Chapter 2
Smittestopp for Android and iOS

Per Magne Florvaag, Henrik Aasen Kjeldsberg, and Sebastian Kenji Mitusch

Abstract Contact tracing is currently a manual and laborious task that requires individuals to recall their interactions with people many days in the past. As a remedy, phones can be used to play a significant role in the response to the COVID-19 pandemic, by easing the burden of healthcare staff. Through novel and sophisticated technology, apps can be used to track infected people, issue quarantine guidelines, and provide the latest news to the public. Along with general public measures, apps can contribute significantly to keeping infection levels low. Generally, digital contract tracing can identify and warn people who may be at risk of being infected because they were in close physical proximity of someone who later tested positive for COVID-19.

2.1 Introduction

Our contact tracing app Smittestopp was released on 16 April 2020 on the Google Play store and Apple's App Store, and later for the Huawei AppGallery. The app supported Android 5.0+ and iOS 12.0+ and required users to register with a Norwegian phone number.

P.M. Florvaag
Department of Computational Physiology, Simula Research Laboratory,
Simula Consulting AS and Pacertool AS
e-mail: permagne@simula.no

H.A. Kjeldsberg
Department of Computational Physiology, Simula Research Laboratory,
e-mail: henriakj@simula.no

S.K. Mitusch
Department of Numerical Analysis and Scientific Computing, Simula Research Laboratory,
e-mail: sebastian@simula.no

A. Elmokashfi et al. (eds.), *Smittestopp – A Case Study on Digital Contact Tracing*,
Simula SpringerBriefs on Computing 11, https://doi.org/10.1007/978-3-031-05466-2_2

To facilitate digital contact tracing, the apps had to collect information that could reveal the proximity between two devices running the app. If the devices are considered sufficiently close to each other, beyond a given threshold, and one (or both) of the users are later confirmed to be infected by COVID-19, the counterpart would be notified that they could be infected as well. The location data provided by Global Positioning System (GPS) can be used to match the locations of two users, but it is too coarse to distinguish distances to a precision of 2 metres, especially in cities and inside buildings. To counter this problem, Bluetooth data were supplied and combined with location data. Bluetooth signals fade rapidly over short distances, and one can semi-reliably determine distances with a precision of 2 metres.

GPS data are, however, very useful for the second purpose of the app: to gather anonymized movement patterns for epidemiological research. Specifically, the location data were intended to be used to evaluate the effect of social distancing measures imposed by the government.

For effective contact tracing, the app would need to run continuously in the background, even through phone reboots and app terminations. However, most users of such an app would open it once and then rarely or never open it again. As we will elaborate in this chapter, this proved to be a challenging aspect, especially for the iOS version of the app. More surprisingly, background location permissions would be essential for the life cycle of the app on iOS.

2.2 Related apps for digital contact tracing

Before the development of Smittestopp began, no digital contact tracing app had ever been released in a Western country. Although China had deployed apps to combat COVID-19, these apps functioned by showing the user's health status based on an algorithm that accounted for the user's travel history. Checkpoints were placed around China where a certain health status was required to pass through, such as when entering a metro station [28].

On 20 March 2020, Singapore released their digital contact tracing app TraceTogether [24]. TraceTogether gathers Bluetooth proximity data and stores them locally on the phone. The app fetches temporary identifiers from a central server and transmits these over Bluetooth. At the same time, the app builds a log of temporary identifiers it obtains from other devices in proximity. This contact log is then kept locally on the phone. When users tests positive for COVID-19, they can voluntarily share this contact log with the health authorities, who will then notify the other users. As other Bluetooth-based contact tracing apps on iOS, TraceTogether did not function properly in the background. Thus, iOS users were asked to use the app as a screen saver when not using the phone, ensuring that the app remained open and Bluetooth would work correctly [1].

On 22 March, Israel released their app HaMagen [23], which gathers location data through GPS (later versions also incorporate Bluetooth data). These data are stored locally on the phone until the user is diagnosed with COVID-19. Infected individuals

can choose to upload these data so that other phones can download and compare these with their locally stored location history. The user is notified by the app if there is a match, but, to preserve privacy, the health authorities are not automatically notified.

The United Kingdom was initially developing a centralized contact tracing app using Bluetooth, but abandoned it in May, after reports that the background problems on iOS led to only a 4% detection rate between two iPhones that were asleep [18]. However, the importance of the 4% detection rate could have been overstated, since the quantity of contacts where both iPhones are asleep account for only a small percentage of overall contacts [19]. Instead, the UK government decided to switch to the Google/Apple Exposure Notifications (GAEN) application programming interface (API).

