Contact tracing beacon testing

2020-07-14

Introduction

The aim of contact tracing is to detect and track close contacts between people to determine whether two people have been in (a) sufficiently close proximity, and (b) for sufficiently long duration, to facilitate disease transmission. The quality of a solution can therefore be measured by its ability to:

1. Identify all people in the vicinity
2. Estimate distance to each person
3. Track distance to each person over time

A perfect solution will accurately track distance to every person in the vicinity, indefinitely, without user action. The result is a log that enables accurate identification of contacts that facilitate disease transmission, thus making it feasible to accurately disrupt the spread of the disease.

In practice, even the most complete solutions are far from perfect. Bluetooth beacon based solutions are unable to identify all other devices due to bandwidth limitations. Distance estimates are unreliable due to signal interference, and tracking over time is challenging due to technical limitations and security features in the operating system and Bluetooth specification. All radio signal based solutions (like Bluetooth and WiFi) shall suffer from similar problems in varying degrees. GPS based solutions should in theory be able to identify all other devices everywhere, but in practice location estimates are variable in urban areas with tall buildings or trees, and location tracking does not work indoors. Distance estimates based on GPS signal is also inaccurate, ranging from several meters to several kilometers, thus inappropriate for contact tracing where disease transmission occurs within meters. Audio based solutions should work well for proximity detection within several meters, and avoids some practical issues of radio signal based solutions (i.e. false detection through walls), but it will require exclusive use of the device speaker and microphone that may be unacceptable to most users and cause noise pollution.

Given the impact of a pandemic, even a poor solution that catches some contacts to assist contact tracing is better than nothing at all. On balance, a Bluetooth based solution is the least disruptive option, thus is also the most popular basis for contact tracing. For testing, a bluetooth beacon app needs to perform the following tasks:

1. Discovery (D) : Detection of new device coming into range.
2. Tracking (T) : Tracking of known device within range.
3. Resume (R) : Resume tracking of known device following a period of being out of range, i.e. device gone out of range and returns. Out of range is simulated by placing the device with screen locked and screen cover on in a Faraday bag (blocks all the radio signals), and then taken out slowly and gently with minimal movement.

An iOS app can be in the following states:

1. Foreground (F) : App is running and being shown on device.
2. Background (B) : App is running but not shown on device, e.g. another app is in foreground. App enters Background state automatically, soon after removal from foreground.
3. Suspended (S) : App is in memory but not actively running, e.g. idle background app. App enters Suspended state automatically, after about 8 seconds of being idle in Background state.
4. Terminated (T) : App is not in memory and not running, e.g. OS taken over background tasks. App will enter Terminated state after a long period of being in Suspended state (i.e. removal by iOS), or alternatively a Terminated state can be triggered by programmatically crashing the app.

An Android app can be in the following states:

1. Foreground (F) : App is running and being shown on device.
2. Background (B) : App is running but not shown on device, e.g. another app is in foreground. App enters Background state automatically, soon after removal from foreground.
3. Terminated (T) : App is not in memory and not running. App will enter Terminated state when the app is closed by the user.

A perfect solution will be able to perform detection, tracking and resume for all iOS and Android devices under all application states. In practice, while an Android app behaves similarly in Foreground and Background states, iOS devices behave differently in different states. The proposed test strategy aims to provide a framework for systematic evaluation of all Bluetooth beacon capabilities across all application states.

C19X Android - Android beacon

Bluetooth functionalities behave the same when an Android app is in Foreground and Background states. No functionality is available when the app is in Terminated state, unless the Bluetooth functionalities are encapsulated in a Background Service that runs separately from the app. For the purpose of evaluation, a Background Service is considered a part of the app, and therefore if the app is terminated, the Background Service is also expected to be terminated.

Android Peripheral / **Android Central**	**Foreground**	**Background**	**Terminated**
Foreground	D, T, R	D, T, R	-
Background	D, T, R	D, T, R	-
Terminated	-	-	-

A key consideration in a practical implementation is that Bluetooth functionalities are likely to malfunction on Android following repeated connect and disconnect calls to a device. This has been observed on Google Pixel 2 and Samsung devices so far, where the Bluetooth functions appear to be operational and discoverable but the actual BLE service no longer accepts connections or characteristics is no longer discoverable. The best mitigation to this problem is to avoid connection to the Android device and only use scan to measure RSSI for known devices.

