Understanding the core of AarogyaSetu App : Bluetooth

Niharika Arora
Senior Software Engineer, 1mg

Agenda

- *Origin Story : AarogyaSetu*
- *COVID Tracing*
- *Bluetooth Low Energy(BLE) & Advantages over Classic Bluetooth*
- *How BLE advertisement and scanning works?*
- *Technical Challenges & Solutions*

Origin Story: AarogyaSetu

- **Manually tracing** the contacts is difficult as it is always dependent on a **person's memory**.

Origin Story: AarogyaSetu

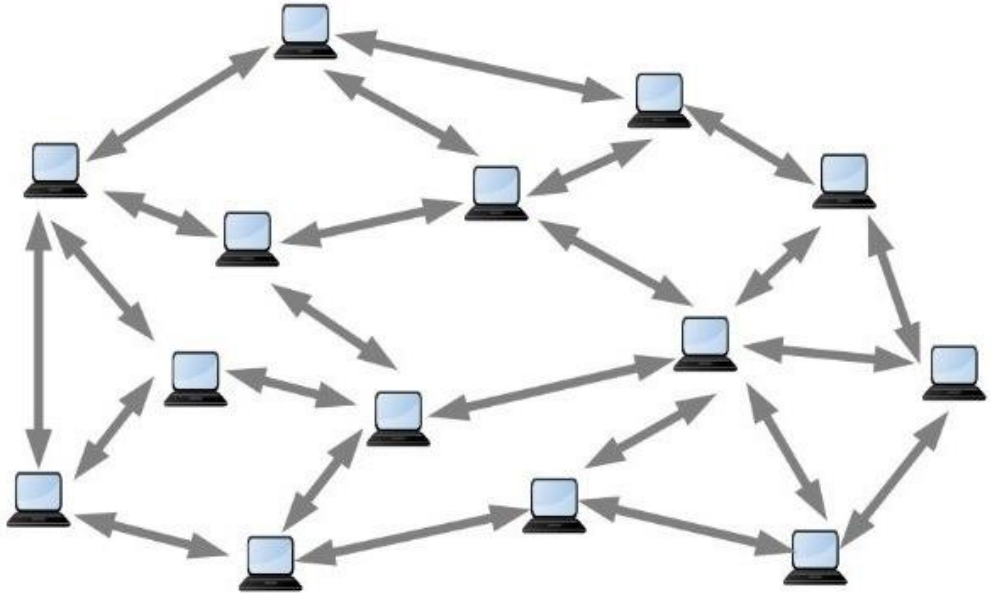
- **Manually tracing** the contacts is difficult as it is always dependent on a **person's memory**.
- Need of **technology** intervention.

Origin Story: AarogyaSetu

- **Manually tracing** the contacts is difficult as it is always dependent on a **person's memory**.
- Need of **technology** intervention.
- **Aarogya Setu** was born with an idea of automatic contact tracing.

COVID TRACING??

COVID TRACING



COVID TRACING

Problem:

How can people get to know whether they can affected by the person whom they came in close contact with or not?

COVID TRACING

Problem:

How can people get to know whether they can affected by the person whom they came in close contact with?

Solution:

Bluetooth v/s GPS

- Bluetooth is able to classify close contacts with a significantly **lower false-positive rate** than GPS.
- Given that **GPS** accuracy decreases in indoor environments, entire shopping malls or skyscrapers would be within the margin of error of a single GPS point.

COVID TRACING

Problem:

How can people get to know whether they can affected by the person whom they came in close contact with?

Solution:

Bluetooth

Problem: Covid Tracing

- Detection

Problem: Covid Tracing

- Detection
- The Communication

Problem: Covid Tracing

- Detection
- The Communication: Scanning and Advertisement through BLE.

Problem: Covid Tracing

- Detection
- The Communication: Scanning and Advertisement through BLE.
- Notification

Classic Bluetooth v/s BLE

What is Bluetooth Low Energy (BLE)?

A low power wireless communication technology that can be used over a short distance to enable smart devices to connect & communicate.

Classic BT v/s BLE

	Classic Bluetooth Technology	BLE Technology
Data Payload Throughput	2 Mbps	~100 kbps
Connection Setup speed	Weak	Strong
Power Consumption	High	Low
Large Scale Network	Weak	Good

BLE Device Roles

BLE Advertisement

BLE Advertisement

- Broadcasting data packets to all nearby devices without having to establish a connection.