The GAEN API [26] is an API implemented by Google and Apple for both the Android and iOS operating systems. GAEN is similar to the *Decentralized Privacy-Preserving Proximity Tracing* (DP-3T) protocol [27], using only Bluetooth for proximity detection and storing all contact logs locally. However, a key difference between the two implementations is that the GAEN key matching occurs at the OS level, whereas that of the DP-3T protocol occurs at the app level. A GAEN app generates an identifier roughly every 15 minutes and transmits it over Bluetooth. When users test positive for COVID-19, they can choose to upload a log of the temporary keys they generated so that these can be downloaded by other users and matched against their locally stored contact logs. GAEN is currently the de facto standard framework for exposure notification.

2.3 App user interface and functionality

Smittestopp's user interface (UI) is mainly divided into four processes, or views. When the user starts Smittestopp for the first time, they are directed through the *onboarding* process. The onboarding process for both the Android and iOS apps is shown sequentially through the screenshots in Figure 2.1. The top row shows the process for Android, while the bottom one shows it for iOS. The onboarding is key to understanding Smittestopp, explaining the main purposes of the app, as shown in the first two panels (columns) of Figure 2.1. Furthermore, the onboarding presents the privacy policy, which the user is required to accept to continue using the app, shown in the third panel of Figure 2.1. Similarly, the user is required to verify that they are above the age of 16, a requirement for using Smittestopp, as shown in the fourth panel of Figure 2.1. In the iOS version, as shown in panel five of Figure 2.1, the onboarding page allows the user to authorize Smittestopp to collect Bluetooth and location service data. The user is given an authorization prompt in both versions of the app after login. Finally, the user is directed to the login services, provided by the Microsoft Authentication Library [22]. The login service requires users to input a Norwegian phone number and to authenticate themselves by replying with a confirmation code sent by SMS.

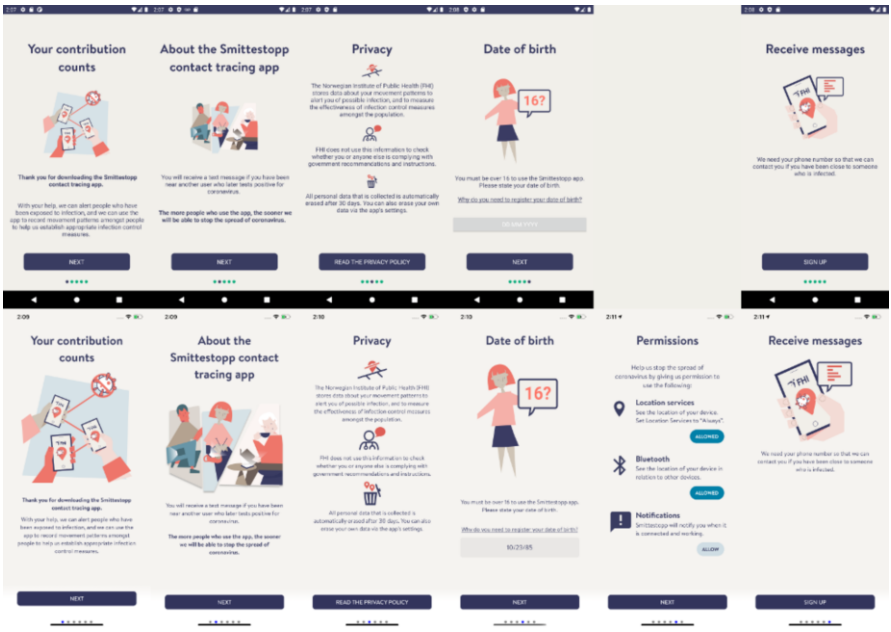
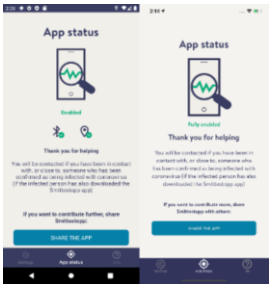
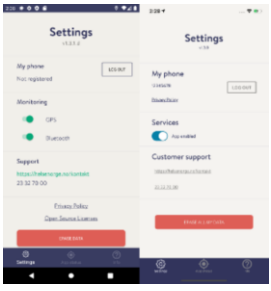


Fig. 2.1: Screenshots from Smittestopp's onboarding process. The top and bottom panels represent the onboarding flow in the Android and iOS versions of Smittestopp, respectively. Each flow involves up to six steps. Note that the Android version does not include the Permissions page.



(a) The monitoring view.



(b) The settings view.

(c) The info view.

Fig. 2.2: The three view components after a successful login: monitoring, settings, and info. For all three images, the left and right screenshots represent the Android and iOS versions of Smittestopp, respectively.

