

IEMS 5722

Mobile Network Programming and Distributed Server Architecture

Lecture 8

Peer-to-Peer Networking in Android

Lecturer: Albert C. M. Au Yeung

3rd March, 2016

Agenda

- Wireless Communication with Peers
- Peer-to-Peer Wi-Fi Communication
- Bluetooth
- Near Field Communication (NFC)

Wireless Communication with Peers

Wireless Communication with Peers

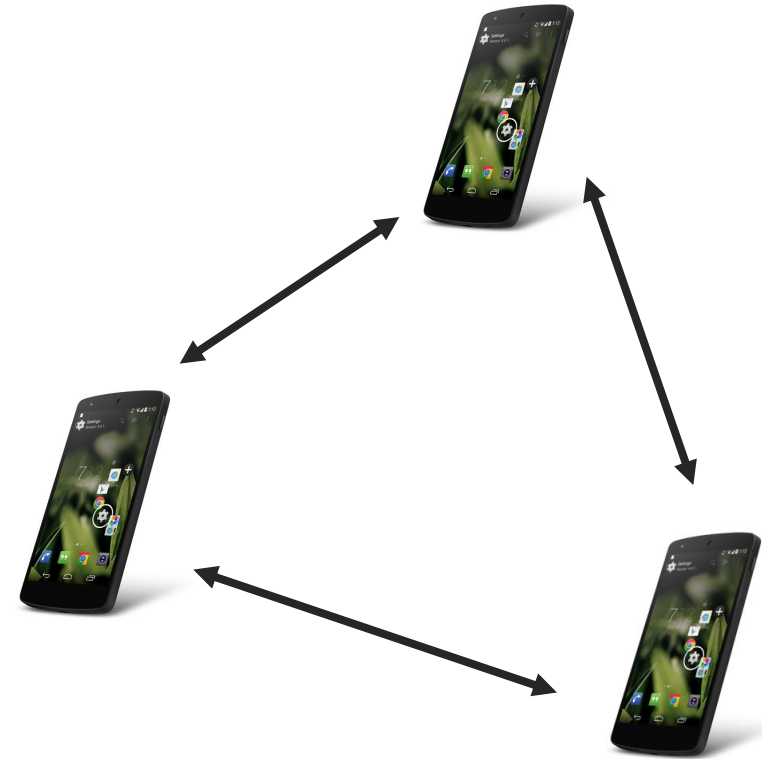
- So far we have mainly discussed the **client-server architecture**
- Clients request data or services from the server through a network (e.g. an Intranet or the Internet)
- There are cases in which we would like to set up **peer-to-peer connections**
 - Sharing files directly with a device nearby
 - Playing multi-player games without an Wi-Fi access point (AP)
 - ...

Wireless Communication with Peers

Client-Server Architecture



Peer-to-Peer Communication



Wireless Communication with Peers

Methods for P2P wireless communication

- Peer-to-peer Wi-Fi (Wi-Fi Direct)
- Bluetooth
- NFC



Wi-Fi

Wi-Fi: “Wireless Fidelity”

- A trademark of the Wi-Fi Alliance
- A family of products using the [IEEE 802.11](#) standards
- Usually used to set up a “wireless local area network” ([WLAN](#))

Protocol	Date of Release	Frequency. (GHz)	Data Rate (Mbps) (Typical / Max)	Range (m) (Indoor/outdoor)
A	Sep 1999	5 / 3.7	20 / 54	35 / 120
B	Sep 1999	2.4	5.5 / 11	35 / 140
G	Jun 2003	2.4	22 / 54	38 / 140
N	Oct 2009	2.4 / 5	110+ / 300+	70 / 250

P2P Wi-Fi / Wi-Fi Direct

- Connects devices directly using Wi-Fi protocols, without requiring a wireless access point (AP)
- Supported in Android 4.0 or above
- Devices communicate with each other by establishing P2P Groups
 - A device will become "Group Owner" (act as an AP in the group)
 - Other device will become "Clients"

Bluetooth



- A wireless solution for data and voice communication over short distances
- Intended for building personal area networks (PANs)
- Range from 1m to 100m, typically 10m

Class	Max Permitted Power (mW)	Typical Range (m)
1	100	~100
2	2.5	~10
3	1	~1

Version	Data Rate (Mbps)
1.2	1
2.0 + EDR (Enhanced Data Rate)	3
3.0 + HS (High Speed)	24
4.0	24

NFC

Near Field Communication (NFC)

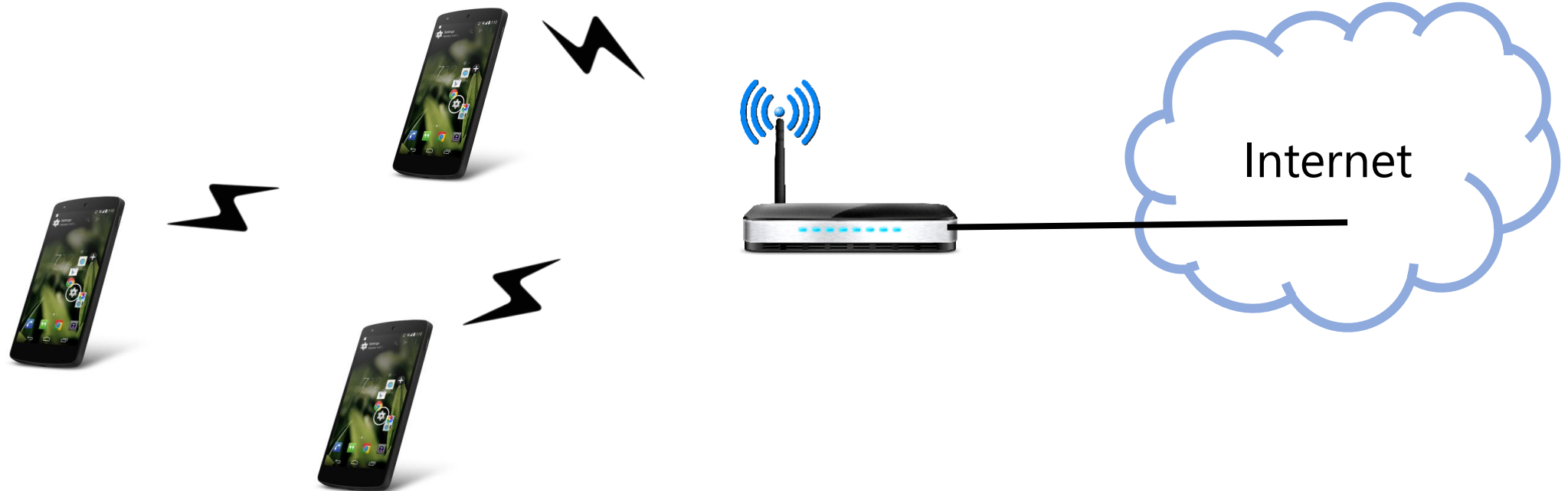
- For short-range (< 10cm) data communication
- One of the **RFID** (Radio Frequency Identification) standards
- Data rate: 106kbps to 424kbps
- Allows **two-way** communication between two devices



Peer-to-Peer Wi-Fi Communication

Common WLAN

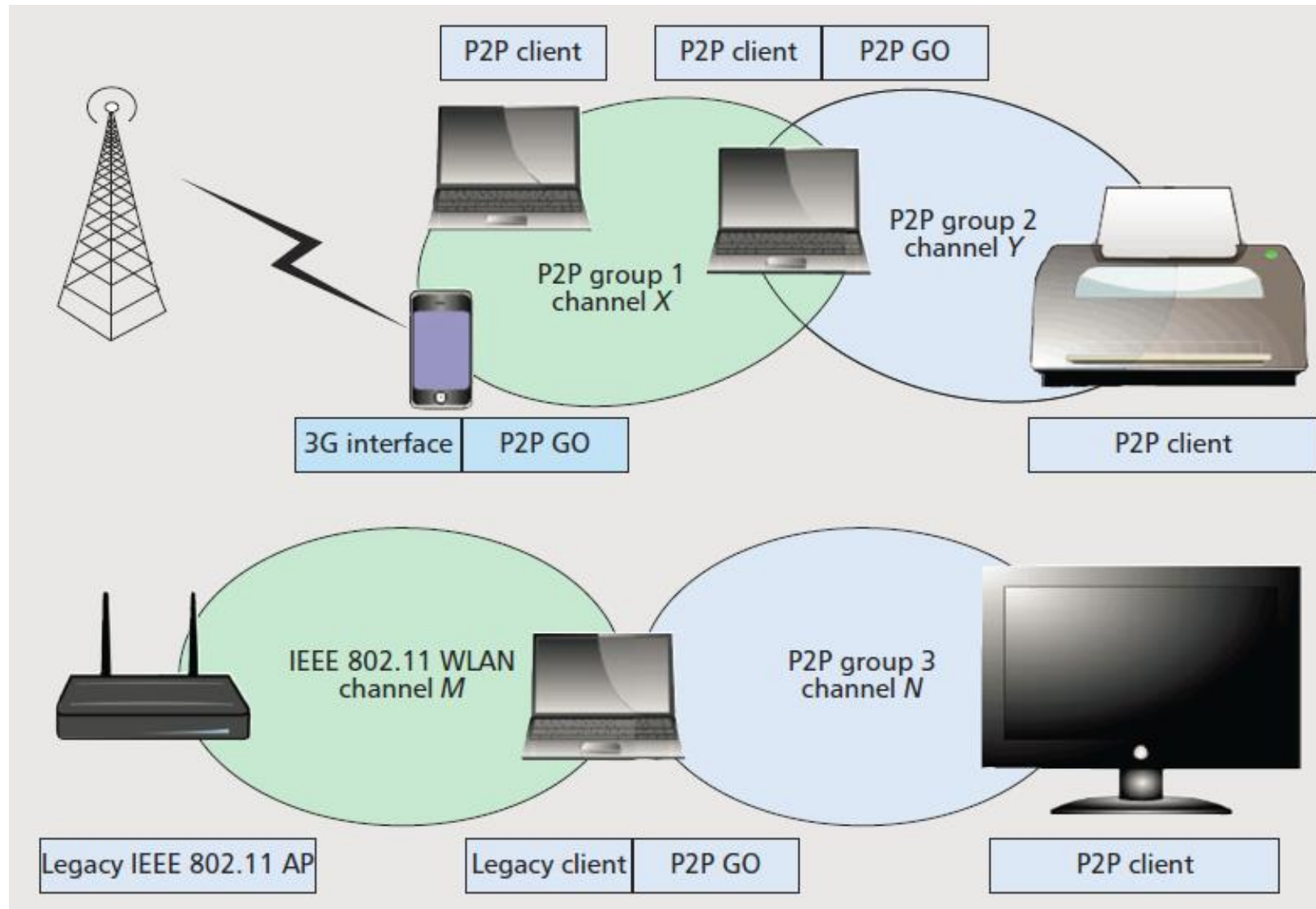
- In a common wireless network, an access point (AP) is present to allow devices to connect to a network
- Devices within the WLAN communicate through the AP, not directly



P2P Wi-Fi

- P2P Wi-Fi allows devices to communicate with one another without the presence of an AP
- Any device may assume the role of an AP
- Communication is done by creating **dynamic groups**
 - Device playing the role of an AP is called the **Group Owner (GO)**
 - Other devices are then **clients** within this group
 - The role of GO cannot be transferred
 - If GO leaves, the group is automatically closed

P2P Wi-Fi



Group Formation in P2P Wi-Fi










- When two devices have discovered each other and want to establish a connection, they will negotiate their roles in the group
- Three group formation technique:
 1. Standard
(Two devices negotiate their roles in the group)
 2. Autonomous
(One device automatically creates a group)
 3. Persistent
(Roles and credentials are stored for future group establishment)

Power Saving in P2P Wi-Fi

- Wi-Fi Direct is specifically designed for mobile devices
- As any devices can potentially become an AP, power saving is very important
- Two mechanisms for saving power in the AP:
 1. **Opportunistic power-saving protocol**
When all clients are sleeping, the AP will go to sleep
 2. **Notice-of-absence protocol**
The AP will announce certain period of time when it will go to sleep to save power, clients are not allowed to access the channel during these times

P2P Wi-Fi Apps

應用程式

 SuperBeam Wi-Fi LiveQoS 免費	 Wifi Direct File Transfer Ever Was Produced 免費	 WiFi Direct + Dev Netcomps 免費	 WiFi File Transfer smarterDroid 免費	 無線網絡直接分享 Galileo Best Free 免費	 Wifi Display (Miracast) ppgirl 免費
 無線文件傳輸 VillaCat 免費	 WiFi File Transfer Smartdroid Solutions 免費	 免費無線網絡連接 Mentisco 免費	 Wi-Fi 高速接續 Gotsun 免費	 Wifi Direct File Transfer John Kim's Apps 免費	 無線網絡文件共享 Designproshop 免費
					

