking Index (VNI): The Zettabyte Era-Trends and Analysis. http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/VNI_Hyperconnectivity_WP.pdf, May 2015.

[3] U. G. Acer, A. Mashhadi, C. Forlivesi, and F. Kawsar. Energy Efficient Scheduling for Mobile Push Notifications. In *Mobile and Ubiquitous Systems: Computing, Networking, and Services - 12th International ICST Conference, (MobiQuitous)*, 2015.

[4] R. Chandra, J. Padhye, L. Ravindranath, and A. Wolman. Beacon-Stuffing: Wi-Fi without Associations. In *Proceedings of the 8th Workshop on Mobile Computing Systems and Applications (HotMobile)*, pages 53–57, 2007.

[5] Y. Chon, S. Kim, S. Lee, D. Kim, Y. Kim, and H. Cha. Sensing WiFi Packets in the Air: Practicality and Implications in Urban Mobility Monitoring. In *Proceedings of the International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp)*, pages 189–200, 2014.

[6] J. Freudiger. How Talkative is Your Mobile Device?: An Experimental Study of Wi-Fi Probe Requests. In *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, pages 8:1–8:6, 2015.

[7] W. Gu, Z. Yang, D. Xuan, W. Jia, and C. Que. Null Data Frame: A Double-Edged Sword in IEEE 802.11 WLANs. *IEEE Transactions on Parallel and Distributed Systems*, 21(7):897–910, 2010.

[8] V. Gupta and M. K. Rohil. Article: Information embedding in ieee 802.11 beacon frame. *IJCA Proceedings on National Conference on Communication Technologies & its impact on Next Generation Computing 2012*, (3):12–16, November 2012.

[9] V. Gupta and M. K. Rohil. Article: Bit-stuffing in 802.11 beacon frame: Embedding non-standard custom information. *International Journal of Computer Applications*, 63(2):6–12, February 2013.

[10] J. Huang, F. Qian, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck. A Close Examination of Performance and Power Characteristics of 4G LTE Networks. In *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*, MobiSys '12, pages 225–238, 2012.

[11] M. Koühne and J. Sieck. Location-based services with ibeacon technology. In *2014 2nd International Conference on Artificial Intelligence, Modelling and Simulation*, pages 315–321, 2014.

[12] N. D. Lane, E. Miluzzo, H. Lu, D. Peebles, T. Choudhury, and A. T. Campbell. A Survey of Mobile Phone Sensing. *Communications Magazine, IEEE*, 48(9):140–150, 2010.

[13] O. Turkes, H. Scholten, and P. J. Havinga. Blessed with opportunistic beacons: A lightweight data dissemination model for smart mobile ad-hoc networks. In *Proceedings of the 10th ACM MobiCom Workshop on Challenged Networks*, CHANTS '15, pages 25–30, New York, NY, USA, 2015. ACM.

[14] O. Waltari and J. Kangasharju. The Wireless Shark: Identifying WiFi Devices Based on Probe Fingerprints. In *Proceedings of the First Workshop on Mobile Data*, MobiData '16, pages 1–6, 2016.

[15] H. Wirtz, T. Zimmermann, M. Ceriotti, and K. Wehrle. CA-Fi: Ubiquitous Mobile Wireless Networking without 802.11 Overhead and Restrictions. In *Proceedings of the 15th International Symposium on a World of Wireless Mobile and Multimedia Networks (WoWMoM)*, 2014.

[16] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang. Accurate Online Power Estimation and Automatic Battery Behavior Based Power Model Generation for Smartphones. In *Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, CODES/ISSS '10, pages 105–114, 2010.