After successfully logging in through Microsoft's service, the user is presented with the *monitoring* page, as shown in Figure 2.2a. The monitoring page serves as an overview of the app's status, which is either **enabled**, **partly enabled**, or **disabled**, although the wording can vary between apps. In Figure 2.2a, the app is

fully enabled, implying the collection of both Bluetooth and location service data has been authorized and is activated.

In contrast, when partly enabled, that is, when either Bluetooth or location services are disabled, a button prompting the user to toggle the respective setting is shown. However, the collection of either data type will still contribute to the app's purposes, although the precision could be affected. Finally, if the monitoring shows the disabled status, then both Bluetooth and location services have been deactivated, either through the app's settings or through the phone's settings.

The supplementary view components consist of the settings and info views, shown in Figures 2.2b and 2.2c, respectively. The settings page displays the phone number with which the app was registered and the ability to log out, which temporarily halts the data collection. The user can also toggle the information that is collected. Customer support information is presented in the next settings panel, including a link to `Helsenorge.no` and its support phone number. In this panel, the user can also erase all the data collected by Smittestopp thus far. Finally, the *Info* view, shown in Figure 2.2c, displays helpful links related to Smittestopp, data collection, and general information about COVID-19.

2.4 System architecture and data flow

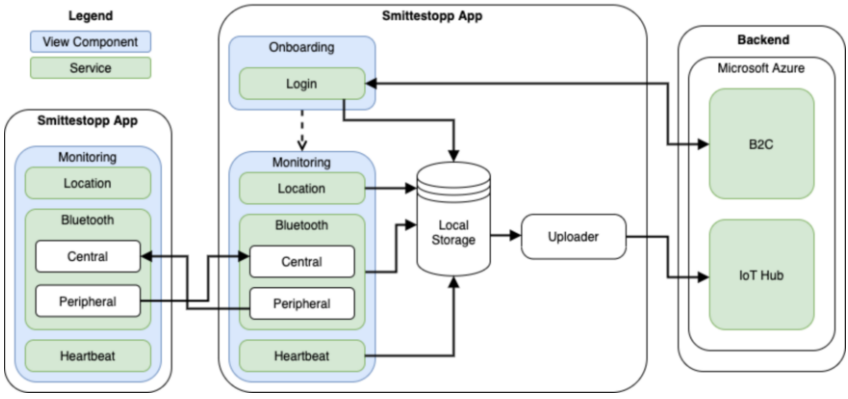


Fig. 2.3: High-level schematic of Smittestopp showing the main components and how they interact with the backend, through Microsoft services here, and with other devices where Smittestopp is installed. The backend consists of Azure Active Directory Business-to-Consumer (Azure AD B2C) and the Azure Internet of Things (IoT) Hub, which connects to the cloud.

A software architecture establishes the fundamental structure of a software system and displays how a collection of components accomplishes a specific task or function.

The main functions of Smittestopp are to aggregate location and Bluetooth data, with minor functions, such as the logging of events and heartbeat monitoring. The main components are presented in Figure 2.3, which is a high-level schematic overview of the process flow in Smittestopp. Although there are minor technical differences between the Android and iOS apps, the main components and data processing in Smittestopp are covered by the schematic.

Starting with their onboarding, users proceed through a login service provided by Azure AD B2C. The users input their Norwegian phone number and receive an access token from Azure AD B2C that is used to authenticate the user for Azure IoT Hub. The IoT hub responds with an authentication key and a Universally Unique Identifier (UUID), which are stored locally on the user's device. The authentication key is used to generate temporary authentication tokens for sending messages to the IoT hub. These messages are sent over HTTPS, with a payload as described below, in addition to the device's UUID.

After successfully logging in, the user is presented with the monitoring view component. Here, three main data types are actively collected and uploaded to the IoT hub, assuming full authorization to location services (GPS) and Bluetooth Low Evergy (BLE). All three events include five common fields, along with the event data, as shown in the following JavaScript Object Notation (JSON) message format example.

```
{
  "appVersion": "1.1.0",
  "model": "iPhone10,5",
  "events": [EventData],
  "platform": "ios",
  "osVersion": "13.4.1",
  "jailbroken": false
}
```

The `jailbroken` flag was added to filter out data collection from rooted and jailbroken devices. We attempt to identify jailbroken or rooted devices by checking if the app can edit certain system files or if the system contains files associated with jailbroken/rooted devices. Furthermore, events is a list of either GPS, Bluetooth, or heartbeat events, but never a combination thereof.