BLE Advertisement

- Broadcasting data packets to all nearby devices without having to establish a connection.
- Have **lower power consumption.**

BLE Advertisement

- Broadcasting data packets to all nearby devices without having to establish a connection.
- Have **lower power consumption.**
- Strict limit of **31 bytes** of advertisement data.

PreRequisites for Advertisement

- Bluetooth must be **ON**.

PreRequisites for Advertisement

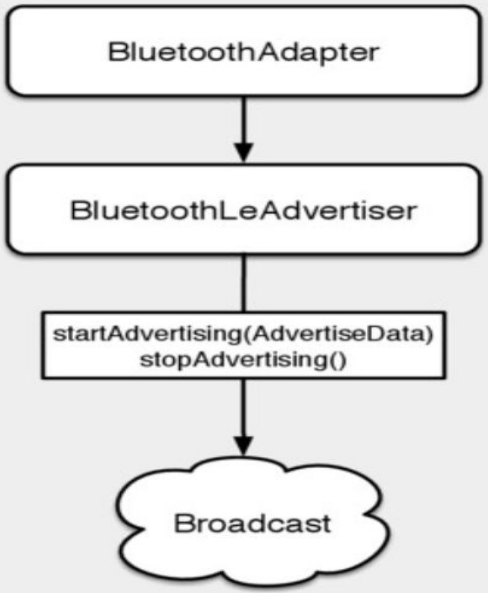
- Bluetooth must be **ON**.
- BluetoothAdapter.**isMultipleAdvertisementSupported**()

PreRequisites for Advertisement

- Bluetooth must be **ON**.
- BluetoothAdapter.**isMultipleAdvertisementSupported**()
- ***Permission***:

  android.Manifest.permission#BLUETOOTH_ADMIN
  android.Manifest.permission#BLUETOOTH

Advertising



AdvertiseData

Device Name
TX Power Level
Manufacturer Data
Service UUIDs
Service Data

AdvertiseSettings

TX Power Level
Connectable
Timeout
Latency Mode

Advertising

```
val defaultAdapter = BluetoothAdapter.getDefaultAdapter()
```

Advertising

```
val defaultAdapter = BluetoothAdapter.getDefaultAdapter()
val advertiser = defaultAdapter.bluetoothLeAdvertiser
```

Advertising

```
val defaultAdapter = BluetoothAdapter.getDefaultAdapter()
val advertiser = defaultAdapter.bluetoothLeAdvertiser
advertiser?.startAdvertising(advertiserSettings, advertiserData, advertisingCallback)
```

Advertising

```
val defaultAdapter = BluetoothAdapter.getDefaultAdapter()
val advertiser = defaultAdapter.bluetoothLeAdvertiser
advertiser?.startAdvertising(advertiserSettings, advertiserData, advertisingCallback)
```

AdvertiseSettings

```
val settingsBuilder = AdvertiseSettings.Builder()
```

AdvertiseSettings

```
val settingsBuilder = AdvertiseSettings.Builder()
                        .setAdvertiseMode(advertisementMode)
```

ADVERTISE_MODE

- **ADVERTISE_MODE_LOW_POWER**

  * *Default and preferred advertising mode. Frequency of advertisement will be less. Advertising interval for 1 packet is 1000 ms i.e 1 sec.*

ADVERTISE_MODE

- **ADVERTISE_MODE_LOW_POWER**

  * *Default and preferred advertising mode. Frequency of advertisement will be less. Advertising interval for 1 packet is 1000 ms i.e 1 sec.*

- **ADVERTISE_MODE_BALANCED**

  * *Balanced between advertising frequency and power consumption. Advertising interval for 1 packet is 250 ms.*

ADVERTISE_MODE

- **ADVERTISE_MODE_LOW_POWER**

  * *Default and preferred advertising mode. Frequency of advertisement will be less. Advertising interval for 1 packet is 1000 ms i.e 1 sec.*

- **ADVERTISE_MODE_BALANCED**

  * *Balanced between advertising frequency and power consumption. Advertising interval for 1 packet is 250 ms.*

- **ADVERTISE_MODE_LOW_LATENCY**

  * *Makes your device discoverable quickly, but with* the highest power consumption*,Advertising interval for 1 packet is 100 ms.*

Which mode we used?