C19X iOS - Android beacon

An iOS app with Bluetooth state preservation and restoration is able to detect, track and resume tracking of Android devices indefinitely, even when the iOS app has been terminated. The basic mechanism for Android detection is to use repeated calls to `scanForPeripheral(withServices)` where every call will report all the Android devices in the vicinity, and a pending call will trigger a suspended app to return to background state for processing, and state restoration to background state when the iOS app is in terminated state.

Android Peripheral **iOS Central**	**Foreground**	**Background**	**Terminated**
Foreground	D, T, R	D, T, R	-
Background	D, T, R	D, T, R	-
Suspended	D, T, R	D, T, R	-
Terminated	D, T, R	D, T, R	-

iOS uses an ephemeral anonymised device identifier for all Bluetooth devices, which is subject to change, e.g. after a long period of being out of range, or Android device power cycling Bluetooth function. As such, tracking the Android device identifier will minimise the need to connect to the device (e.g. to obtain additional details), but connect calls will be necessary regularly, though infrequently to obtain the details for the same device under a new identity.

C19X Android - iOS beacon

An Android app is able to detect, track and resume tracking of an iOS app with Bluetooth state preservation and restoration, even when the iOS app has been terminated. The iOS device will keep advertising the beacon service, even when the app is terminated. State restoration is triggered when the Android app initiates connection to the iOS app.

iOS Peripheral / Android Central	Foreground	Background	Suspended	Terminated
Foreground	D, T, R	D, T, R	D, T, R	D, T, R
Background	D, T, R	D, T, R	D, T, R	D, T, R
Terminated	-	-	-	-

iOS Bluetooth adverts are different when it is in Foreground state than other states. While the app is in Foreground state, the advert takes a standard form which contains the service UUID, thus making it trivial for detection and filtering by an Android app. In all other states, the service UUID is hidden in an “overflow area” which is not readable by an Android app, thus an Android app will need to filter adverts by manufacturer code and connect to the iOS device to obtain the full details, i.e. via service discovery. Like iOS, Android uses an ephemeral anonymised device identifier, thus it is possible to cache device details to minimise connections, and rely primarily on scan to obtain RSSI (signal strength) updates for distance estimates.

In general, minimising connection to devices will increase the chance of device detection, because the Bluetooth radio is a shared resource on all devices with limited or no ability for simultaneous receive and transmission. In other words, at the hardware level, it can either read or write data, thus it can either scan (listen) or advertise (broadcast). Reducing time performing connection and data exchange will maximise its ability to scan for and register detected peripherals. There is also some practical evidence that WiFi will interfere with Bluetooth, especially on older devices (e.g. iPhone 4S), where switching off WiFi increases the Bluetooth stack’s ability to scan and connect to other devices. Once again, that is to be expected as all wireless capabilities (WiFi, Bluetooth, NFC, Mobile) share the same physical resource on most devices.

C19X iOS - iOS beacon

There are two versions of the C19X iOS beacon. The first does not rely on location permission, as it does not use beacon ranging to trigger the operating system to read the "overflow area" data while the app is in Background, Suspended or Terminated states and the screen is on (e.g. to show a notification). This is a low power implementation of a Bluetooth beacon that relies solely on CoreBluetooth functions. A key limitation of this solution is that new iOS devices are not detected if both iOS devices are in Background, Suspended or Terminated states, which is expected to be the norm. If the app on either device is in Foreground state for a moment, detection will be successful and tracking will continue indefinitely while the devices are within range of each other, even if neither of them are in Foreground state after the initial discovery. Tracking will resume if the devices go out of range for under 20 minutes, however beyond this time, a completely idle device is likely to take on a new identifier and must be discovered as a new device when they are back in range, i.e. requiring one of the devices to return to Foreground state.

In practice, this means most devices won't be discovered, but devices will be tracked once they have been discovered. The discovery is also not transitive, i.e. discovery of one device does not trigger discovery of other devices. Consider a situation where three iOS devices A, B and C come into range of each other while in Background, Suspended or Terminated states. If the app on A enters Foreground state, A will discover B and C, and A will also be discovered by B and C; however B won't discover C and C won't discover B because they are still in Background, Suspended or Terminated state. Numerous attempts have been made to trigger discovery when the app is not in Foreground. Ideas that have been tested and failed included:

a. Using background app refresh or background processing to trigger scan and introduce background processing time.

b. Using another Android or iOS app in Foreground state to trigger discovery, but the non-transitive nature of the discovery process means devices that are not in Foreground state are not discovered.

c. Eliminating all strong references to discovered peripherals to trigger re-discovery of known iOS peripherals.

d. Disposing CoreBluetooth manager objects at regular intervals and reissuing new scan requests to trigger re-discovery of known iOS peripherals.

e. Rotating service UUID being advertised and UUID being scanned at regular intervals to trigger discovery of peripherals offering new services.

f. Rotating service UUID being advertised with known change to iOS background advert bitmask to trigger discovery.