GPS data are collected on a regular basis, although the uploading frequency and data precision can vary, depending on the user's activity. For GPS events, the following example message format shows the structure of the IoT hub telemetry message sent from the app to the cloud, including the common fields addressed above.

```
{
  "timeFrom": "2020-04-30T12:38:30Z",
  "timeTo": "2020-04-30T12:38:30Z",
  "latitude": 61.93372532454498,
  "longitude": 10.728583389659596,
```

```
  "accuracy": 65.0,
  "speed": 2.10,
  "altitude": 71.1960678100586,
  "altitudeAccuracy": 10.0
}
```

Newly collected GPS data are aggregated with previous GPS events for the period lasting from `timeFrom` to `timeTo`. The current location is determined by the `latitude` and `longitude` coordinates, along with the `altitude` above sea level, measured in metres. The `accuracy` and the `altitudeAccuracy` are measures of the location and altitude accuracy, respectively. In addition, the measured `speed` of the device, in metres per second, is registered [14, 8].

Devices that support BLE, particularly Android and iOS devices, can act as a peripheral device and a central device, as explained further in Section 2.6.3. As illustrated in Figure 2.3, a device acting as a central device can be detected by peripheral devices, which triggers an exchange of UUIDs between the devices involved. In the initial release of Smittestopp, devices used static UUIDs. However, the use of static UUIDs can expose user devices to tracking by scanners configured to detect Smittestopp UUIDs. As a remedy, rotating UUIDs can be used, an improvement that was later implemented and tested but never released to the public, because Smittestopp was abruptly halted. To link the BLE information to GPS data, the BLE data are aggregated with the last known GPS positions, saved in the `location` field, as shown in the following message format representing a BLE event.

```
{
  "deviceId": "123456789abcd",
  "rssi": -90,
  "txPower": 12,
  "time": "2020-04-30T12:38:30Z",
  "location": {
    "latitude": 61.93372532454498,
    "longitude": 10.728583389659596,
    "accuracy": 65.0,
    "timestamp": "2020-04-30T12:35:30Z"
  }
}
```

The contact timestamp is registered in the `time` field, along with the UUID, `deviceId`, of the discovered device. Here, `rssi` represents the signal strength that the central device receives, while `txPower` represents the transmission power of the peripheral device.

The third and final event consists of heartbeat messages, a custom message used to determine a device's authorization of location services. The heartbeat message can also determine that the app is still installed and running on a device. A heartbeat event is sent once every 24 hours to the IoT hub, and it includes information on which

kind of data collection is enabled on the device, as shown in the following message format.

```
{
  "timestamp": "2020-04-30T12:38:30Z",
  "state": 0
}
```

Here, the `timestamp` is updated when the message is uploaded, and the integer value of `state` is set to one of four values.

Smittestopp is also connected to Azure App Center for logging purposes [20]. Note that the App Center is used by the app in parallel to the main app services, thus running independently of the Smittestopp backend. Mainly errors and warnings are logged and uploaded to the App Center, including information about failed authorization, database errors, and failed requests and responses related to event uploading. In addition, the operating system, mobile operator, version number, and phone model is added to the App Center payload, used to improve the quality of the collected data by distinguishing different phone models and operating systems. Azure App Center does not store any personal information or link the collected data to a user [21]. It is important to emphasize that the information collected by App Center was only used to identify problems with certain phone models or operating systems.

2.5 App life cycle

Whether running on the iOS or Android operating system, every mobile app passes through multiple states throughout its runtime, known as the app's life cycle. Of the different states an app can transition through, we mainly focus on those when the app is running in the background, since most users of Smittestopp rarely opened the app. Users bring apps to the foreground to interact with them. Consequently, such apps will be prioritized when it comes to accessing systems resources. In contrast, apps in the background are not visible to users. An app goes into the background if has been stopped or has entered a suspended state. Most apps are usually in a background or suspended state, to save as much power as possible. Optimally, the app does as little work as possible, preferably nothing when off-screen. There are also intermediate states, when the app's state changes from the foreground to the background and vice versa, but these are not our focus here.

2.5.1 Android

Smittestopp supports Android 5.0+ and controls its life cycle through different *activities*, as shown in Figure 2.4. The different activities describe the actions that

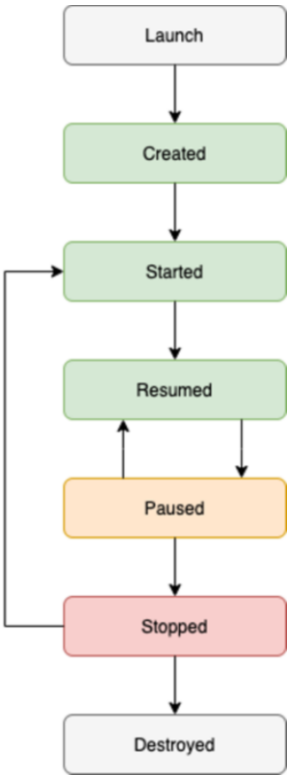


Fig. 2.4: A state diagram that describes the transitions between the states for the Android life cycle.

users can perform to make the app enter different life cycle states. An Android app cycles between the four following life cycle states: active, paused, stopped, and terminated [16]. The states are entered through different activities, which we describe next.