A Mix of All three.

Why?

Will explain in a bit.

AdvertiseSettings

```
val settingsBuilder = AdvertiseSettings.Builder()
                .setAdvertiseMode(advertisementMode)
                .setTxPowerLevel(AdvertiseSettings.ADVERTISE_TX_POWER_ULTRA_LOW)
```

AdvertiseSettings

a) **TX_POWER_LEVEL : *transmission (TX) power level***
Defines the visibility range of advertising packets

Different Levels:

- ADVERTISE_TX_POWER_ULTRA_LOW
- ADVERTISE_TX_POWER_LOW
- ADVERTISE_TX_POWER_MEDIUM
- ADVERTISE_TX_POWER_HIGH

AdvertiseSettings

a) **TX_POWER_LEVEL** : *transmission (TX) power level*
Defines the visibility range of advertising packets

Different Levels:

- ADVERTISE_TX_POWER_ULTRA_LOW
- ADVERTISE_TX_POWER_LOW
- ADVERTISE_TX_POWER_MEDIUM
- ADVERTISE_TX_POWER_HIGH

AdvertiseSettings

```
val settingsBuilder = AdvertiseSettings.Builder()
                .setAdvertiseMode(advertisementMode)
                .setTxPowerLevel(AdvertiseSettings.ADVERTISE_TX_POWER_ULTRA_LOW)
                .setConnectable(true)
```

AdvertiseSettings

```
val settingsBuilder = AdvertiseSettings.Builder()
                .setAdvertiseMode(advertisementMode)
                .setTxPowerLevel(AdvertiseSettings.ADVERTISE_TX_POWER_ULTRA_LOW)
                .setConnectable(true)
                .build
```

Advertising

```
val defaultAdapter = BluetoothAdapter.getDefaultAdapter()
val advertiser = defaultAdapter.bluetoothLeAdvertiser
advertiser?.startAdvertising(advertiserSettings, advertiserData, advertisingCallback)
```

AdvertiseData

```
val data = AdvertiseData.Builder()
```

AdvertiseData

- Service UUID
- Name
- Manufacturer Specific Data
- TX Power Level
- Service Data

AdvertiseData

```
val data = AdvertiseData.Builder()
            .setIncludeDeviceName(true)
```

.

AdvertiseData

```
val data = AdvertiseData.Builder()
            .setIncludeDeviceName(true)
            .addServiceUuid(ParcelUuid(UUID.fromString(“your UUID”)
```

.

AdvertiseData

```
val data = AdvertiseData.Builder()
            .setIncludeDeviceName(true)
            .addServiceUuid(pUuid)
            .build()
```

.

Advertising

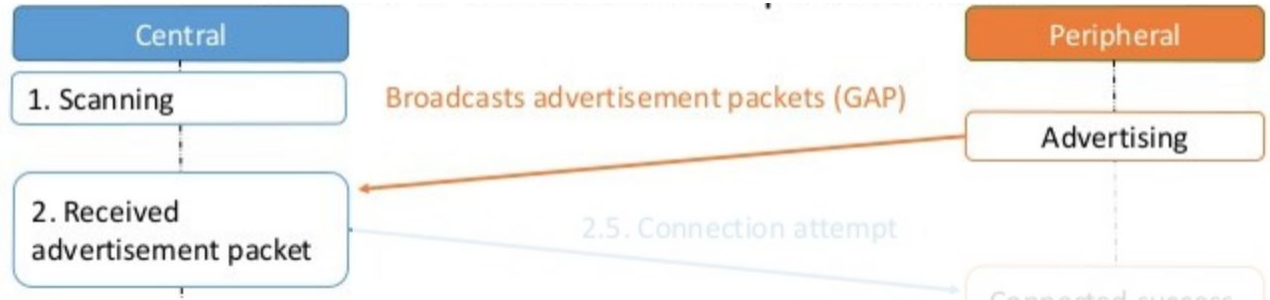
```
val defaultAdapter = BluetoothAdapter.getDefaultAdapter()
val advertiser = defaultAdapter.bluetoothLeAdvertiser
advertiser?.startAdvertising(advertiserSettings, advertiserData, advertisingCallback)
```

