Android Network Sensor

What are we going to learn?

- Bluetooth Sensor
- Wi-Fi P2P Sensor

Bluetooth Sensor

- All modern mobile devices have Bluetooth
- The following permissions are required to be added to Android manifest file:
  - <uses-permission android: name="android.permission.BLUETOOTH">
  - <uses-permission android: name="android.permission.BLUETOOTH_ADMIN">
- Setting Up Bluetooth
- Finding Devices
- Connecting Devices

Setting up Devices

- Step 1: Get the BluetoothAdapter by calling getDefaultAdapter()

```
BluetoothAdapter bluetoothAdapter;
bluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
```

- Step 2 Enable Bluetooth with ACTION_REQUEST_ENABLE action intent

```
if (bluetoothAdapter.isEnabled()==false){
    Intent intent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
    startActivity(intent);
}
```

If the Bluetooth is not enabled, a dialog will appear requesting user permission to enable Bluetooth

Finding Devices

- Step 1: Start discovering devices, simply call startDiscovery()

```
bluetoothAdapter.startDiscovery();
```

- Step 2: Register a BroadcastReceiver for the ACTION_FOUND Intent to receive information about each device discovered.

```
//Create a BroadcastRecevier for ACTION_FOUND
private final BroadcastReceiver receiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
        //when discovery finds a device
        if (BluetoothDevice.ACTION_FOUND.equals(action)){
            // Get the BluetoothDevice object from the Intent
            BluetoothDevice device = intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
            // Add device to the list
            System.out.println("@ Device name: "+device.getName());
        }
    }
};
```

Finding Devices

```
//register the BroadcastReceiver to broadcast discovered devices
filter = new IntentFilter();
filter.addAction(BluetoothDevice.ACTION_FOUND);
registerReceiver(receiver, filter);
```

- How to get Paired devices

```
//return paired devices
Set<BluetoothDevice> pairedDevices = bluetoothAdapter.getBondedDevices();
if (pairedDevices.size() > 0) {
    // Loop through paired devices
    for (BluetoothDevice device : pairedDevices) {
        // Add the name and address to an array adapter to show in a ListView
        System.out.println("@ paired devices: "+device.getName());
    }
}
```

Connecting Devices

- Connecting as a server

```
private static final UUID MY_UUID = UUID.fromString("00001101-0000-1000-8000-00805F9B34FB");
private class ServerThread extends Thread {
    private final BluetoothServerSocket myServSocket;

    public ServerThread() {
        BluetoothServerSocket tmp = null;

        try {
            tmp = mBluetoothAdapter.listenUsingRfcommWithServiceRecord("myServer", MY_UUID);
        } catch (IOException e) {
            Log.e("Bluetooth", "Server establishing failed");
        }

        myServSocket = tmp;
    }

    connectSocket = serverSocket.accept();
```

Connecting Devices

Connecting as a Client

```
private class ConnectThread extends Thread {
    private final BluetoothSocket mySocket;

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

        try {
            tmp = device.createRfcommSocketToServiceRecord(MY_UUID);
        } catch (IOException e) {
            Log.e("Bluetooth", "Could not connect");
        }
        mySocket = tmp;
    }
 public void run() {
     try {
         mySocket.connect();
```

Wi-Fi Peer-to-Peer connection

Wi-Fi P2P allows Android 4.0 (API level 14) or later devices with the appropriate hardware to connect directly to each other via Wi-Fi without an intermediate access point

- WifiP2pManager class provides methods to interact with the WiFi hardware on your device to discover and connect to peers
  - Initialize(): register the application with the Wi-Fi framework
  - discoverPeers(): initiates peer discovery
  - connect(): starts a peer-to-peer connection
  - cancelConnection(): cancel any ongoing p2p group negotiation
  - requestConnectInfo(): requests a connection information

Wi-Fi Peer-to-Peer connection

Step 1: add the permissions in manifest files

```
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
<uses-permission android:name="android.permission.CHANGE_NETWORK_STATE" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Step 2: Define a WifiP2pManager, Channel

```
WifiP2pManager mManager;
WifiP2pManager.Channel mChannel;
mManager = (WifiP2pManager) getSystemService(Context.WIFI_P2P_SERVICE);
mChannel = mManager.initialize(this, getMainLooper(), null);
```

The returned channel object is used to connect your app to the WiFi P2P framework

Wi-Fi Peer-to-Peer connection

Step 3: Create a broadcast receiver to listen for the changes of the system's WIFI P2P state