The only solution that has worked consistently well to enable discovery of devices while both iOS apps are not in Foreground state is the use of CoreLocation beacon ranging function which requires the GPS function to be active (see next test).

iOS Peripheral **iOS Central**	**Foreground**	**Background**	**Suspended**	**Terminated**
Foreground	D, T, R	D, T, R	D, T, R	D, T, R
Background	D, T, R	T, R[1]	T, R[1]	T, R[1]
Suspended	D, T, R	T, R[1]	T, R[1]	T, R[1]
Terminated	D, T, R	T, R[1]	T, R[1]	T, R[1]

Note 1: Resume limited to peripheral being away for under 20 minutes.

C19X iOS - iOS beacon with location permission

A solution that enables iOS beacons to detect each other while they are both not in Foreground state is the use of CoreLocation beacon ranging functions to trigger reading of iOS background advert “overflow area” data, when (a) CoreLocation location update is active and (b) screen is on. The former means the GPS is always on, albeit tracking location at a coarse resolution (3km), but still consumes more battery power and requiring the user to agree to location permission (like Android). The latter means the display is lit for any reason, e.g. open case, raise to wake, display notification, normal use.

In practice, this solution means the app will discover other iOS devices in Background, Suspended or Terminated states whenever the device is active for any reason, e.g. during normal use. Experiments have been conducted to reveal that even a relatively light phone user unlocks the phone once every 27 minutes on average during waking hours, without being triggered by any messages or social media notifications. The screen is on for about 2 hours per day in total. As such this solution offers many opportunities for device discovery under normal usage conditions. This can be further improved by using a repeating self-cancelling local notification to trigger brief screen on at regular intervals, however this will only work if the device is not covered, i.e. open on a desk, not in a pocket or a bag.

iOS Peripheral iOS Central	Foreground	Background	Suspended	Terminated
Foreground	D, T, R	D, T, R	N/A	D, T, R
Background	D, T, R	D[1], T, R[2]	N/A	D[1], T, R[2]
Suspended	N/A	N/A	N/A	N/A
Terminated	D, T, R	D[1], T, R[2]	N/A	D[1], T, R[2]

Note 1: Discovery requires screen ON to trigger reading of overflow area.

Note 2: Resumes while peripheral has the same identifier (circa 20 minutes), but requires screen ON for re-discovery of peripheral after identifier change.

The app will never be in Suspended state because location update is active, thus a closed app will only be in Background or Terminated states. Experiments have been conducted to avoid the use of active location updates while maintaining the ability to read the overflow area data. Ideas that have been tested and failed include:

1. No location update means screen on does not trigger data read.
2. Region monitoring was insufficient, even when the region is updated upon region exit to change the region of interest.

3. Beacon monitoring was insufficient, no detection was triggered.
4. Visits monitoring was insufficient.

Only the combination of beacon ranging and location update enables device discovery in Background and Terminated states. Furthermore, discovery is one-sided, i.e. given two devices A and B, both in Background (or Terminated) state, if the screen comes on for A, B will be discovered by A because the Bluetooth daemon process in A has been triggered (bluetoothd process shows "screen state is on" in Console log) to read the overflow area data. However, B will not discover A in this instance as its screen is still off. This problem is solved by a process where A connects to B after screen on and discovery and then subscribes to a notifying characteristic offered by B. This sequence of activities will bring B to Background state (if terminated), and trigger a callback to the peripheral manager delegate function `didSubscribeToCharacteristic` which includes a reference to the subscribing central (i.e. A). The reference can now be used by B to obtain the identifier for A and to initiate connection to enable two way connection even when only one of the two devices has the screen on. In other words, this solution has double the chance of detection being triggered as only one of the two devices need to be active to trigger detection. In practice, detection will only fail if both devices are completely idle, has screen cover on, and in a locked state for the full duration of an encounter.