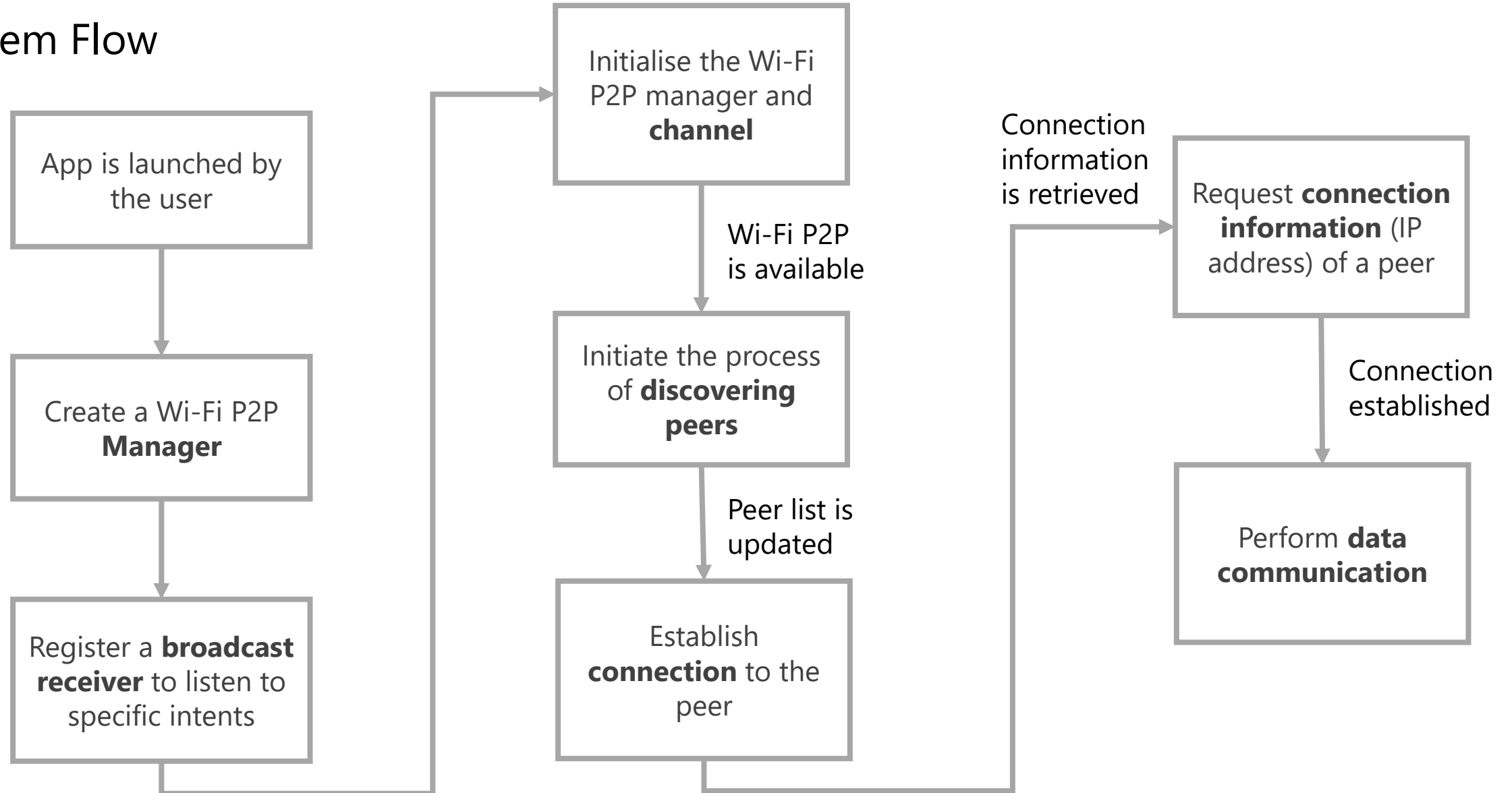
P2P Wi-Fi in Android

P2P Wi-Fi on Android

- The Wi-Fi P2P APIs are available in [Android 4.0](#) or later
- [Four](#) major operations in a Wi-Fi P2P app
 - Creating and registering a [broadcast receiver](#)
 - [Discovering](#) peers
 - [Connecting](#) to a peer
 - [Transferring data](#) to a peer

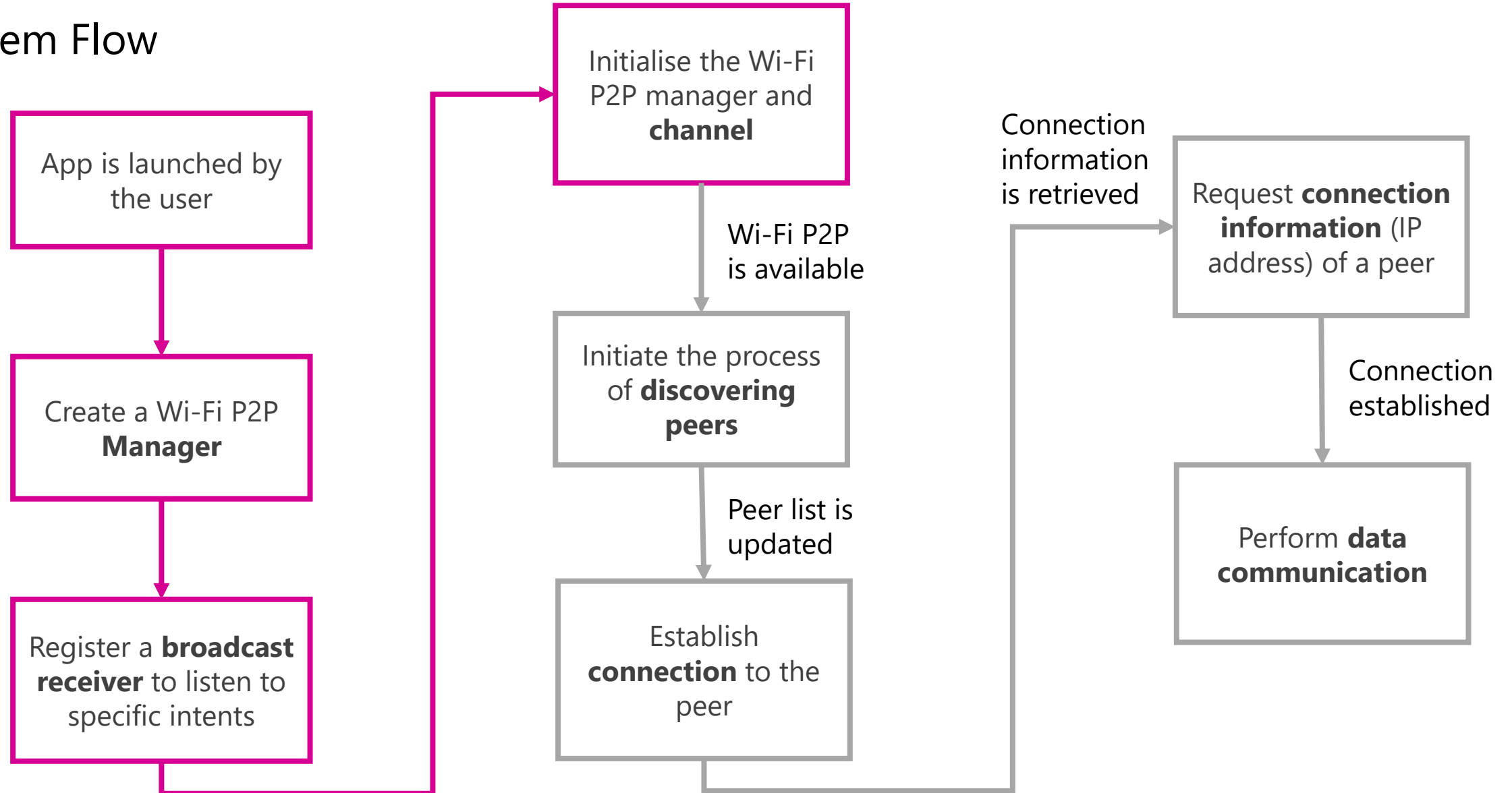
P2P Wi-Fi on Android

System Flow



P2P Wi-Fi on Android

System Flow



P2P Wi-Fi on Android - Setup

P2P Wi-Fi related events that will be broadcasted in the Android system

- **WIFI_P2P_STATE_CHANGED_ACTION**
Indicates whether Wi-Fi P2P is enabled
- **WIFI_P2P_PEERS_CHANGED_ACTION**
Indicates that the available peer list has changed
- **WIFI_P2P_CONNECTION_CHANGED_ACTION**
Indicates the state of Wi-Fi P2P connectivity has changed
- **WIFI_P2P_THIS_DEVICE_CHANGED_ACTION**
Indicates this device's configuration details have changed

P2P Wi-Fi on Android - Setup

You need to implement a broadcast receiver to listen to broadcasts

```
@Override
public void onReceive(Context context, Intent intent) {
    String action = intent.getAction();
    if (WifiP2pManager.WIFI_P2P_STATE_CHANGED_ACTION.equals(action)) {
        // Determine if Wifi P2P mode is enabled or
        int state = intent.getIntExtra(WifiP2pManager.EXTRA_WIFI_STATE, -1);
        if (state == WifiP2pManager.WIFI_P2P_STATE_ENABLED) {
            // Wifi P2P mode is enabled
        } else {
            // Wifi P2P mode is disabled
        }
    } else if (WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION.equals(action)) {
        // The peer list has changed!
    } else if (WifiP2pManager.WIFI_P2P_CONNECTION_CHANGED_ACTION.equals(action)) {
        // Connection state changed!
    } else if (WifiP2pManager.WIFI_P2P_THIS_DEVICE_CHANGED_ACTION.equals(action)) {
        // Configuration of this device has changed!
    }
}
```

P2P Wi-Fi on Android - Setup

Create an intent filter to indicate which intents the broadcast receiver should listen to

```
private final IntentFilter intentFilter = new IntentFilter();

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    // Indicates a change in the Wi-Fi P2P status.
    intentFilter.addAction(WifiP2pManager.WIFI_P2P_STATE_CHANGED_ACTION);

    // Indicates a change in the list of available peers.
    intentFilter.addAction(WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION);

    // Indicates the state of Wi-Fi P2P connectivity has changed.
    intentFilter.addAction(WifiP2pManager.WIFI_P2P_CONNECTION_CHANGED_ACTION);

    // Indicates this device's details have changed.
    intentFilter.addAction(WifiP2pManager.WIFI_P2P_THIS_DEVICE_CHANGED_ACTION);
    ...
}
```


P2P Wi-Fi on Android - Setup

Register the broadcast receiver when the activity starts, and unregister it when the activity is closed

```
/** register the BroadcastReceiver with the intent values to be matched */  
@Override  
public void onResume() {  
    super.onResume();  
    receiver = new WifiDirectBroadcastReceiver(mManager, mChannel, this);  
    registerReceiver(receiver, intentFilter);  
}  
  
@Override  
public void onPause() {  
    super.onPause();  
    unregisterReceiver(receiver);  
}
```

(In the Java source code of the activity)

P2P Wi-Fi on Android - Setup

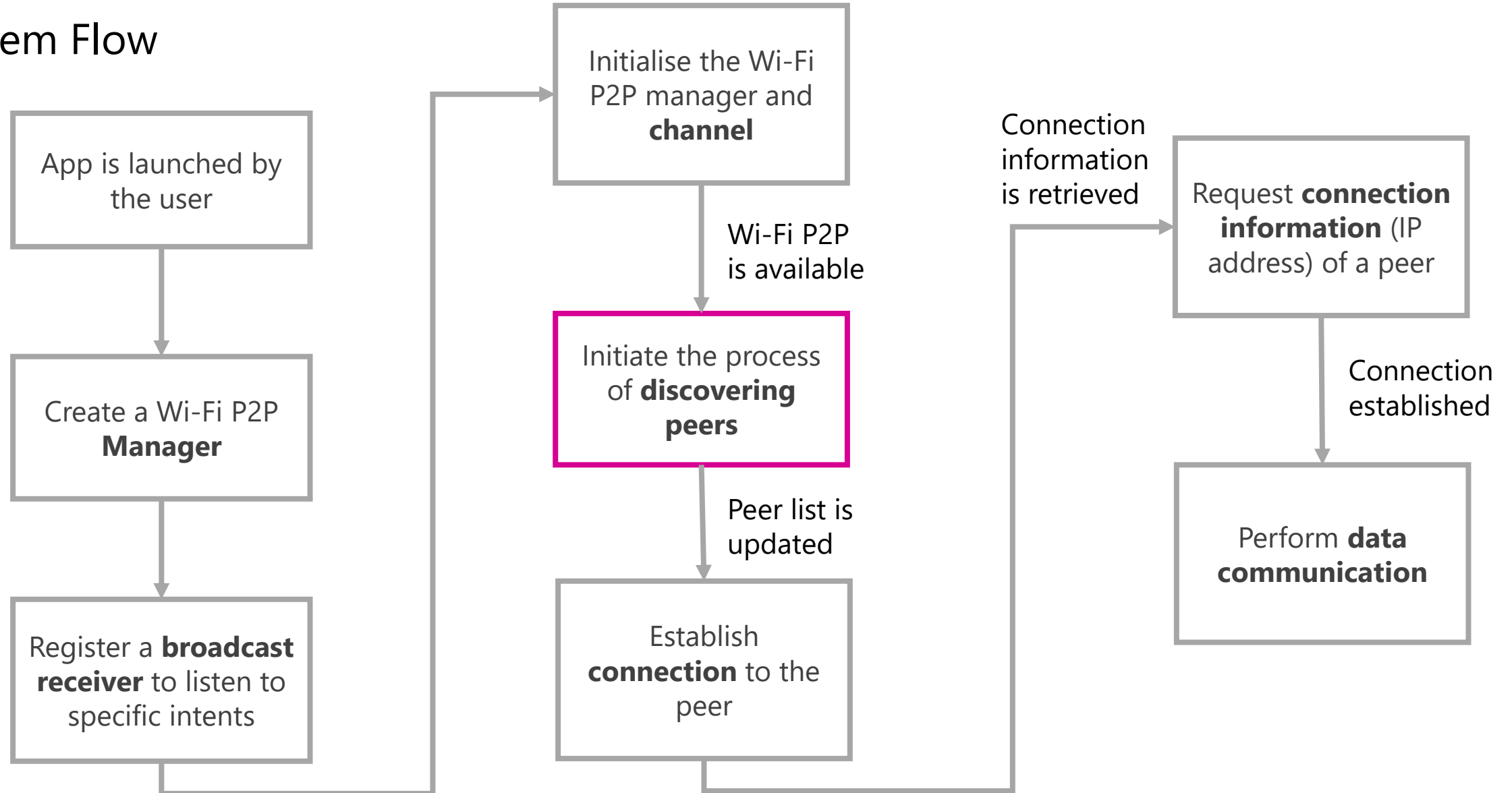
Get the Wi-Fi P2P service manager and initialize it, it will return a channel object which you can use to make connections later

```
Channel mChannel;  
  
@Override  
public void onCreate(Bundle savedInstanceState) {  
    ....  
    mManager = (WifiP2pManager) getSystemService(Context.WIFI_P2P_SERVICE);  
    mChannel = mManager.initialize(this, getMainLooper(), null);  
}
```