Advertising Callback

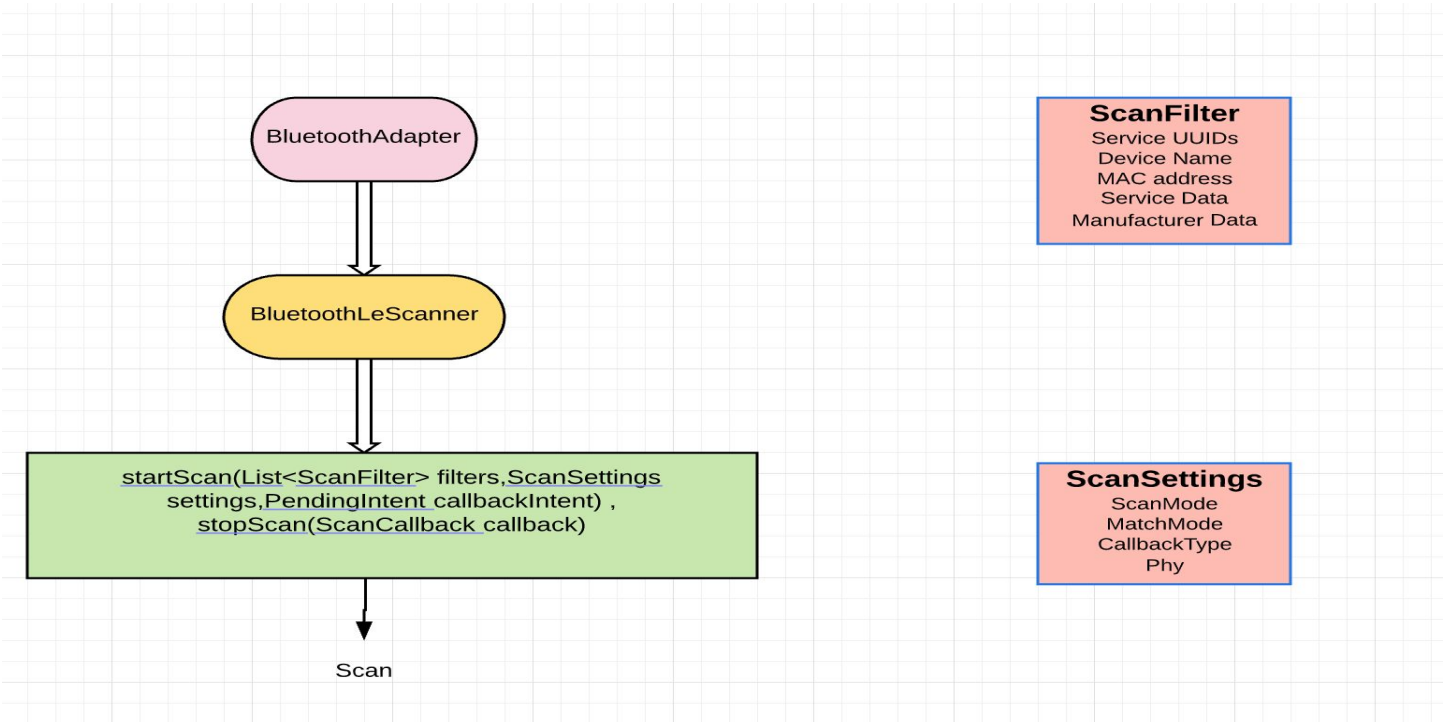
- **onStartSuccess**(AdvertiseSettings settingsInEffect){ ……. }
- **onStartFailure**(**int** errorCode) { …… }

BLE Scan

BLE Scan


Central
Peripheral
1. Scanning
Broadcasts advertisement packets (GAP)
Advertising
2. Received advertisement packet
2.5. Connection attempt

BLEScan


BluetoothAdapter
BluetoothLeScanner
startScan(List<ScanFilter> filters,ScanSettings settings,PendingIntent callbackIntent) , stopScan(ScanCallback callback)
Scan
ScanFilter
Service UUIDs
Device Name
MAC address
Service Data
Manufacturer Data
ScanSettings
ScanMode
MatchMode
CallbackType
Phy