```
@Override
public void onReceive(Context context, Intent intent) {
    String action = intent.getAction();

    if (WifiP2pManager.WIFI_P2P_STATE_CHANGED_ACTION.equals(action)) {
        // Check to see if Wi-Fi is enabled and notify appropriate activity
    } else if (WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION.equals(action)) {
        // Call WifiP2pManager.requestPeers() to get a list of current peers
    } else if (WifiP2pManager.WIFI_P2P_CONNECTION_CHANGED_ACTION.equals(action)) {
        // Respond to new connection or disconnections
    } else if (WifiP2pManager.WIFI_P2P_THIS_DEVICE_CHANGED_ACTION.equals(action)) {
        // Respond to this device's wifi state changing
    }
}
```

Wi-Fi Peer-to-Peer connection

Step 4: Register the defined BroadcastRecevier

```
IntentFilter filter;
filter = new IntentFilter();
//Broadcast when Wi-Fi P2P is enabled or disabled on the device.
filter.addAction(WifiP2pManager.WIFI_P2P_STATE_CHANGED_ACTION);
//Broadcast when you call discoverPeers(). You usually want to call requestPeers()
filter.addAction(WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION);
//Broadcast when the state of the device's Wi-Fi connection changes.
filter.addAction(WifiP2pManager.WIFI_P2P_CONNECTION_CHANGED_ACTION);
//Broadcast when a device's details have changed, such as the device's name
filter.addAction(WifiP2pManager.WIFI_P2P_THIS_DEVICE_CHANGED_ACTION);
registerReceiver(mReceiver,filter);
```

Wi-Fi Peer-to-Peer connection

Step 5: call discoverPeers() to discover peer devices in the range

```
public void startScanPeers(){
    mManager.discoverPeers(mChannel, new WifiP2pManager.ActionListener() {
        @Override
        public void onSuccess() { System.out.println("@ successfully"); }

        @Override
        public void onFailure(int i) { System.out.println("@ fail"); }
    });
```

Note: this method only start peer discovery process, onSuccess() get called if the process is successfully started otherwise onFailure() method

Wi-Fi Peer-to-Peer connection

Step 6: retrieve the list of peers by calling requestPeers() method

```
} else if (WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION.equals(action)) {
    // Call WifiP2pManager.requestPeers() to get a list of current peers
    if (mManager != null) {
        mManager.requestPeers(mChannel, new WifiP2pManager.PeerListListener() {
            @Override
            public void onPeersAvailable(WifiP2pDeviceList wifiP2pDeviceList) {
                peers.clear();
                peers.addAll(wifiP2pDeviceList.getDeviceList());
                adapter.notifyDataSetChanged();
                for (int i =0 ; i<peers.size();i++){
                    System.out.println("@ "+peers);
                }
                if (peers.size() == 0) {
                    System.out.println("@ none");
                    return;
                }

            }
        });
    }
    System.out.println("@ WIFI_P2P_PEERS_CHANGED_ACTION");
```

Wi-Fi Peer-to-Peer connection

Step 7: Connect to a peer device by calling connect() method

```
WifiP2pDevice device;
public void connectToPeer(int position){
    device = (WifiP2pDevice) peers.get(position);
    WifiP2pConfig config = new WifiP2pConfig();
    config.deviceAddress = device.deviceAddress;
    mManager.connect(mChannel, config, new WifiP2pManager.ActionListener() {
        @Override
        public void onSuccess() {
            // WiFiDirectBroadcastReceiver will notify us. Ignore for now.

        }
        @Override
        public void onFailure(int reason) {
            Toast.makeText(MainActivity.this, "Connect failed. Retry.",
                    Toast.LENGTH_SHORT).show();
        }
    });
```

Wi-Fi Peer-to-Peer connection

Step 8: request a device's connect information by calling requestConnectionInfo()

```
} 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()) {

        // We are connected with the other device, request connection
        // info to find group owner IP

        mManager.requestConnectionInfo(mChannel, connectionListener);
    }
```

```
mManager.requestConnectionInfo(mChannel, new WifiP2pManager.ConnectionInfoListener() {
```

Wi-Fi Peer-to-Peer connection

Step 9: Get the group owner address and create a thread for connection

```
@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) {
        // Do whatever tasks are specific to the group owner.
        // One common case is creating a server thread and accepting
        // incoming connections.
    } else if (info.groupFormed) {
        // The other device acts as the client. In this case,
        // you'll want to create a client thread that connects to the group
        // owner.
    }
}
```