The app becomes active by going through the three activities *create*, *start*, and *resume*. When the user opens an app, the create activity is triggered. The app continues by executing the start activity. In this phase, the activity is still not rendered, but is about to become visible to the user. The final phase of being active involves the app entering the resume activity, where the app is finally visible to the user and becomes interactive.

At this point the app can be paused, stopped, or terminated. In the paused state, the app can still be visible to the user, but the user cannot interact with it. This state can be entered when the app is no longer in focus or before transitioning to the stopped or terminated state. The app enters the stopped state when it is not visible to the user, which can happen if a new activity is started or the current one is being terminated. Although the app is still active in the background, Android Runtime can

terminate the app in case of scarce resources. Finally, if the app is terminated, it will destroy the current instance of the activity to save memory.

Considering that most apps are usually not active, we used a designated background service in Android that allows the app to execute events in the background, as well as showing a constant notification.

2.5.2 iOS

Smittestopp supports iOS 12.0+ and uses app delegate objects to manage the app's shared behaviours [9]. Generally, an iOS app can enter one of five states that constitute the app life cycle: terminated, inactive, active, background, and suspended, as shown in Figure 2.5. For the sake of completeness, we describe here the five states before focusing, in the next section, on the main challenges faced by iOS apps when they run in the background. An app in the *terminated* state has either not been started yet or has been closed by the user or the system. As soon as the user enters the app, the enters an intermediate state where it is *inactive*. In the inactive state, the app's UI is not visible to the user and does not receive or send any events. The inactive state is also entered every time the app transitions to a different state.

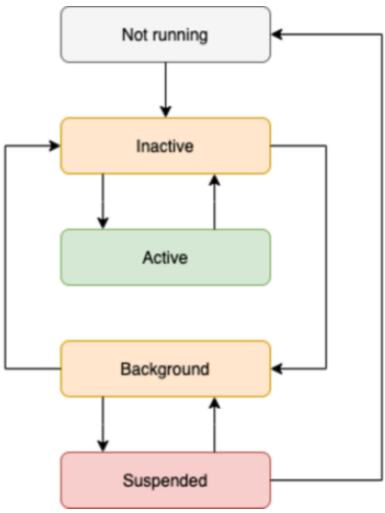


Fig. 2.5: A state diagram describing the transitions between the states for the iOS life cycle.

When the app is fully loaded, the app enters the *active* state. In the active state, the app is fully functional and visible to the user and the user can interact with the UI. Additionally, the app can both send and receive events if this is part of the its functionality. When the user exits the app, it transitions from the active state to the inactive state before reaching the *background* state. Similarly, when reopening, the app will transition from the background state to the inactive state before eventually becoming active. The background state is usually only a temporary state in which the app's code is still executed, meaning that events can be sent and the app works in the background, although the UI is not visible to the user. After being in the background state for a short time, the app will enter the *suspended* state. The time it takes before the app transitions from the background state to the suspended state can vary, since this state can be extended, if needed, by the app. Most apps are automatically suspended by the system after entering the background state. In this state, the app does not execute code, but it is still saved in

the iPhone's memory without affecting the battery life. In case the system runs low on memory, a suspended app can automatically be terminated by the system. An app can also enter the terminated state if it is manually terminated by the user.

2.6 Design choices

In this section, we discuss the various design choices that we made when developing the Smittestopp app. Section 2.6.1 gives a short description of how data are stored locally on the device and the security measures implemented for this storage. Battery usage is a prominent issue for a continuously running app such as Smittestopp, especially considering that it provides no immediate and obvious benefit to the user. Most of the battery drain was caused by location tracking, and details on how this was handled are provided in Section 2.6.2.Finally, Section 2.6.3 describes how proximity detection over Bluetooth was implemented.

2.6.1 Storage and security

Local storage consists of two different systems. One system is for preference data, such as the phone number, consent to the privacy policy, and the login token. The other system is a local encrypted SQLite database for measurement data, such as location and Bluetooth encounters. Preference data are stored in UserDefaults and the Keychain [7] on iOS, and in SharedPreferences and Keystore [13] on Android. Keychain and Keystore are used to store sensitive information such as the phone number, the login token, and the encryption key for the local SQLite database.

The local database system choice was SQLite [25] because it is an embedded database with easily accessible libraries. On Android, SQLite is part of the AndroidX library [15]. On iOS, an open source library was used [12]. The database consists of two tables, one for GPS data and one for BLE data. As the data are being uploaded, they are marked for deletion and, upon successful upload, the marked data are deleted. Since the location and proximity data can persist on a phone for hours before they are successfully uploaded, it is important that they are stored as securely as possible. This entails the entire database being encrypted using a key that is generated and stored on the device in Keychain (iOS) or Keystore (Android). Although someone with full access to the phone could theoretically access the encryption key, this database encryption provides a basic level of security.