PreRequisites for Scanning

- Bluetooth must be **ON**.
- **Permission**: android.Manifest.permission**#ACESS_COARSE_LOCATION** (Android 9 or lower) or Permission: android.Manifest.permission**#ACESS_FINE_LOCATION**
- **Permission**: android.Manifest.permission**#BLUETOOTH_ADMIN**
- **Permission**: android.Manifest.permission**#BLUETOOTH**

A Curious Relationship: Android BLE and Location

- The Android BluetoothLeScanner's API documentation of startScan(List<ScanFilters>, ScanSettings, ScanCallback) method states:

  An app must hold **ACCESS_COARSE_LOCATION** or **ACCESS_FINE_LOCATION**.

  Bluetooth scan can be used to gather information about the location of the user.

BLE Scan

```
val adapter = BluetoothAdapter.getDefaultAdapter() ?: return
```

BLE Scan

```
val adapter = BluetoothAdapter.getDefaultAdapter() ?: return
val bluetoothLeScanner = adapter.bluetoothLeScanner
```

BLE Scan

```
val adapter = BluetoothAdapter.getDefaultAdapter() ?: return
val bluetoothLeScanner = adapter.bluetoothLeScanner
bluetoothLeScanner?.startScan(scanFilter, scanSettings,scanCallback)
```

BLE Scan

```
val adapter = BluetoothAdapter.getDefaultAdapter() ?: return
val bluetoothLeScanner = adapter.bluetoothLeScanner
bluetoothLeScanner?.startScan(scanFilter, scanSettings,scanCallback)
```

ScanFilter

```
val filter = ScanFilter.Builder()
```

ScanFilter

- Service UUIDs
- Name
- Mac address
- Service data
- Manufacturer specific data

ScanFilter

```
val filter = ScanFilter.Builder()
                .setServiceUuid(ParcelUuid(UUID.fromString(“your UUID”))
```

AdvertiseData

```
val data = AdvertiseData.Builder()
            .setIncludeDeviceName(true)
            .addServiceUuid(ParcelUuid(UUID.fromString(“your UUID”)
```

.

ScanFilter

```
val filter = ScanFilter.Builder()
                .setServiceUuid(ParcelUuid(UUID.fromString(BuildConfig.SERVICE_UUID)))
                .build()
```

BLE Scan

```
val adapter = BluetoothAdapter.getDefaultAdapter() ?: return
val bluetoothLeScanner = adapter.bluetoothLeScanner
bluetoothLeScanner?.startScan(scanFilter, scanSettings,scanCallback)
```

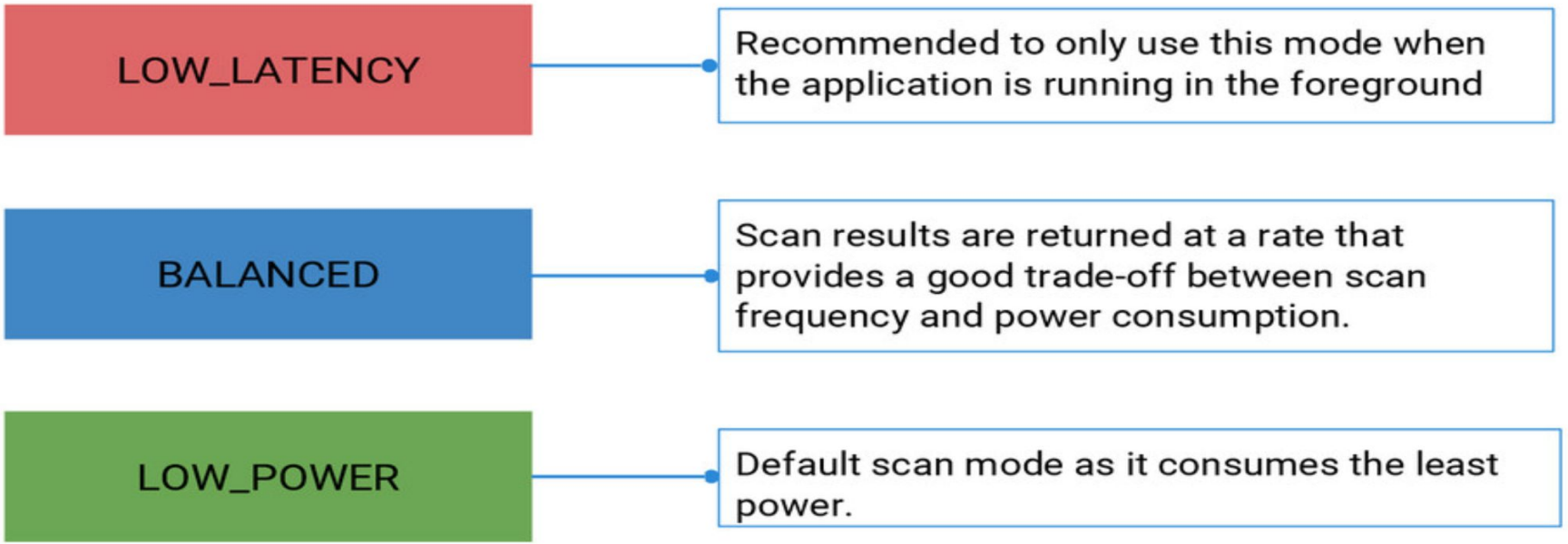
ScanSettings

```
val settings = ScanSettings.Builder()
```