(In the Java source code of the activity)

P2P Wi-Fi on Android

System Flow



P2P Wi-Fi on Android – Peers Discovery

Discover peer devices which are nearby (with P2P Wi-Fi enabled)

```
mManager.discoverPeers(mChannel, new WifiP2pManager.ActionListener() {  
  
    @Override  
    public void onSuccess() {  
        // When the discovery action is successful  
        // Usually nothing needs to be put here  
    }  
  
    @Override  
    public void onFailure(int reasonCode) {  
        // When the discovery action is not successful  
        // You may want to alert the user here  
    }  
  
});
```

When peers are discovered, a **WIFI_P2P_PEERS_CHANGED_ACTION** will be broadcasted

P2P Wi-Fi on Android – Peers Discovery

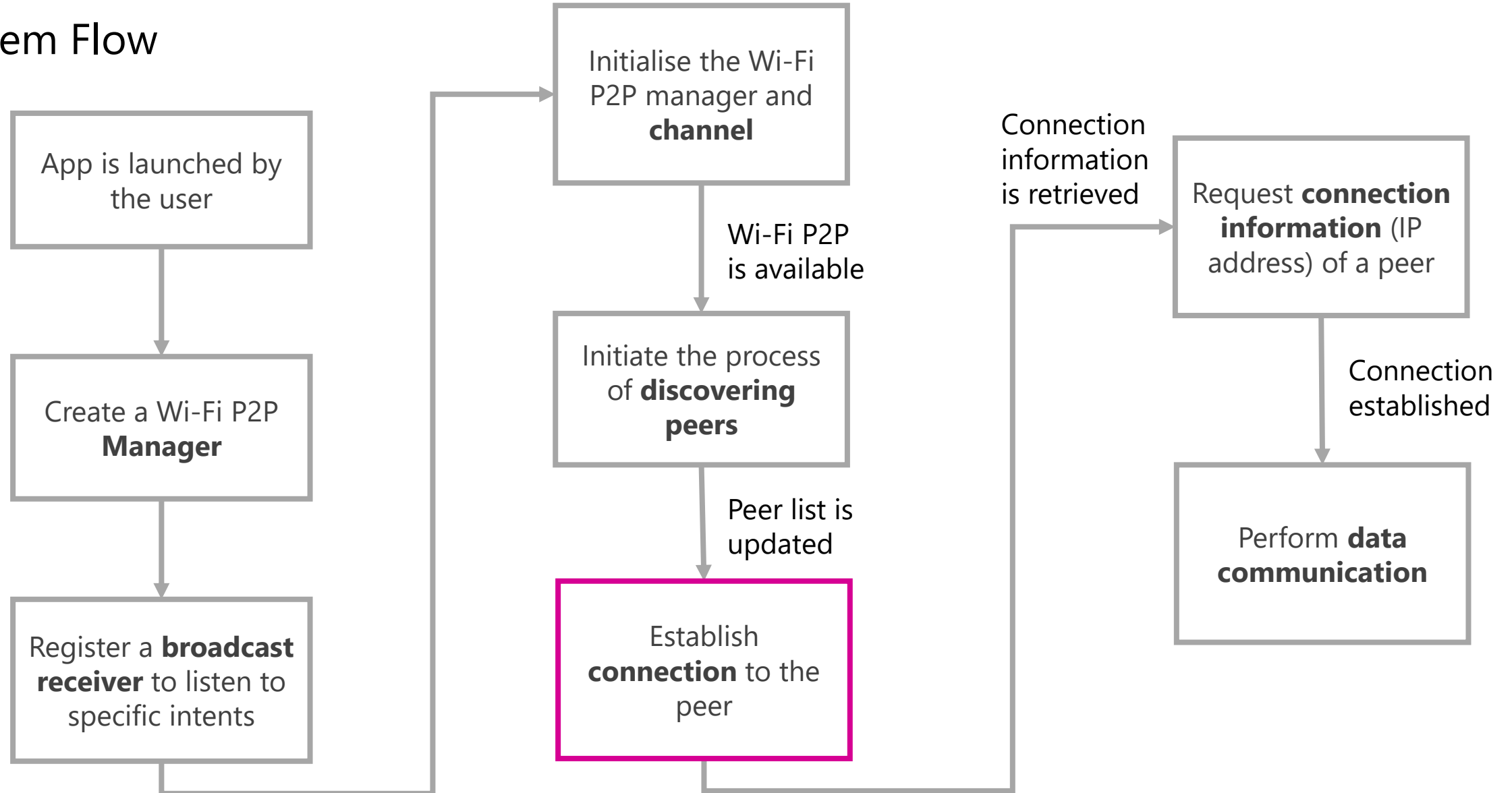
When your broadcast receiver receives the **WIFI_P2P_PEERS_CHANGED_ACTION**, you can call the **requestPeers()** method to retrieve the list of peers

```
if (WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION.equals(action)) {  
    if (mManager != null) {  
        mManager.requestPeers(mChannel, new PeerListListener(){  
            @Override  
            public void onPeersAvailable(WifiP2pDeviceList peerList) {  
                // Get the list of peers and present it to the user  
                List peers = peerList.getDeviceList();  
                ...  
            }  
        });  
    }  
}
```

(Insider the broadcast receiver)

P2P Wi-Fi on Android

System Flow



P2P Wi-Fi on Android – Connecting to a Peer

To connect to a peer, you create a `WifiP2pConfig` object, and use the device information to make a connection

```
WifiP2pDevice device = peers.get(0);
WifiP2pConfig config = new WifiP2pConfig();
config.deviceAddress = device.deviceAddress;
config.wps.setup = WpsInfo.PBC;

mManager.connect(mChannel, config, new ActionListener() {

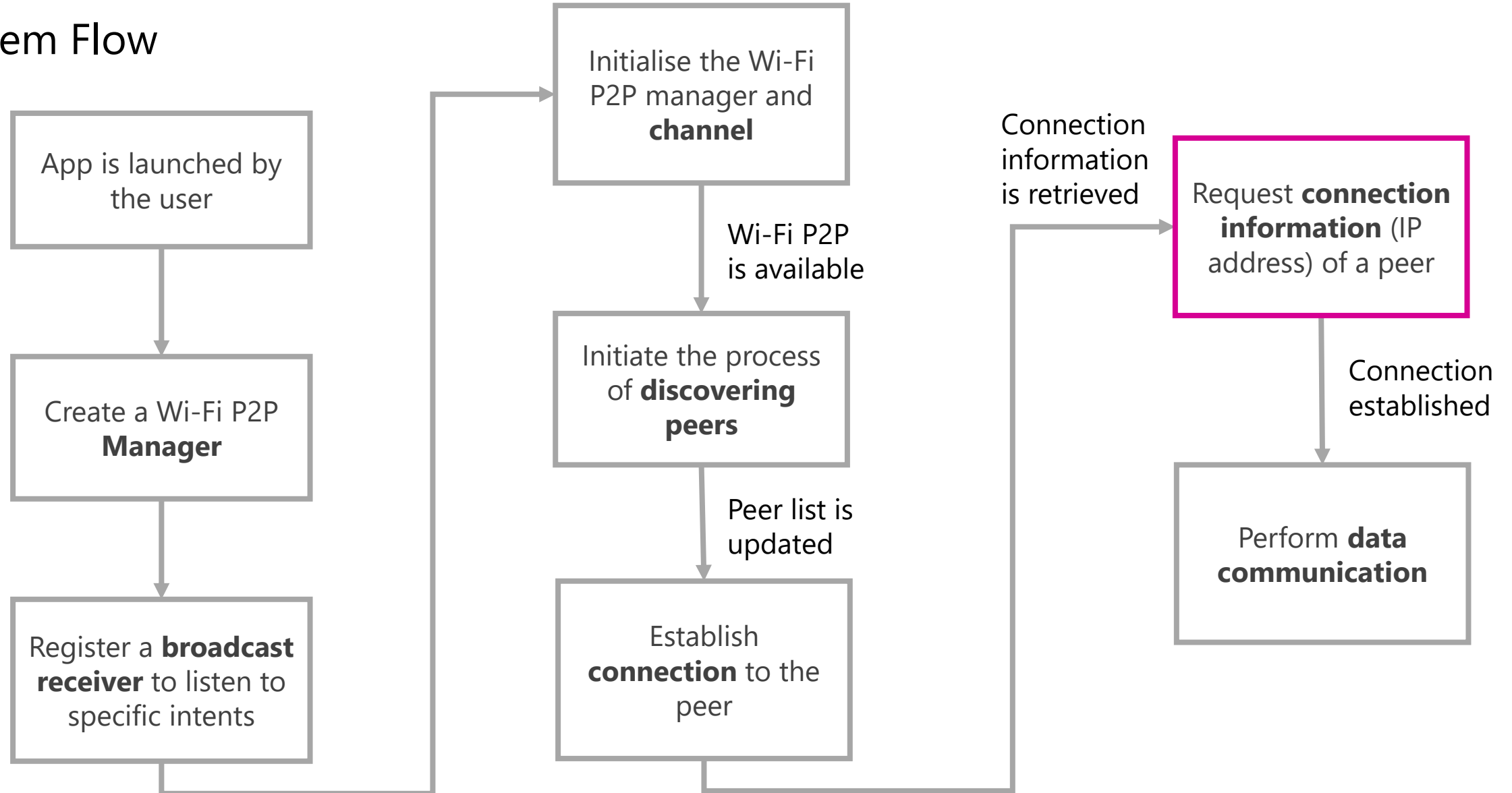
    @Override
    public void onSuccess() {
        // Check action in broadcast receiver, no action needed here
    }

    @Override
    public void onFailure(int reason) {
        // Alert the user that the connection cannot be established
    }

});
```

P2P Wi-Fi on Android

System Flow



P2P Wi-Fi on Android – Getting Connection Information

Once the P2P Wi-Fi connection is established, you can request for information about the connection, this is done in the broadcast receiver

```
...
} else if (WifiP2pManager.WIFI_P2P_CONNECTION_CHANGED_ACTION.equals(action)) {
    if (mManager == null) {
        return;
    }

    NetworkInfo networkInfo = (NetworkInfo)
intent.getParcelableExtra(WifiP2pManager.EXTRA_NETWORK_INFO);
    if (networkInfo.isConnected()) {
        mManager.requestConnectionInfo(mChannel, connectionListener);
    }
    ...
}
```

This method will call the `onConnectionInfoAvailable()` method in this listener when the information is ready (see next slide)