2.6.2 Location services

The collection of location data in Smittestopp is performed for two purposes: for digital contact tracing in combination with BLE event data and for gathering movement patterns in the population for epidemiological research, to understand the effectiveness of recommended public measures. By default, both the Android and iOS apps fetch location data at regular intervals, merging similar location data points to avoid sending too much data. However, the continuous tracking of location services was one of the main power usages of Smittestopp, as reflected in the number of reviews on both Apple's App Store and the Google Play store complaining about the app's impact on battery life.To address the high power consumption, Smittestopp would change the accuracy of the GPS data adaptively.

While, on Android, only the intervals at which location data were retrieved was tweaked, the iOS version of Smittestopp combined multiple different features of the CoreLocation [2] framework. By default, the app uses the standard `CoreLocation` location updates with the highest possible precision. If the GPS data show the same position, or roughly the same within a threshold dependent on the GPS accuracy, for more than five minutes, high precision location updates are disabled and, instead, *region monitoring* [10] is used. Region monitoring, also known as geofencing, constructs a circular region around a position with a given radius and tracks if the user moves outside the region. The region is defined as a circle with a 40-metre radius around the last known position. iOS then alerts the app once the device moves outside the region and stays outside for at least 20 seconds.

When an iOS user grants Smittestopp permission to fetch location data in the background, the iOS app can run continuously in the background state, never moving to suspended state. The standard location updates [4] in the `CoreLocation` framework prevent the app from being suspended when moving to the background. However, turning off standard location updates would mean the app will be suspended. Thus, when switching to region monitoring, standard location updates are not turned off, but the accuracy is significantly lowered and the filter for how far apart each update needs to be is significantly increased. This means the app remains enabled but there will be no standard location updates, which prevents the app from moving to the suspended state.

Apart from the suspended state, the terminated state could also be a problem for Smittestopp on iOS. The app could enter the terminated state if the user manually terminated the app, which, one can imagine, is a very normal occurrence; however, when the user moves outside the currently monitored region, the app is launched automatically by the operating system [5]. In addition, a `CoreLocation` feature called *significant location updates* [4], which provides updates if the device moves by roughly 500 metres or more, is always enabled. Region monitoring and significant location updates in conjunction meant that, after app termination, the app would relaunch and continue as normal if the device moved 40 to 500 metres from its last known position.

Although the app could run continuously in the background, on iOS, this was entirely dependent on the user granting full background location permissions. Without

those permissions, the app would not be able to gather location data in the background and would almost entirely rely on Bluetooth to wake the app. Because the Bluetooth background mode does not work after app termination and most users will not regularly launch the app, the iOS version of Smittestopp was extremely reliant on full location permissions for long-term consistency. This point was made tougher by iOS 13, which does not allow for background location permissions to be asked directly. Instead, one can only ask for permissions when the app is in use, and the user would later, at the discretion of the operating system, be prompted for background location permissions. This made it difficult to communicate to users what permissions they should grant, and the released version of Smittestopp did not attempt to convey the importance of background location permissions.

2.6.3 Bluetooth Low Energy

Bluetooth Low Energy (BLE) communication consists of two devices: one device advertising its presence and the other scanning for advertising BLE devices. BLE advertisement packets usually contain one or more UUIDs that inform the scanning devices of the types of services supported by the device in question. Such a UUID is referred to as a service UUID. The Smittestopp app advertises and scans for a service UUID specific to the app, providing a way for the scanning device to detect devices in its proximity. While BLE supports attaching some (limited size) data to the advertisement packet, there is limited support for accessing these data on iOS devices. Specifically, the data are not accessible when the scanning device has Smittestopp running in the background, and not in the foreground. Thus, Smittestopp instead connects to the advertising device when the app-specific service UUID is present. These connections are short-lived, since the scanning device only requests a device identifier from the advertising device and disconnects as soon as this has been received.

Running BLE in the background on an iOS app is supported through background modes, but the app will be suspended after a few seconds in the background. It will then transition from the suspended to the background mode every time a relevant BLE service UUID is found in a received advertisement packet or when a device connects. The app will then have approximately 30 seconds in the background state before it is suspended again.

Additionally, when the iOS app is in the background, the advertisement packet for the app changes to a proprietary format in which service UUIDs are found in the so-called overflow area [11]. The overflow area is a 128-bit array, and each service UUID corresponds to exactly one of the bits being set to one. When an iOS device scans for a specific service UUID, it will match advertisements where the corresponding bit is set to one. Of course, service UUIDs can be 128-bit numbers, for a many-to-one mapping from a service UUID. Thus, there is a possibility that a false positive can be detected if a device advertises a service UUID that has the

same corresponding overflow area bit. For more information on the overflow area, see the notes by Rossum [29] and Young [30].