ScanSettings

```
val settings = ScanSettings.Builder()
                    .setScanMode(scanMode)
```

BLE Scan Mode



	LOW_POWER	LOW_LATENCY
Test Duration	134 minutes	152 minutes
Battery Level Change	-13%	-25%
Battery Drain Rate*	268mA	454mA
Relative Battery Savings	41%	–
Typical time between detections	4400 ms	100 ms

Which mode we used?

A Mix of All three.

Why?

Will explain in a bit.

ScanSettings

```
val settings = ScanSettings.Builder()
                .setScanMode(scanMode)
                .setPhy(BluetoothDevice.PHY_LE_1M)
```

ScanSettings

PHY :

- Lowest(Physical) layer of the Bluetooth low energy protocol stack.
- Configures the range of radio transmission and reception.

setPhy

- PHY_LE_1M
- PHY_LE_ALL_SUPPORTED
- PHY_LE_2M
- PHY_LE_CODED
- PHY_LE_1M_MASK
- PHY_LE_2M_MASK
- PHY_LE_CODED_MASK

setPhy

- PHY_LE_1M
- **PHY_LE_ALL_SUPPORTED**
- PHY_LE_2M
- PHY_LE_CODED
- PHY_LE_1M_MASK
- PHY_LE_2M_MASK
- PHY_LE_CODED_MASK

.

setPhy

- **PHY_LE_1M**
- PHY_LE_ALL_SUPPORTED
- PHY_LE_2M
- PHY_LE_CODED
- PHY_LE_1M_MASK
- PHY_LE_2M_MASK
- PHY_LE_CODED_MASK

BLE Scan

```
val adapter = BluetoothAdapter.getDefaultAdapter() ?: return
val bluetoothLeScanner = adapter.bluetoothLeScanner
bluetoothLeScanner?.startScan(scanFilter, scanSettings,scanCallback)
```

ScanCallback

Methods:

- **onScanResult**(**int** callbackType, ScanResult result) { .... }
  Returns single scan result at a time

onScanResult()

Hold various useful pieces of information:

- **BluetoothDevice**: Name and address
- **RSSI**: Received signal strength indication
- **Timestamp**
- **ScanRecord**
  - Advertisement Flags: Discoverable mode and capabilities of the device like **txPowerLevel**
  - Manufacturer Specific Data: Info useful when filtering
  - Service UUIDs

onScanResult()

Hold various useful pieces of information:

- **BluetoothDevice**: Name and address
- **RSSI**: Received signal strength indication
- **Timestamp**
- **ScanRecord**
  - Advertisement Flags: Discoverable mode and capabilities of the device like **txPowerLevel**
  - Manufacturer Specific Data: Info useful when filtering
  - Service UUIDs

RSSI (Received signal strength indication)

- Valid range is [-127, 126]
- TXpower is the **RSSI** value at 1m distance of your beacon.