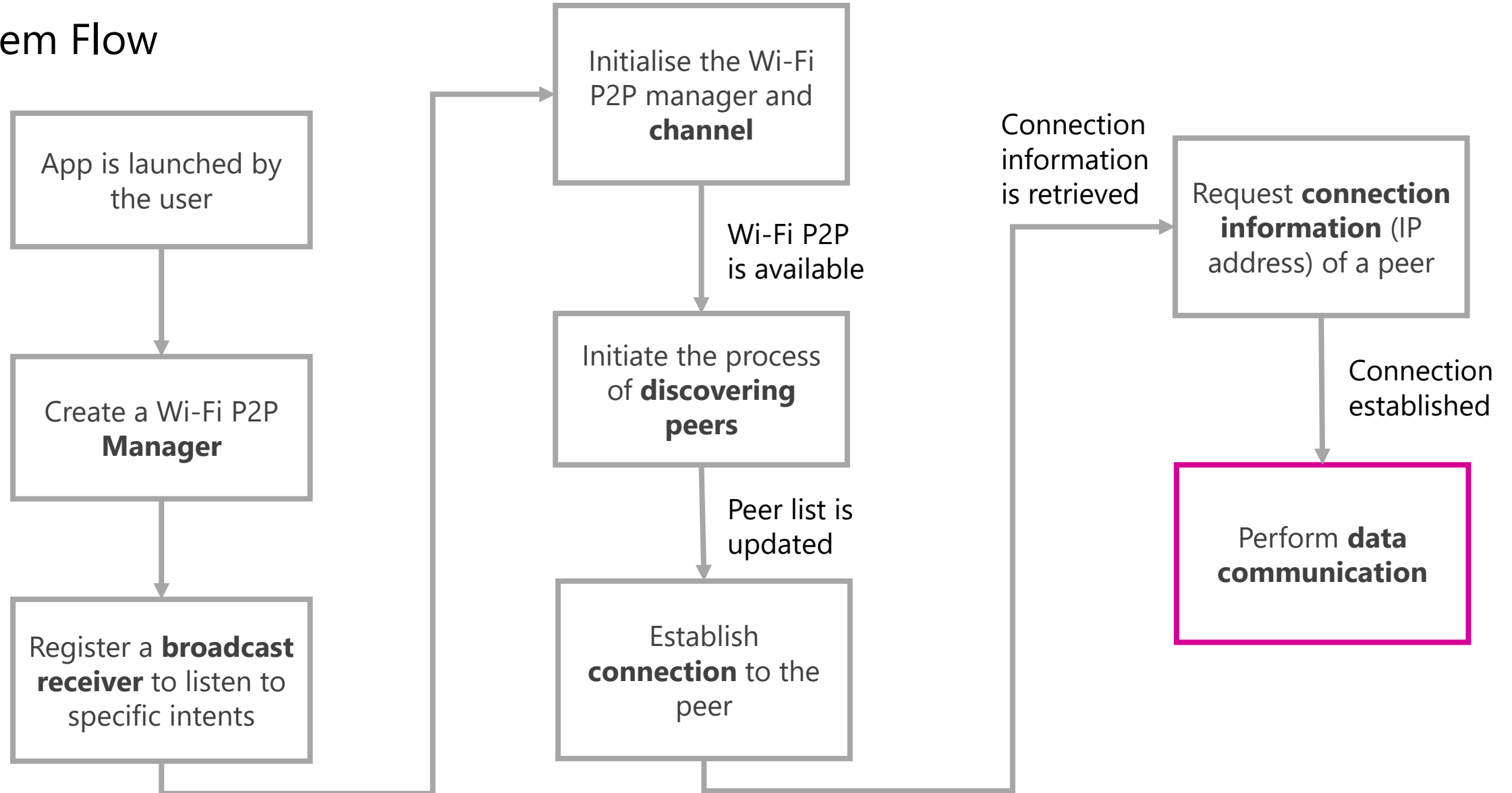
P2P Wi-Fi on Android – Getting Connection Information

Implementing a ConnectionInfoListener

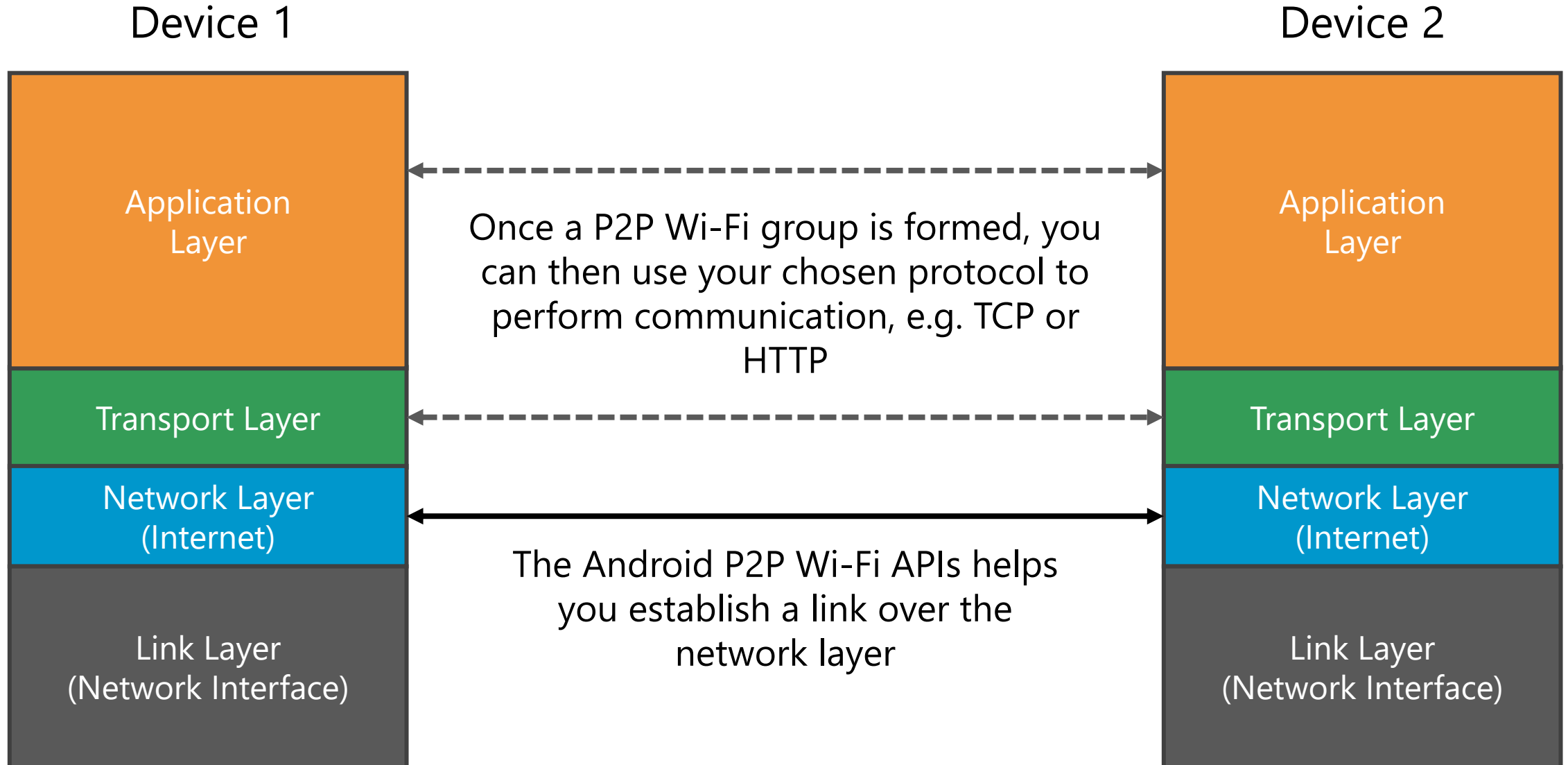
```
ConnectionInfoListener connectionListener = new ConnectionInfoListener() {  
    @Override  
    public void onConnectionInfoAvailable(final WifiP2pInfo info) {  
  
        // InetAddress from WifiP2pInfo struct.  
        InetAddress groupOwnerAddress = info.groupOwnerAddress.getHostAddress();  
  
        // After the group negotiation, we can determine the group owner.  
        if (info.groupFormed && info.isGroupOwner) {  
            // This device is the group owner, start a server thread...  
  
        } else if (info.groupFormed) {  
            // This device is a client, connects to the group owner in a new  
            thread...  
  
        }  
    }  
};
```

P2P Wi-Fi on Android

System Flow



P2P Wi-Fi on Android



P2P Wi-Fi on Android

For more information, refer to the documentation on official Android guide:

- <http://developer.android.com/guide/topics/connectivity/wifip2p.html>

Check the following project for a demo app that use P2P Wi-Fi

- https://android.googlesource.com/platform/development/+/_with/68279ecebc39e93e6ed4f0067edce034c5dd091f/samples/WiFiDirectDemo/
- (You can also download samples using the SDK Manager)

Bluetooth

Bluetooth



- A wireless solution for data and voice communication over short distances
- Range from 1m to 100m, typically 10m
- Use frequencies in the 2.4 GHz range (the globally available, license-free ISM band)

Class	Max Permitted Power (mW)	Typical Range (m)
1	100	~100
2	2.5	~10
3	1	~1

Version	Data Rate (Mbps)
1.2	1
2.0 + EDR (Enhanced Data Rate)	3
3.0 + HS (High Speed)	24
4.0	24

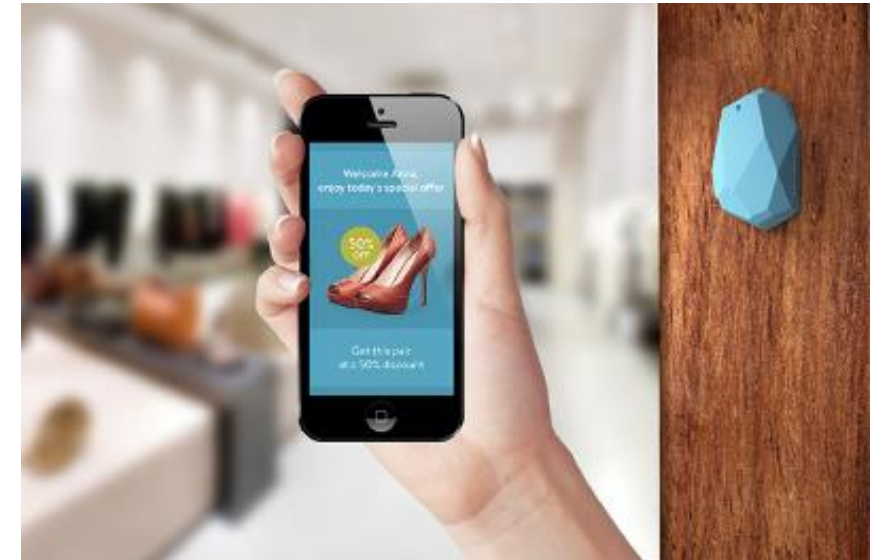
Bluetooth

- The Bluetooth specification is an open specification that is governed by the **Bluetooth Special Interest Group (SIG)**
- The Bluetooth SIG is lead by a group of companies which forms the “**Promoter Group**”: Intel, Ericsson, Microsoft, Toshiba, Lenovo and Nokia
- Initiated by Ericsson in 1994, when it investigated the feasibility of a low power, low cost radio interface between mobile phones and their accessories
- The Bluetooth SIG was officially formed in **1998**

Bluetooth

Common applications

- Wireless mouse and keyboard
- Wireless earphones, headphones and microphones
- File transmission between mobile devices
- Wearable devices (e.g. smart watches)
- Sensing (e.g. iBeacon technology by Apple)



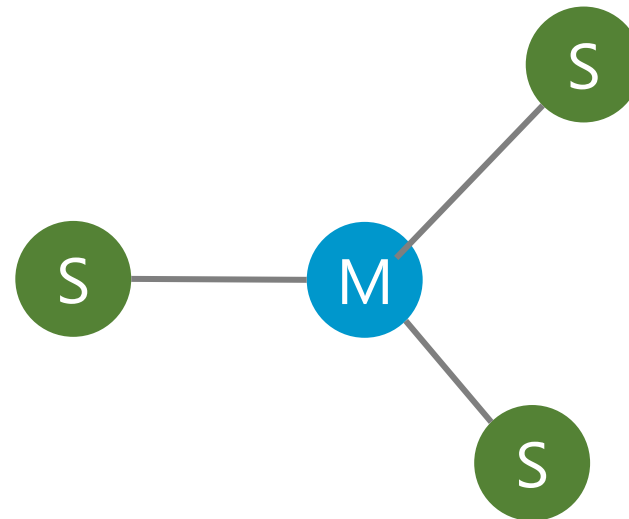
Bluetooth Network Topology (1)

Bluetooth devices are connected to each other to form a “Piconet”

- Piconet has a 3-bit address system – up to 8 devices can participate in a piconet
- A piconet has a master and one or more slaves