While the BLE stack in iOS has a built-in system for handling the overflow advertisement format, an Android device does not know how to translate service UUIDs to their corresponding bit. To do so, the Android implementation of Smittestopp, in addition to scanning for the normal app-specific service UUID, also scans for packets with the corresponding bit set to one. Since the code for mapping service UUIDs to a bit is not public, the bit is found by scanning the BLE advertisement packets that the iOS app transmits in the background. This is possible because the corresponding bit is always the same for a specific service UUID.

The difficulties with BLE on iOS while the app is in the background are not only limited to the advertisement packets. When the scanning app is in the background, iOS will not relay any overflow advertisement packets detected to the app. Thus, while in the background state, an iOS app will not be able to detect other backgrounded iOS apps. While the app will function fine between iOS and Android devices, detection between two backgrounded iOS apps will not work. Furthermore, more than half of the phone users in Norway use an iPhone. Because the average user will have the app almost exclusively in the background state, this was a major problem and one of the main topics in meetings with collaborating countries.

In early April, two weeks before the app would eventually launch, we discovered a way to partially circumvent this limitation. Using the iBeacon feature [6], found in the `CoreLocation` [2] framework, the operating system will continue to relay overflow advertisement packets to the app, even in the background. Specifically, by ranging for iBeacons [3], a method that allows one to determine the proximity of other devices with iBeacon, the app can continue to receive overflow advertisements while the device screen is on. The iBeacons for which the app scans do not have to be present at all. The app can scan for a random iBeacon UUID. However, if the device screen is off, this method will not help the app detect BLE advertisements. Notably, whether the device is locked or not does not matter, as long as the screen is on. This means that if, for example, the phone is locked but receives a notification, it will light up and the app will receive BLE advertisements until the screen goes black again. With this workaround, it is possible to detect all encounters where at least one phone is in use. This method is also briefly mentioned by Young [30].

Although ranging for an iBeacon would help with BLE detection, a major concern was the impact on battery life. Hence, we aimed to minimize the time the app spends ranging for an iBeacon. The documented iOS app API from Apple does not include a way to detect whether the phone screen is on or off. Therefore, the app potentially ranges for an iBeacon when it has no effect, negatively impacting the battery life for no benefit. However, using an undocumented API, we can register callbacks that are invoked when the screen is turned off or on. The use of such undocumented APIs usually means that the app will be rejected in the App Store review process, and Apple made no exception for Smittestopp regarding this matter. Thus, the Smittestopp app ended up ranging for an iBeacon every five minutes for 10 seconds, regardless of whether the screen is on or off.

Ranging for iBeacons requires location permissions on iOS, even when only used to improve BLE background consistency. Additionally, to turn iBeacons on and off while in the background, the app must not be in a suspended state and must have background location permissions. From the user's perspective, requiring location permissions to improve BLE capabilities is not intuitive. Communicating the need for location permissions is therefore a significant challenge for an app using this workaround.

2.7 Testing

Smittestopp's codebase was tested using unit tests for the functionality, UI, and snapshot tests [17] to test the UI. The UI tests simulate interactions with the UI and check for the expected behaviour. Meanwhile, Snapshot tests compare old screenshots of the UI to the current UI, to ensure that no unintended changes occur.

2.8 Conclusions and lessons learned

Designing an app that runs almost exclusively in the background is much more straightforward on Android than on iOS. To conserve battery life, most iOS apps are not allowed to execute code when they are backgrounded, and, even if an app asks for time in the background, the iOS system will only rarely or even never grant background execution time if the app is rarely used.

Furthermore, the iOS system limits the functionality of BLE in the background. This limitation can be partially alleviated by background location permissions, but the user must explicitly grant these. This is perhaps a good thing from a privacy perspective, since it makes it difficult for apps to implement tracking mechanisms over BLE. However, it was one of the main challenges for countries implementing BLE contact tracing apps.

References

[1] G. T. Agency. 6 things about OpenTrace, the open-source code published by the TraceTogether team. https://www.tech.gov.sg/media/technews/six-things-about-opentrace\#6-last-but-not-least-an-extra-step-for-ios-users, 2020 (accessed October 26, 2020).

[2] Apple Developer Documentation. Core Location. https://developer.apple.com/documentation/corelocation/, 2020 (accessed October 26, 2020).