ScanCallback

Methods:

- **onScanResult**(**int** callbackType, ScanResult result) { .... }
  Returns single scan result at a time

- **onBatchScanResults** (List<ScanResult> results) { …. }
  Queue up and deliver the scan results after the requested delay

ScanCallback

Methods:

- **onScanResult**(**int** callbackType, ScanResult result) { .... }
  Returns single scan result at a time

- **onBatchScanResults** (List<ScanResult> results) { …. }
  Queue up and deliver the scan results after the requested delay

- **onScanFailed**(**int** errorCode) { ….. }
  Callback when scan could not be started with the error code for cause.

Technical Challenges
&
Solutions

Technical Challenges

- Collision Handling & Packets Dropping
- iOS to iOS & iOS to Android Background Scanning limitations.
- Android to iOS Background Scanning limitations
- Android 7 BLE Scan TimeOut
- Other Bluetooth Vulnerabilities

Collision Handling & Packets Dropping

Problem:

How to handle **collision handling** and **packets dropping** when there are large number devices around for scanning as well as advertisement?

Collision Handling & Packets Dropping

Problem:

How to handle **collision handling** and **packets dropping** when there are large number devices around for scanning as well as advertisement?

Solution:

To counter this, We implemented **Adaptive scanning**.

Adaptive Scanning

- Alternative to **backoff schemes** :

MODES

ADVERTISE_MODE SCAN_MODE

MODES

Adaptive Scanning

- Alternative to **backoff schemes** :
- Helped us in adjusting the consumption based on the number of **close contacts.**

Adaptive Scanning

- Alternative to **backoff schemes** :
- Helped us in adjusting the consumption based on the number of **close contacts.**
- **Better scanning performance** and covering more people around.

Resolved!!

iOS to iOS & iOS to Android Background Scanning limitations.

Problem:

- As per the iOS limitation, iOS is not able to scan if it goes in **background** due to their framework restrictions.

iOS to iOS & iOS to Android Background Scanning limitations.

Problem:

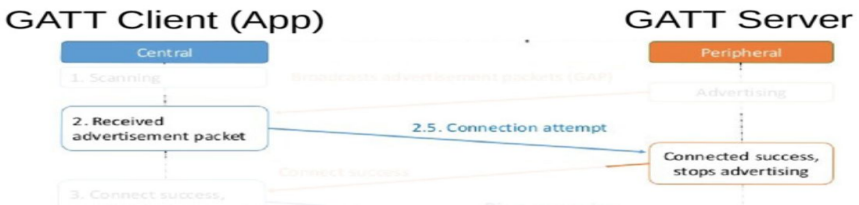
- As per the iOS limitation, iOS is not able to scan if it goes in **background** due to their framework restrictions.

Solution:

GATT server implementation

GATT

General ATTribute profile



AdvertiseSettings

```
val settingsBuilder = AdvertiseSettings.Builder()
                .setAdvertiseMode(advertisementMode)
                .setTxPowerLevel(AdvertiseSettings.ADVERTISE_TX_POWER_ULTRA_LOW)
                .setConnectable(true)
```

GATT

```
bluetoothGattServer = bluetoothManager?.openGattServer(context, gattServerCallback)
```

GATT Example

```
bluetoothGattServer = bluetoothManager?.openGattServer(context, gattServerCallback)
val service = BluetoothGattService(UUID.fromString(BuildConfig.SERVICE_UUID),
BluetoothGattService.SERVICE_TYPE_PRIMARY)
```

GATT Example

```
mBluetoothGattServer = mBluetoothManager?.openGattServer(mContext, mGattServerCallback)
val service = BluetoothGattService(UUID.fromString(BuildConfig.SERVICE_UUID),
BluetoothGattService.SERVICE_TYPE_PRIMARY)
val uniqueIdChar = BluetoothGattCharacteristic(UUID.fromString(BuildConfig.DID_UUID),
BluetoothGattCharacteristic.PROPERTY_READ,BluetoothGattCharacteristic.PERMISSION_READ)
uniqueIdChar.setValue(uniqueId)
val pingerChar = BluetoothGattCharacteristic(UUID.fromString(BuildConfig.PINGER_UUID),
BluetoothGattCharacteristic.PROPERTY_READ,BluetoothGattCharacteristic.PERMISSION_READ)
pingerChar.setValue(true.toString())
```