A piconet with one master
and one slave

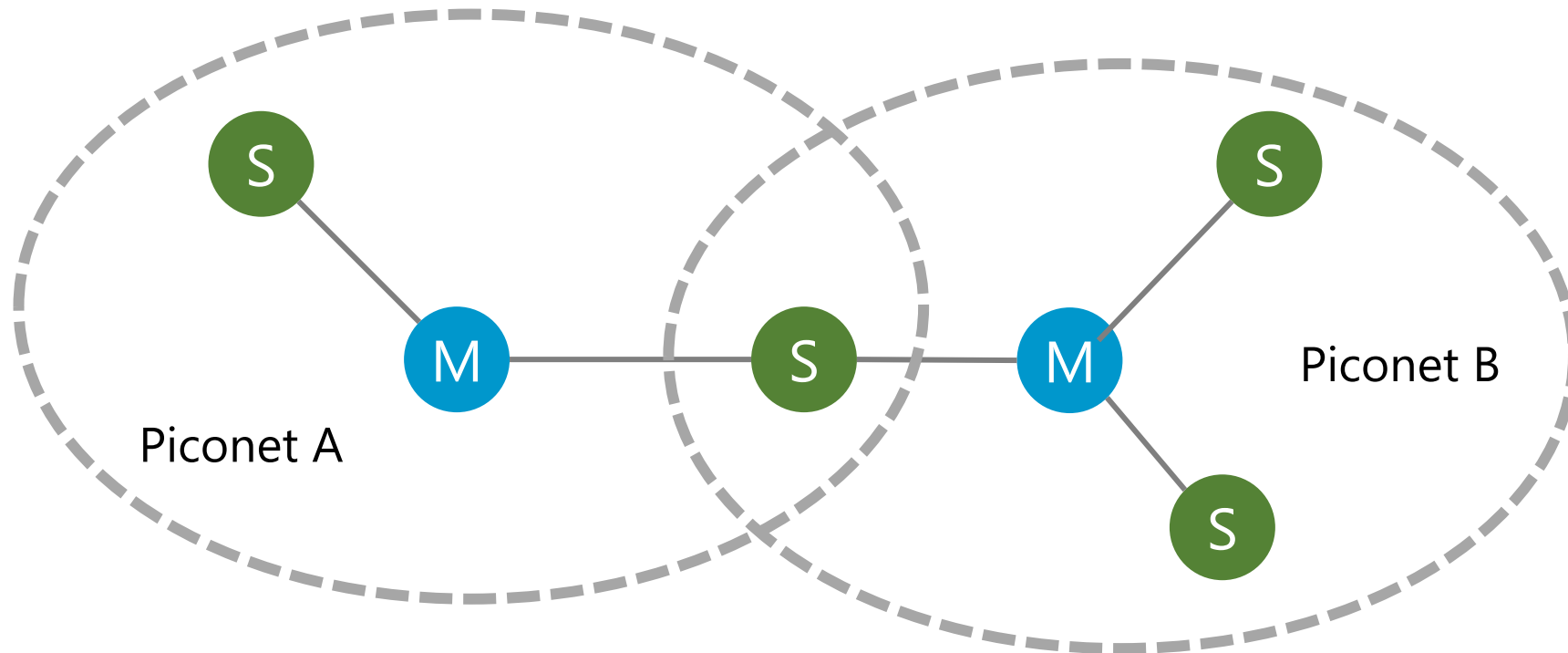


A piconet with one master
and multiple slaves

Bluetooth Network Topology (2)

When more than 8 devices would like to communicate with Bluetooth, a “Scatternet” has to be created

- Scatternet = a number of interconnected piconets
- Member of one piconet participates in another piconet as a **slave**



Bluetooth Profiles

Bluetooth profiles are additional protocols that build upon the basic Bluetooth standard to more clearly define what kind of data a Bluetooth module is transmitting.

The profile(s) a Bluetooth device supports determine(s) what application it's geared towards

- Headset profile (HSP)
- Human interface device profile (HID)
- Advanced Audio Distribution Profile (A2DP)

Note: for two Bluetooth devices to be compatible, they must support the **same profiles**.

References

- Bluetooth Technology
<http://www.bluetooth.com/>
- Bluetooth Special Interest Group (SIG)
<https://www.bluetooth.org/>

Android Bluetooth APIs

Android Bluetooth APIs

To develop Android apps that use Bluetooth for data communication, we can use Android's Bluetooth APIs in the `android.bluetooth` package

First, we ask for permission to use Bluetooth in the AndroidManifest.xml file:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
    ...
    <uses-permission android:name="android.permission.BLUETOOTH" />
    ...
</manifest>
```

Detect Availability of Bluetooth

Before using Bluetooth, make sure that it is **available** in the device running the app, and that it is **enabled**:

```
BluetoothAdapter mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();  
if (mBluetoothAdapter == null) {  
    // This device does not support Bluetooth  
    // You should disable all Bluetooth features in the app  
    ...  
}
```

```
if (!mBluetoothAdapter.isEnabled()) {  
    Intent enableBtIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);  
    startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT);  
}  
...
```

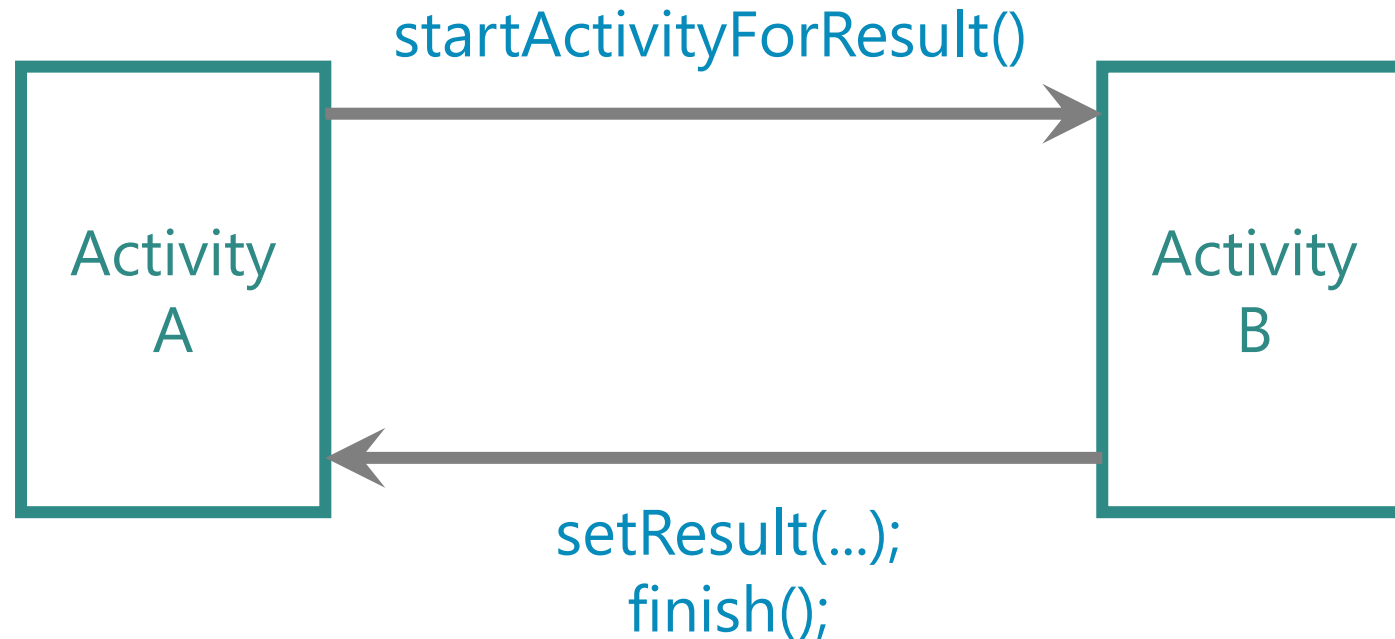
An app wants to turn on Bluetooth.

Deny

Allow

Starting New Activity for Data

- Sometimes you may need to start an activity, in which some processes will be carried out, and then return back to the previous activity with some data
- If so, you should start the new activity using the `startActivityForResult()` method



Starting New Activity for Data - Example

Activity A

```
int code = 10; // self-defined number to identify this action
Intent intent = new Intent(this, ActivityB.class);
Intent.putExtra("data", "...");
...
startActivityForResult(intent, code);
```

Activity B

```
Intent intent = new Intent();
Intent.putExtra("result", "...");
setResult(Activity.RESULT_OK, intent);
finish();
```

Starting New Activity for Data - Example

Getting back data in Activity A returned by Activity B

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if (requestCode == code) {
        // This is from the action identified by the self-defined code
        ...
        Bundle data = data.getExtras();
        String s = data.getString("data1");
        ...
    }
}
```

requestCode

The code you pass to
startActivityForResult
when you start Activity B

resultCode

The code you pass to
setResult before you
finish Activity B

data

The data you embed in
the intent before you
finish Activity B

Establishing a Bluetooth Connection

Establishing a Bluetooth connection is a **multi-step** process:

1. Inquiry

- Once device sends out the inquiry request to try to **discover** other devices nearby
- Other devices listening for such request will respond with its address, name and other information

2. Paging

- Forming a connection between two Bluetooth devices
- Addresses of the two devices are obtained from the inquiry process

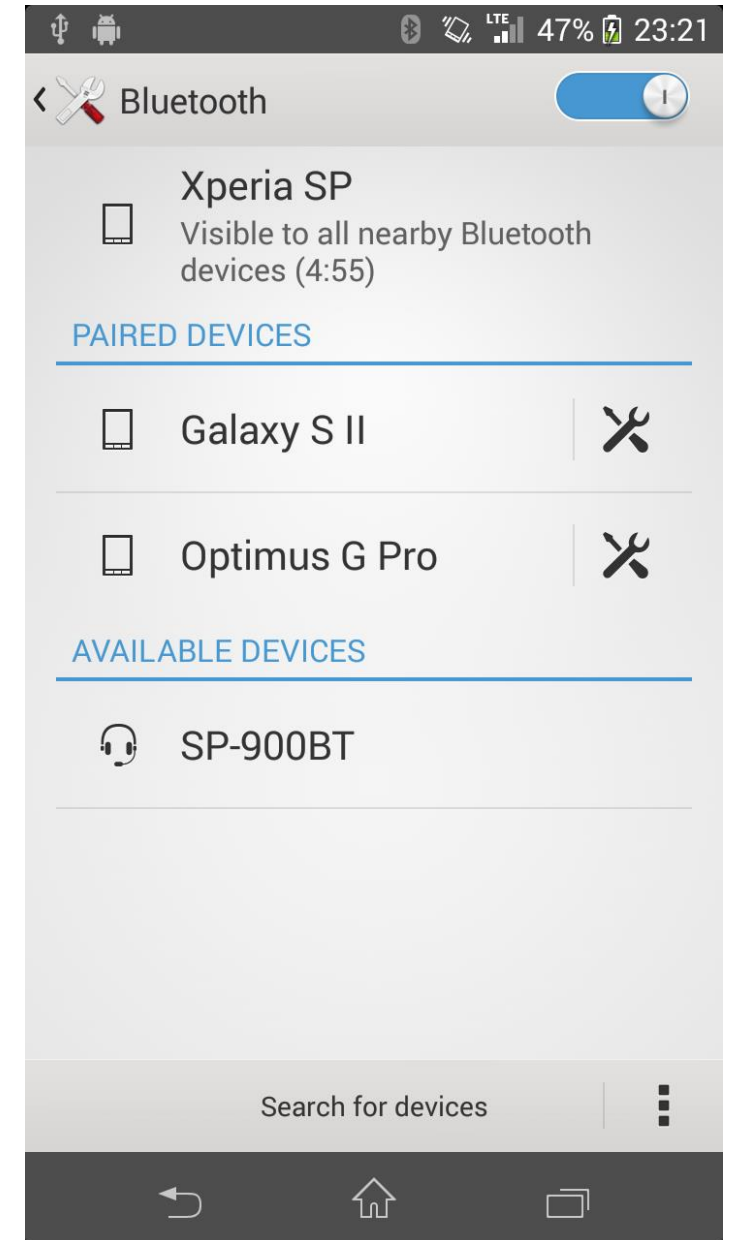
3. Connection

- Connection is established and is ready for data communication

Pairing Between Devices

Two Bluetooth devices can be “paired” or “bonded”, such that they know each other, and may automatically establish a connection when they are close to each other

- In Android, two devices are automatically paired when they try to establish a connection
- Paired devices share a common secret key for establishing a secure channel for connection
- An authentication process is usually required



Retrieve a List of Paired Devices

You can retrieve a list of paired devices with the following codes:

```
Set<BluetoothDevice> pairedDevices = mBluetoothAdapter.getBondedDevices();

// If there are paired devices
if (pairedDevices.size() > 0) {

    // Loop through paired devices
    for (BluetoothDevice device : pairedDevices) {

        // Present the paired devices to the user...
    }

}
```

Discovering Bluetooth Devices Nearby

To discover Bluetooth devices nearby, you need to create a broadcast receiver to listen to the broadcast:

```
// Create a BroadcastReceiver for ACTION_FOUND
private final BroadcastReceiver mReceiver = new BroadcastReceiver() {

    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
        if (BluetoothDevice.ACTION_FOUND.equals(action)) {

            // Get the BluetoothDevice object from the Intent
            BluetoothDevice device =
intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);

            // Perform some actions
            // e.g. Show the user an updated list of devices nearby
            ...

        }
    }
};
```

Other information is also available, check out the Android API Reference for more details

Discovering Bluetooth Devices Nearby

Register the broadcast receiver to listen to `Bluetooth.ACTION_FOUND` intent

```
// Register the BroadcastReceiver
IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION_FOUND);
registerReceiver(mReceiver, filter);

...

// Remember to unregister the receiver when you no longer need it, e.g.:
@Override
public void onPause() {
    unregisterReceiver(mReceiver);
    super.onPause();
}
```


Discovering Bluetooth Devices Nearby

When ready, you can start the discovery process by:

```
...  
mBluetoothAdapter.startDiscovery();  
...
```

Note that device discovery is a heavy process and consumes a lot of resources, you should stop the process as soon as you no longer need to scan for new devices:

```
...  
mBluetoothAdapter.cancelDiscovery();  
...
```