[3] Apple Developer Documentation. Determining the Proximity to an iBeacon Device. https://developer.apple.com/documentation/corelocation/determining_the_proximity_to_an_ibeacon_device, 2020 (accessed October 26, 2020).
[4] Apple Developer Documentation. Getting the User's Location. https://developer.apple.com/documentation/corelocation/getting_the_user_s_location, 2020 (accessed October 26, 2020).
[5] Apple Developer Documentation. Handling Location Events in the Background. https://developer.apple.com/documentation/corelocation/getting_the_user_s_location/handling_location_events_in_the_background, 2020 (accessed October 26, 2020).
[6] Apple Developer Documentation. iBeacon. https://developer.apple.com/ibeacon/, 2020 (accessed October 26, 2020).
[7] Apple Developer Documentation. Keychain Services. https://developer.apple.com/documentation/security/keychain_services, 2020 (accessed October 26, 2020).
[8] Apple Developer Documentation. Location Services. https://developer.apple.com/documentation/corelocation/cllocationmanager, 2020 (accessed October 26, 2020).
[9] Apple Developer Documentation. Managing Your App's Life Cycle. https://developer.apple.com/documentation/uikit/app_and_environment/managing_your_app_s_life_cycle, 2020 (accessed October 26, 2020).
[10] Apple Developer Documentation. Monitoring the User's Proximity to Geographic Regions. https://developer.apple.com/documentation/corelocation/monitoring_the_user_s_proximity_to_geographic_regions, 2020 (accessed October 26, 2020).
[11] Apple Developer Documentation. startAdvertising(_:). https://developer.apple.com/documentation/corebluetooth/cbperipheralmanager/1393252-startadvertising, 2020 (accessed October 26, 2020).
[12] S. Celis. A type-safe, Swift-language layer over SQLite3. https://github.com/stephencelis/SQLite.swift, 2020 (accessed October 26, 2020).
[13] A. Developers. Android keystore system. https://developer.android.com/training/articles/keystore, 2020 (accessed October 26, 2020).
[14] A. Developers. Android location services. https://developer.android.com/reference/android/location/Location, 2020 (accessed October 26, 2020).
[15] A. Developers. Sqlite. https://developer.android.com/jetpack/androidx/releases/sqlite, 2020 (accessed October 26, 2020).
[16] A. Developers. Understand the Activity Lifecycle. https://developer.android.com/guide/components/activities/activity-lifecycle, 2020 (accessed October 26, 2020).

[17] Github. Delightful Swift snapshot testing. https://github.com/pointfreeco/swift-snapshot-testing, 2020 (accessed October 23, 2020).
[18] L. Kelion. Coronavirus: England's contact-tracing app gets green light for trial. https://www.bbc.com/news/technology-53753678, 2020 (accessed October 23, 2020).
[19] L. Kelion. Coronavirus: Ireland set to launch contact-trace app. https://www.bbc.com/news/technology-53137816, 2021 (accessed February 18, 2021).
[20] Microsoft Azure. Visual Studio App Center. https://azure.microsoft.com/en-us/services/app-center, 2020 (accessed October 26, 2020).
[21] Microsoft Azure. Visual Studio App Center. https://docs.microsoft.com/en-us/appcenter/gdpr/faq\#data-use, 2021 (accessed February 25, 2021).
[22] Microsoft Azure. Microsoft Authentication Library (MSAL). https://docs.microsoft.com/en-us/azure/active-directory/develop/msal-overview, 2021 (accessed February 26, 2021).
[23] I. M. of Health. HaMagen. https://govextra.gov.il/ministry-of-health/hamagen-app/download-en/, 2020 (accessed October 23, 2020).
[24] G. of Singapore. TraceTogether. https://www.tracetogether.gov.sg/, 2020 (accessed October 26, 2020).
[25] SQLite. SQLite Home Page. https://www.sqlite.org/, 2020 (accessed October 26, 2020).
[26] P.-P. C. Tracing. Apple and Google. https://covid19.apple.com/contacttracing, 2020 (accessed October 26, 2020).
[27] C. Troncoso, M. Payer, J.-P. Hubaux, M. Salathé, J. Larus, E. Bugnion, W. Lueks, T. Stadler, A. Pyrgelis, D. Antonioli, et al. Decentralized privacy-preserving proximity tracing. *arXiv preprint arXiv:2005.12273*, 2020.
[28] J. Utzerath, R. Bird, and G. Cheng. Contact tracing apps in China, Hong Kong, Singapore and South Korea. https://www.lexology.com/library/detail.aspx?g=99dca469-455d-4f7a-b025-00bf1d10ff6b, 2020 (accessed October 23, 2020).
[29] A. van Rossum. Smartphone localization. https://github.com/crownstone/bluenet-ios-basic-localization/blob/master/BROADCASTING_AS_BEACON.md, 2020 (accessed October 26, 2020).
[30] D. G. Young. Hacking The Overflow Area. http://www.davidgyoungtech.com/2020/05/07/hacking-the-overflow-area, 2020 (accessed October 26, 2020).