GATT Example

```
mBluetoothGattServer = mBluetoothManager?.openGattServer(mContext, mGattServerCallback)
val service = BluetoothGattService(UUID.fromString(BuildConfig.SERVICE_UUID),
BluetoothGattService.SERVICE_TYPE_PRIMARY)
val uniqueIdChar = BluetoothGattCharacteristic(UUID.fromString(BuildConfig.DID_UUID),
BluetoothGattCharacteristic.PROPERTY_READ,BluetoothGattCharacteristic.PERMISSION_READ)
uniqueIdChar.setValue(uniqueId)
val pingerChar = BluetoothGattCharacteristic(UUID.fromString(BuildConfig.PINGER_UUID),
BluetoothGattCharacteristic.PROPERTY_READ,BluetoothGattCharacteristic.PERMISSION_READ)
pingerChar.setValue(true.toString())
service.addCharacteristic(uniqueIdChar)
service.addCharacteristic(pingerChar)
```

GATT Example

```
mBluetoothGattServer = mBluetoothManager?.openGattServer(mContext, mGattServerCallback)
val service = BluetoothGattService(UUID.fromString(BuildConfig.SERVICE_UUID),
BluetoothGattService.SERVICE_TYPE_PRIMARY)
val uniqueIdChar = BluetoothGattCharacteristic(UUID.fromString(BuildConfig.DID_UUID),
BluetoothGattCharacteristic.PROPERTY_READ,BluetoothGattCharacteristic.PERMISSION_READ)
uniqueIdChar.setValue(uniqueId)
val pingerChar = BluetoothGattCharacteristic(UUID.fromString(BuildConfig.PINGER_UUID),
BluetoothGattCharacteristic.PROPERTY_READ,BluetoothGattCharacteristic.PERMISSION_READ)
pingerChar.setValue(true.toString())
service.addCharacteristic(uniqueIdChar)
service.addCharacteristic(pingerChar)
bluetoothGattServer?.addService(service)
```

Advantage of implementing GATT server

- iOS GATT client started **pinging** the device in background for connections.
- App remains active and the connection breaks only if the device goes too far.

Resolved!!

Android to iOS Background Scanning limitations

Problem:

iOS app advertises in a **proprietary advertisement** format that is not part of the Bluetooth standard and thus not readable by non-iOS devices.

Android to iOS Background Scanning limitations

Problem:

iOS app advertises in a **proprietary advertisement** format that is not part of the Bluetooth standard and thus not readable by non-iOS devices.

Solution:

Implemented **reverse search** on backend.

Resolved!!

Android 7 BLE Scan TimeOut

Problem:

Android 7.0 introduced a BLE scan timeout, where any scan running for 30 minutes or more is effectively stopped automatically and can only resume “opportunistically”

Android 7 BLE Scan TimeOut

Problem:

Android 7.0 introduced a BLE scan timeout, where any scan running for 30 minutes or more is effectively stopped automatically and can only resume “opportunistically”

Solution:

Starting the scan again after an interval

Resolved!!

Bluetooth Vulnerabilities

There are vulnerabilities in Bluetooth technology that have to be patched at the operating system-level, and we, therefore, urge users to ensure that their operating systems are regularly patched.

That’s All!!

Resources

Github Link -

https://github.com/nic-delhi/AarogyaSetu_Android

Medium link -

https://medium.com/aarogyasetu/understanding-the-core-of-aarogya-setu-bluetooth-c09de3143fd2

Adaptive Scanning Flow Diagram:

https://ibb.co/KGgzzFY

Download Link -

https://play.google.com/store/apps/details?id=nic.goi.aarogyasetu

https://apps.apple.com/in/app/aarogyasetu/id1505825357

Listen to my Podcast

https://www.truepush.com/blog/podcast-ep-9-the-story-behind-the-inception-of-indias-aarogya-setu-app/?fbclid=IwAR3AQK1TDyyUEZHHghHFHwR6hp7M0tFUdagfPi0dBtZMpWFeV3178c5zP6I

Contacts

- LinkedIn : https://www.linkedin.com/in/niharika-arora-4874967a/
- Medium : https://medium.com/@nik.arora8059
- Github : https://github.com/niharika2810
- Twitter : https://twitter.com/Niharik36712833

THANK YOU FOR LISTENING !!

Any Questions ?