Device Discovery



Device A

Start discovering Bluetooth devices nearby, will discover B and C, but not D



Device B



Device C



Device D

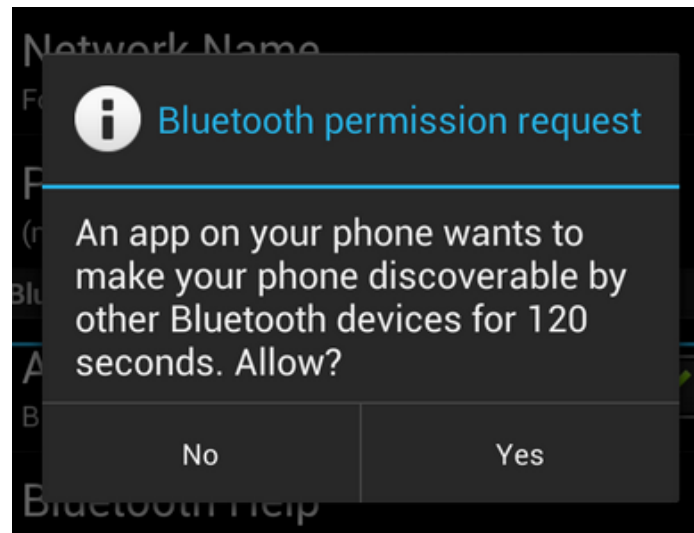
Made discoverable to other devices, will be discovered by A

Has not switched on discoverability, will not be discovered by A

Discoverability of Your Device

By default, Android devices are not set to be discoverable, if you want your device to be discovered by other devices, you need to switch on discoverability:

```
Intent discoverableIntent = new  
Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);  
discoverableIntent.putExtra(BluetoothAdapter.EXTRA_DISCOVERABLE_DURATION, 120);  
startActivity(discoverableIntent);
```



Establishing a Connection

After you have discovered one or more devices, you can try to establish a Bluetooth connection to one of them

Note: a connection between two Bluetooth devices still follows the **client-server model**, you will need to implement both the **server-side** and **client-side** codes

1 Scan for devices

2 Discover Y

3 Connect to X
as a client



X



Y

Start server and
wait for connection

Server-side Mechanism (1)

Two types of sockets will be used to implement a Bluetooth server

1. BluetoothServerSocket

- This is opened to **listen to in-coming Bluetooth connection** (e.g. when a client wants to connect to this device)

2. BluetoothSocket

- This is a socket you will obtain once a connection is **established**
- Use this socket to perform **data communication** between the client and the server

Server-side Mechanism (2)

The server-side app should listen for new connection in a new thread

```
private class AcceptThread extends Thread {  
    private final BluetoothServerSocket mmServerSocket;  
  
    public AcceptThread() {  
        BluetoothServerSocket tmp = null;  
        try {  
            tmp = mBluetoothAdapter.listenUsingRfcommWithServiceRecord(NAME, "A_UNIQUE_ID");  
        } catch (IOException e) { }  
        mmServerSocket = tmp;  
    }  
  
    public void run() {  
        ...  
    }  
    public void cancel() {  
        ...  
    }  
}
```

A name to identify this server program, can be arbitrary

An ID for identifying this Bluetooth service, needs to be known by the client as well

Generating a UUID

UUID stands for universally unique identifier

- A 128 bit value
- Usually represented in hexadecimal text in the form of 8-4-4-4-12 for readability, e.g. **de305d54-75b4-431b-adb2-eb6b9e546013**
- “Unique” means “practically unique”
- Different versions using different methods and seeds to generate the UUID

Generating a UUID

You can generate a UUID in Android using the UUID class

```
...  
String uuid = UUID.randomUUID().toString();  
...
```

For the purpose of generating a UUID for your Bluetooth app:

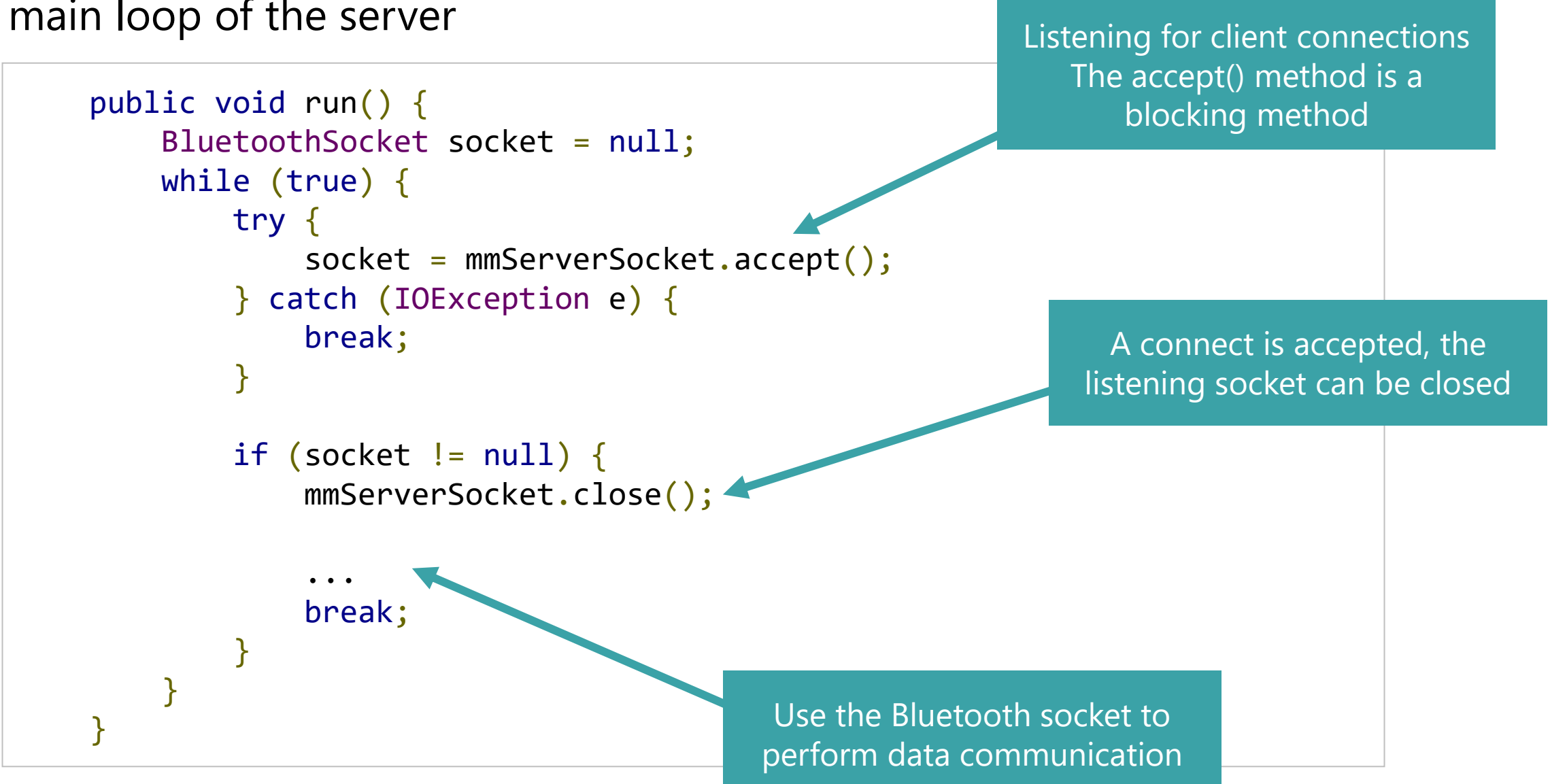
- <https://www.uuidgenerator.net/>

Server-side Mechanism (3)

The main loop of the server

```
public void run() {  
    BluetoothSocket socket = null;  
    while (true) {  
        try {  
            socket = mmServerSocket.accept();  
        } catch (IOException e) {  
            break;  
        }  
  
        if (socket != null) {  
            mmServerSocket.close();  
  
            ...  
            break;  
        }  
    }  
}
```

Listening for client connections
The accept() method is a
blocking method



The diagram illustrates the server's main loop with three annotations. The first annotation, 'Listening for client connections. The accept() method is a blocking method', has an arrow pointing to the 'accept()' call in the try block. The second annotation, 'A connect is accepted, the listening socket can be closed', has an arrow pointing to the 'close()' call. The third annotation, 'Use the Bluetooth socket to perform data communication', has an arrow pointing to the 'break;' statement, indicating the loop ends after accepting a connection.

A connect is accepted, the
listening socket can be closed

Use the Bluetooth socket to
perform data communication

Server-side Mechanism (4)

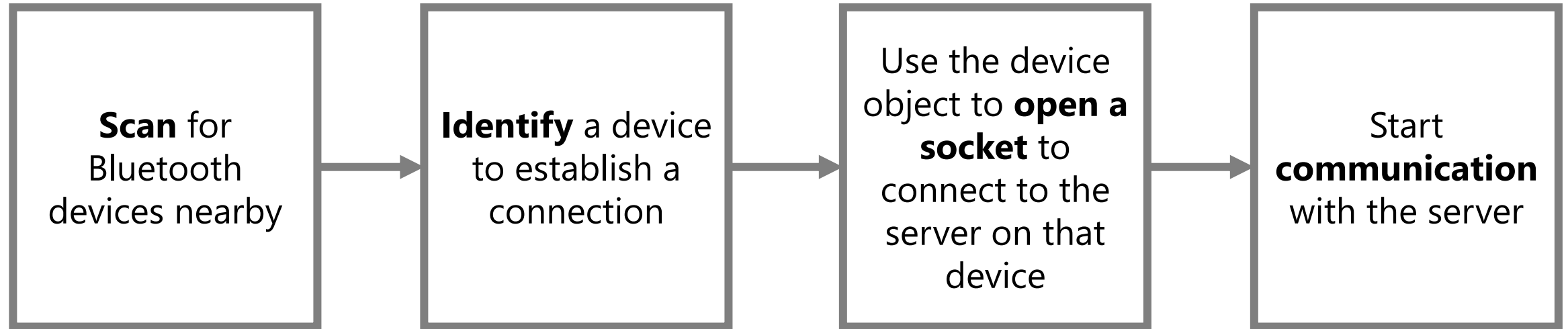
In addition, you should have a method in the thread to **close the server socket**, so that there is a way to stop the server appropriately

```
public void cancel() {  
    try {  
        mmServerSocket.close();  
    } catch (IOException e) {  
    }  
}
```

(Note: closing the server socket will cause an exception in the accept() method, thus the main loop will break and the thread will finish)

Client-side Mechanism (1)

The flow on the client side



Client-side Mechanism (2)


The connection involves some blocking functions: do this in a new thread

```
private class ConnectThread extends Thread {
    private final BluetoothSocket mmSocket;
    private final BluetoothDevice mmDevice;

    public ConnectThread(BluetoothDevice device) {
        BluetoothSocket tmp = null;
        mmDevice = device;

        try {
            tmp = device.createRfcommSocketToServiceRecord("A_UNIQUE_ID");
        } catch (IOException e) {}
        mmSocket = tmp;
    }

    public void run() { ... }
    public void cancel() { ... }
}
```



This ID should match
the one used in the
server code

Client-side Mechanism (3)

The main loop of the client

```
public void run() {  
    mBluetoothAdapter.cancelDiscovery();  
  
    try {  
        mmSocket.connect();  
    } catch (IOException connectException) {  
        try {  
            mmSocket.close();  
        } catch (IOException closeException) { }  
        return;  
    }  
  
    ...  
}
```

Cancel device
discovery because it
consumes a lot of
resources

Try to establish
connection to the
server
(This function is
blocking)

Perform data
communication here
with the server using
the socket

Client-side Mechanism (4)

Provide a method to cancel the connection and close the socket

```
public void cancel() {  
    try {  
        mmSocket.close();  
    } catch (IOException e) { }  
}
```

Sending and Receiving Data

When a connection has established between the client and the server

- Each of them will have a connected `BluetoothSocket` object
- To **send out data**:
 - Get an `OutputStream` object from the socket and use `write()`
- To **receive data**:
 - Get an `InputStream` object from the socket and use `read()`

Sending and Receiving Data (An Example)

```
private class ConnectedThread extends Thread {
    private final BluetoothSocket mmSocket;
    private final InputStream mmInStream;
    private final OutputStream mmOutStream;

    public ConnectedThread(BluetoothSocket socket) {
        mmSocket = socket;
        InputStream tmpIn = null;
        OutputStream tmpOut = null;

        // Get the input and output streams, using temp objects because
        // member streams are final
        try {
            tmpIn = socket.getInputStream();
            tmpOut = socket.getOutputStream();
        } catch (IOException e) { }

        mmInStream = tmpIn;
        mmOutStream = tmpOut;
    }
    ...
}
```


Sending and Receiving Data (An Example)

```
...
public void run() {
    byte[] buffer = new byte[1024]; // buffer store for the stream
    int bytes; // bytes returned from read()

    // Keep listening to the InputStream until an exception occurs
    while (true) {
        try {
            // Read from the InputStream
            bytes = mmInStream.read(buffer);
            // Send the obtained bytes to the UI activity
            mHandler.obtainMessage(MESSAGE_READ, bytes, -1, buffer)
                .sendToTarget();
        } catch (IOException e) {
            break;
        }
    }
}
...
```

Sending and Receiving Data (An Example)

```
...

/* Call this from the main activity to send data to the remote device */
public void write(byte[] bytes) {
    try {
        mmOutputStream.write(bytes);
    } catch (IOException e) { }
}

/* Call this from the main activity to shutdown the connection */
public void cancel() {
    try {
        mmSocket.close();
    } catch (IOException e) { }
}
}
```

References

- Android Bluetooth Guide
<http://developer.android.com/guide/topics/connectivity/bluetooth.html>
- Bluetooth Adapter API Reference
<http://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html>
- Bluetooth Chat Sample App
<http://developer.android.com/samples/BluetoothChat/index.html>

Using Bluetooth for Wireless Sensing

Bluetooth can also be used to discover and communicate sensors, such as in Apple's iBeacon technology

- Bluetooth LE can be used to communicate with devices that operate in a low energy status (e.g. sensors and beacons)
- Distance from sensors can be estimated by the radio signal strength indicator (RSSI)
- Starting from Android 4.3, Bluetooth LE (Low Energy) is supported
<http://developer.android.com/guide/topics/connectivity/bluetooth-le.html>

The Bluetooth Chat App Example

<https://android.googlesource.com/platform/development/+/ec-lair-passion-release/samples/BluetoothChat>

Near Field Communication (NFC)

Near Field Communication

- A set of short-range wireless technologies, typically requiring a distance of 4cm or less to initiate a connection.
- Share data between
 1. An NFC tag and an Android-powered device, or
 2. Between two Android-powered devices
- Data is exchanged using an NFC Forum standard called NFC Data Exchange Format (NDEF)



Near Field Communication

Some history of NFC:

- 2002: Philips and Sony attempted to establish a new RFID technology specification
- 2004: Nokia, Philips and Sony established the NFC Forum
- 2006: Initial specification of NFC Tags released
- 2010: The first Android NFC phone – Samsung Nexus S
- ...

Near Field Communication

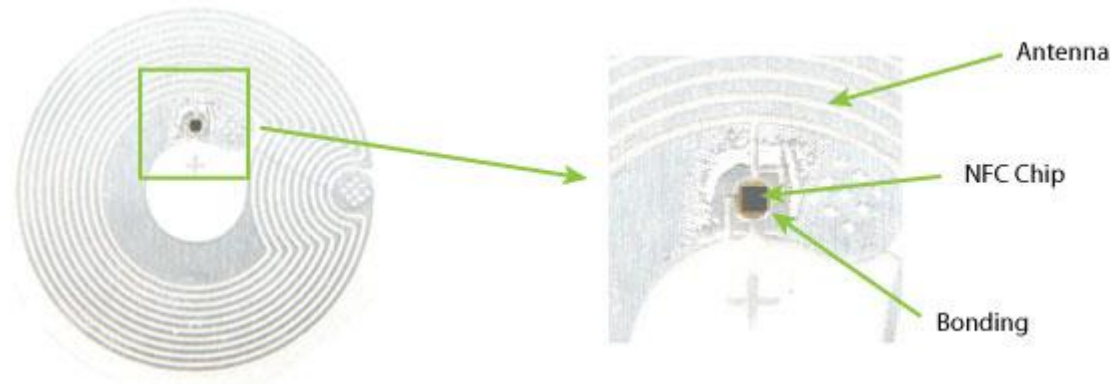
Modes of Operation

- **Reader/Writer Mode:**
Allowing the NFC device to read and/or write passive NFC tags and stickers
- **P2P Mode:**
Allowing the NFC device to exchange data with other NFC peers; this operation mode is used by Android Beam
- **Card Emulation Mode:**
Allowing the NFC device itself to act as an NFC card. The emulated NFC card can then be accessed by an external NFC reader, such as an NFC point-of-sale terminal.

NFC Tags

NFC Tags are **passive devices**

- An active device (e.g. a mobile phone) would generate a magnetic field which induces electric current in the antenna of the tag
- The tag then creates another magnetic field that can be read by the device to enable data transfer



NFC Applications

Common applications of NFC include

- Identification
- Mobile payment
- Pairing devices for Bluetooth or Wi-Fi communication
- Data transfer between devices
- ...



From

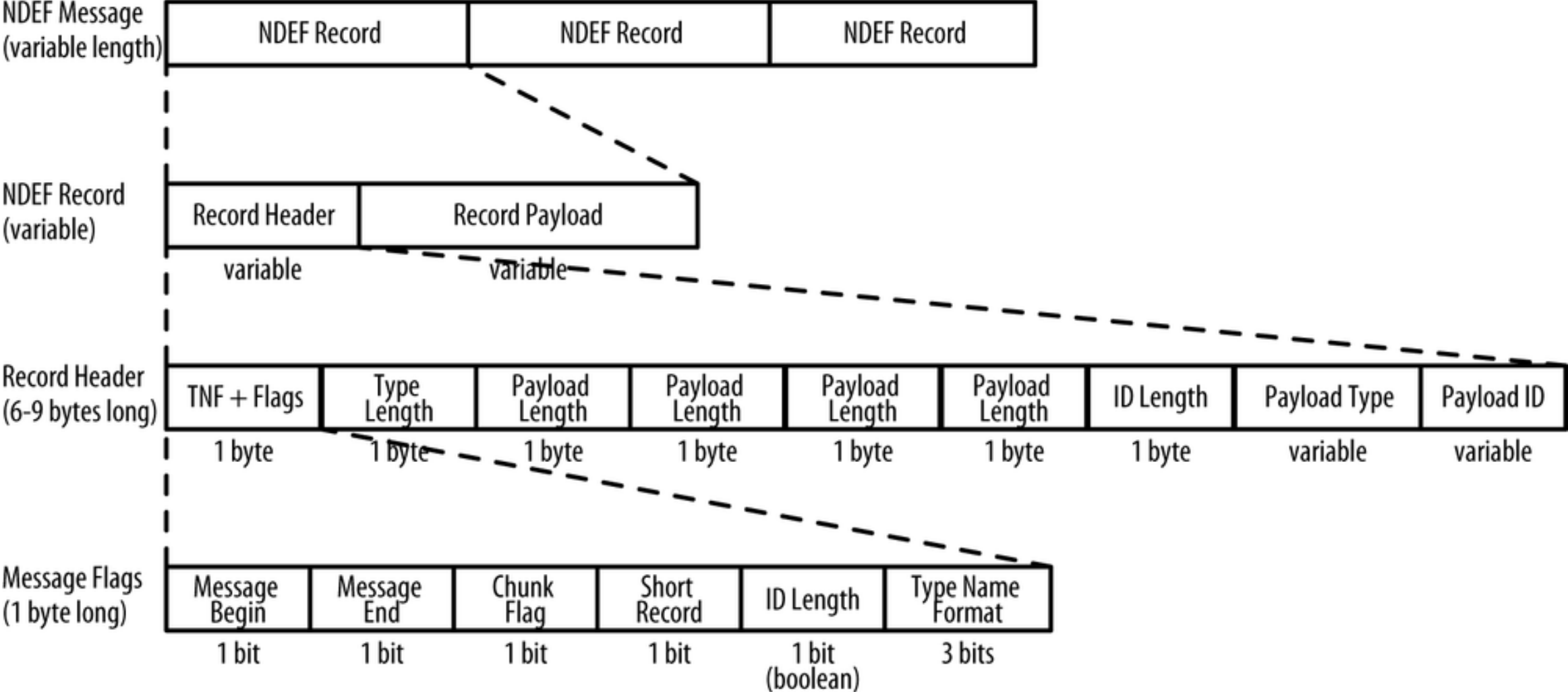
<http://www.techspot.com/guides/385-everything-about-nfc/>

NFC Forum Data Exchange Format

NFC Forum Data Exchange Format (NDEF):

- A lightweight binary message format designed to encapsulate one or more application-defined payloads into a single message construct
- A NDEF message contains one or more **NDEF records**, each carrying a payload of arbitrary type and up to $2^{32}-1$ octets in size
- Records can be chained together to support larger payloads
- An NDEF record carries three parameters for describing its payload: **the payload length**, **the payload type**, and an optional **payload identifier**.

NDEF



Android NFC APIs

NFC on Android

On Android, there are three major use cases of NFC technology

1. [Scanning NFC tags](#), perform read or write operations
2. Sending data from one device to another using [Android Beam](#)
3. Making the device [emulate an NFC tag](#) and interact with an NFC reader

Application Scenarios

When you are developing for an app that uses NFC, mostly like it will be one of the following scenarios

1

Read & Write NFC Tags
(All by yourself)



You design the NFC tags and how data is stored

2

Read & Write NFC Tags
(By others)



Someone else design the tags and tell you the data format

3

Exchange Data between
Two Devices



Both devices have your app installed

Application Scenarios

Android has the most support for NFC tags with data stored according to **NDEF**

- If you have control over the tags, use NDEF if possible
- We will talk about how the following:
 1. How to read/write NDEF tags or to carry out actions
 2. How to exchange data using Android Beam
- For more advanced topics (e.g. handling non-NDEF data), refer to:
<https://developer.android.com/guide/topics/connectivity/nfc/advanced-nfc.html>

Read From an NFC Tag

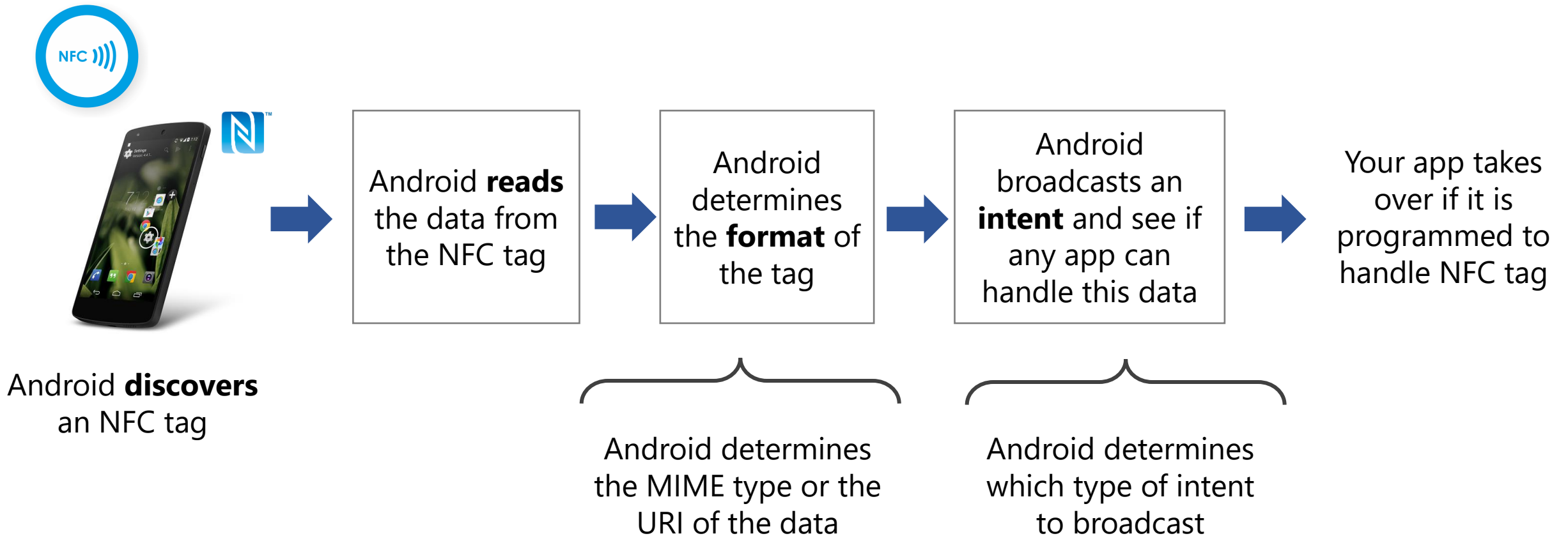
Android devices usually scan for NFC tags when

- 1) NFC is enabled, and
- 2) the screen is unlocked



Android NFC Tag Dispatch System

When Android discovers an NFC tag around, it will read the data from the tag, and then dispatch the data to the apps installed in the device

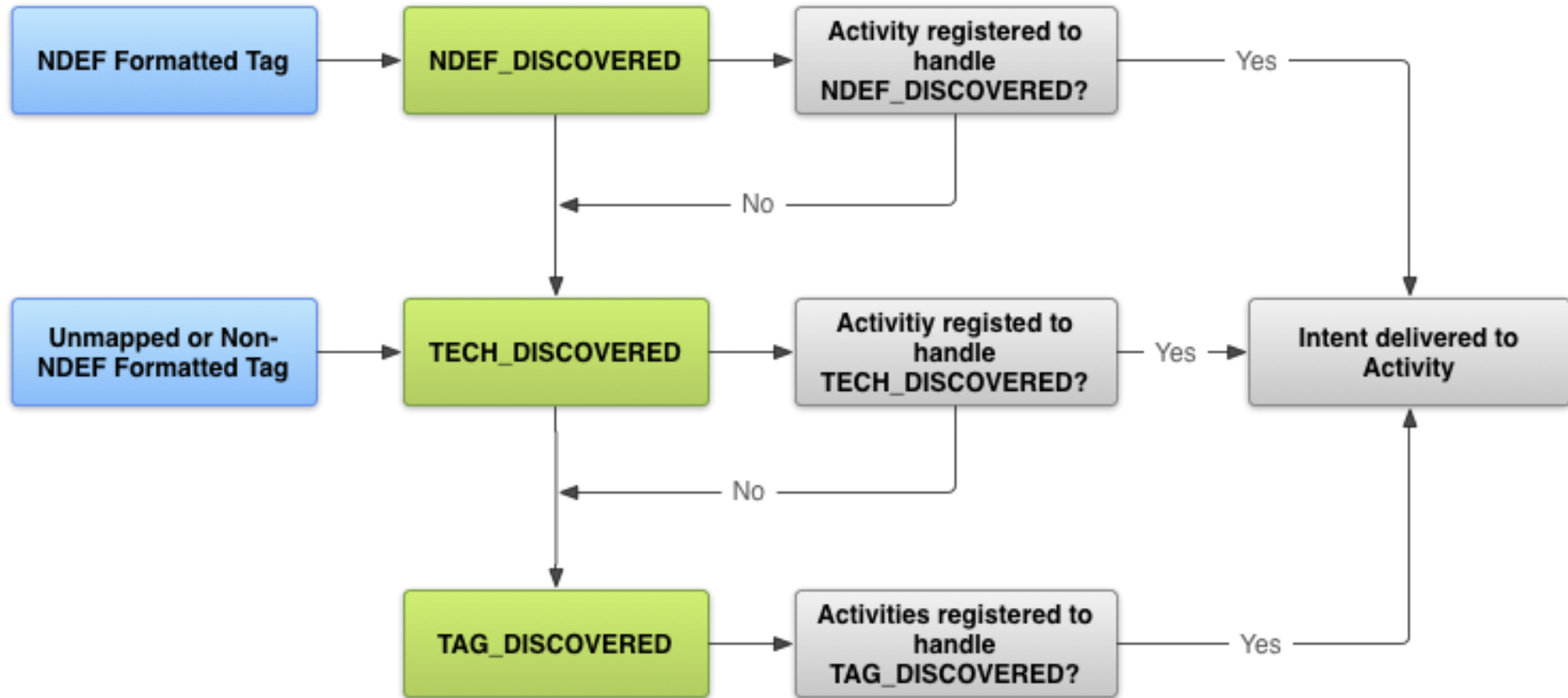


Android NFC Tag Dispatch System

- When Android discovers an NFC tag, it will try to parse it, and determine the format and data type
- The process mainly involves parsing the header of the first record in the NDEF formatted message (see reference)
- Once the data is parsed, Android will create an intent and broadcast it in the system
- Three different intents (in order of priority):
 1. ACTION_NDEF_DISCOVERED
 2. ACTION_TECH_DISCOVERED
 3. ACTION_TAG_DISCOVERED

Android NFC Tag Dispatch System

When Android discovers an NFC tag around, it will read the data from the tag, and then dispatch the data to the apps installed in the device



NFC on Android

To use NFC features in your app, you have to ask for permission from the system

```
<uses-permission android:name="android.permission.NFC" />
```

Also, comprehensive NFC APIs are only available since Android 2.3.3 (and the Android Beam API is only available since Android 4.0):

```
<uses-sdk android:minSdkVersion="10" />
```

And if your app strictly requires NFC to be available on the device, you can write:

```
<uses-feature  
    android:name="android.hardware.nfc"  
    android:required="true" />
```

Intent Filters

- Next, you will have to prepare your application to filter for NFC related **intents**
- You can filter for the **ACTION_NDEF_DISCOVERED**, and specify the data type or URI your app can handle
- For example: the following declaration will allow the activity to filter for **text** data embedded in an NFC tag

```
<activity ...>
  <intent-filter>
    <action android:name="android.nfc.action.NDEF_DISCOVERED"/>
    <category android:name="android.intent.category.DEFAULT"/>
    <data android:mimeType="text/plain" />
  </intent-filter>
</activity>
```

Intent Filters

- Another example: if you would like to handle URLs with a particular format

```
<intent-filter>
  <action android:name="android.nfc.action.NDEF_DISCOVERED"/>
  <category android:name="android.intent.category.DEFAULT"/>
  <data android:scheme="http"
        android:host="yourdomain.com"
        android:pathPrefix="/index.html" />
</intent-filter>
```

In the above example, when the URL "http://yourdomain.com/index.html" is embedded in the NFC tag, your app will be able to pick up this intent

Creating NDEF Messages in Android

Creating NDEF Messages

Before sending data from one device to another device, you need to put your data into a well-formatted NDEF message

- An NDEF message is made up of one or more NDEF records
- The type of the message should be embedded in the FIRST record
- Starting from Android 4.1 (API Level 16), the following methods are available for you to create NDEF records more easily
 - `NdefRecord.createUri()`
 - `NdefRecord.createMime()`

Creating NDEF Messages

Creating a NDEF record that contains a URI or a URL

```
String url = "http://www.mydomain.com/appdata";  
NdefRecord record = NdefRecord.createUri(url);  
NdefMessage message = new NdefMessage(new NdefRecord[] { record });
```



Create a record
containing a URI

Create a NDEF message
containing this record

Creating NDEF Messages

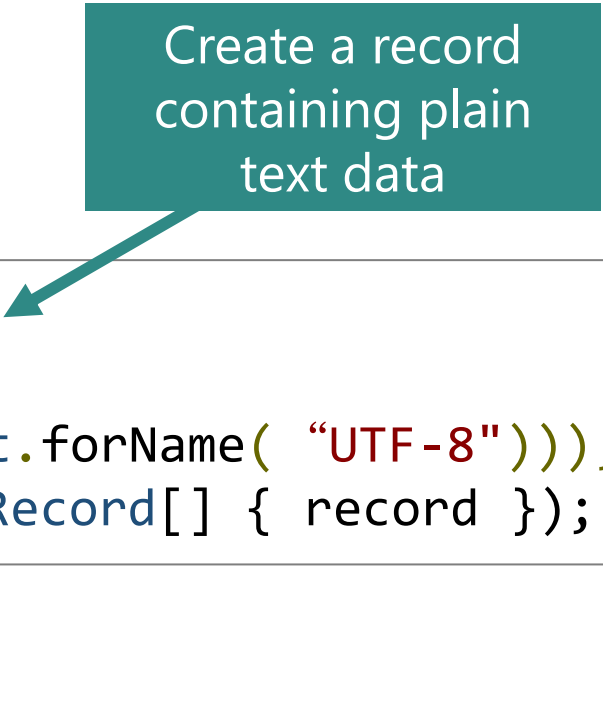
The corresponding intent filter for the above NDEF message

```
<intent-filter>
  <action android:name="android.nfc.action.NDEF_DISCOVERED" />
  <category android:name="android.intent.category.DEFAULT" />
  <data android:scheme="http"
        android:host="www.mydomain.com"
        android:pathPrefix="/appdata" />
</intent-filter>
```

Creating NDEF Messages

Creating a NDEF record containing data of a certain MIME type

Create a record
containing plain
text data



```
NdefRecord mimeRecord = NdefRecord.createMime(  
    "text/plain",  
    "Beam me up, Android".getBytes(Charset.forName( "UTF-8" )));  
NdefMessage message = new NdefMessage(new NdefRecord[] { record });
```

Create a NDEF message
containing this record

Creating NDEF Messages

The corresponding intent filter for the above NDEF message

```
<intent-filter>  
  <action android:name="android.nfc.action.NDEF_DISCOVERED" />  
  <category android:name="android.intent.category.DEFAULT" />  
  <data android:mimeType="text/plain" />  
</intent-filter>
```

The Android Beam API

Android Beam

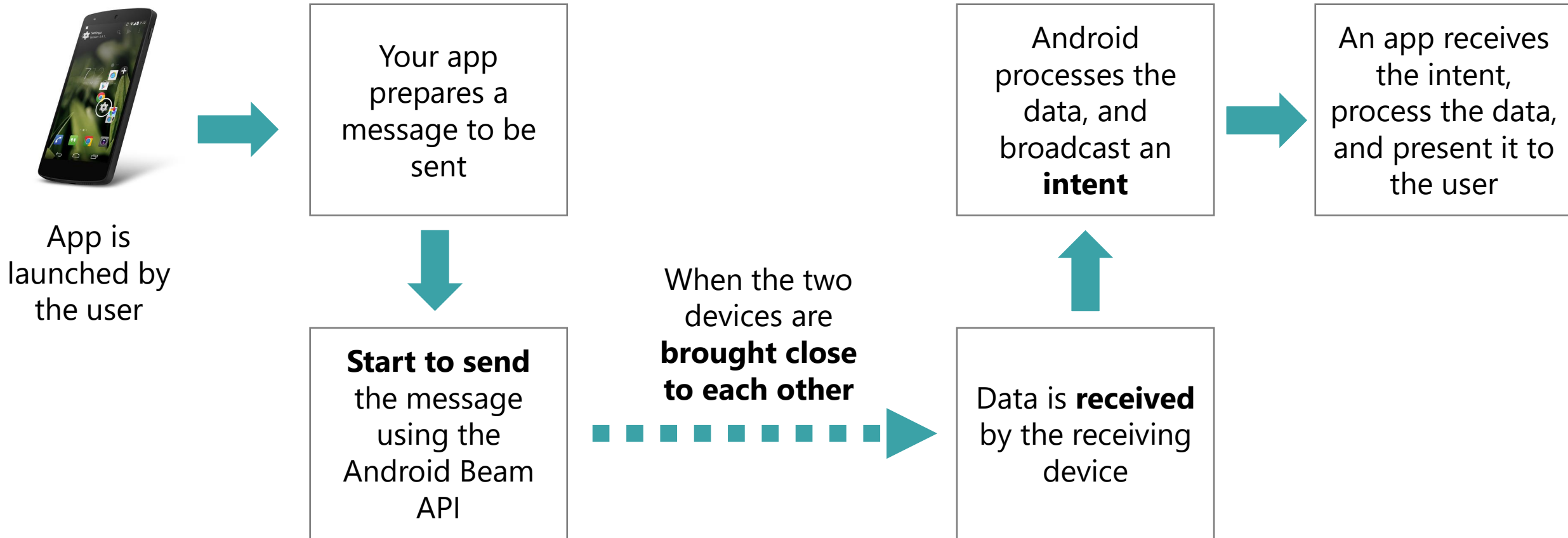
Android Beam is a feature on Android devices that allow simple peer-to-peer data exchange using NFC technology

Conditions for Android Beam to be executed:

- The app that wants to send out data must be in the foreground of the device
- The device accepting the data must NOT be locked

Android Beam

Flow of using Android Beam



Android Beam

Example of pushing an NDEF message

```
public class Beam extends Activity implements CreateNdefMessageCallback {
    NfcAdapter mNfcAdapter;

    @Override
    public void onCreate(Bundle savedInstanceState) {

        mNfcAdapter = NfcAdapter.getDefaultAdapter(this);
        if (mNfcAdapter == null) {
            // NFC not available, skip
            finish();
            return;
        }
        // Register callback
        mNfcAdapter.setNdefPushMessageCallback(this, this);
    }
    ...
}
```

An object implementing the
CreateNdefMessageCallback
interface

The activity itself

Android Beam

Example of pushing an NDEF message (continue)

```
@Override
public NdefMessage createNdefMessage(NfcEvent event) {

    String text = ("Some text to be beamed to another device");

    NdefMessage msg = new NdefMessage(
        new NdefRecord[] {
            createMime(
                "application/vnd.com.example.android.beam",
                text.getBytes()
            )
        }
    );
    return msg;
}
```

Replace this with the package
name of your app
(e.g. vnd.com.iems5722.app)

Android Beam

Receiving the data in the receiving device

Firstly, you should set the intent filter in one of the activities in your app

```
...  
<intent-filter>  
    <action android:name="android.nfc.action.NDEF_DISCOVERED"/>  
    <category android:name="android.intent.category.DEFAULT"/>  
    <data android:mimeType="application/vnd.com.example.android.beam"/>  
</intent-filter>  
...
```

Then, when a beam message is received by Android with the above MIME type, the activity in your application will be started

Android Beam

In the activity, override the following methods to receive the intent

```
@Override
public void onResume() {
    super.onResume();
    // Check to see that the Activity started due to an Android Beam
    if (NfcAdapter.ACTION_NDEF_DISCOVERED.equals(getIntent().getAction())) {
        processIntent(getIntent());
    }
}

@Override
public void onNewIntent(Intent intent) {
    // Set the new intent, onResume will be called after this
    setIntent(intent);
}
```

Android Beam

Process the Intent and extract the data

```
void processIntent(Intent intent) {  
    Parcelable[] rawMsgs = intent.getParcelableArrayExtra(  
        NfcAdapter.EXTRA_NDEF_MESSAGES);  
  
    // only one message sent during the beam  
    NdefMessage msg = (NdefMessage) rawMsgs[0];  
  
    // record 0 contains the MIME type  
    String data = String(msg.getRecords()[0].getPayload());  
  
    ...  
}
```

References

- The NFC Forum
<http://nfc-forum.org/>
- Android NFC APIs
<https://developer.android.com/guide/topics/connectivity/nfc/index.html>
- nearfieldcommunication
<http://www.nearfieldcommunication.com/>
<http://www.nearfieldcommunication.com/nfc-resources/nfc-developer-tools/>

Try NFC

Try NFC

- If you have an NFC-enabled smartphone:
 - Using a NFC app to read from or write to NFC tags, e.g. the Trigger app:
<https://play.google.com/store/apps/details?id=com.jwsoft.nfcactionlauncher>
 - Or the NXP TagWriter app:
<https://play.google.com/store/apps/details?id=com.nxp.nfc.tagwriter>
 - NFC tags can be easily obtained from online stores, e.g.:
<http://www.nfchome.org/>

End of Lecture 8