Creating P2P Connections with Wi-Fi

The Wi-Fi peer-to-peer (P2P) APIs allow applications to connect to nearby devices without needing to connect to a network or hotspot (Android's Wi-Fi P2P framework complies with the Wi-Fi Direct ™ certification program). Wi-Fi P2P allows your application to quickly find and interact with nearby devices, at a range beyond the capabilities of Bluetooth.

This lesson shows you how to find and connect to nearby devices using Wi-Fi P2P.

PREVIOUS | NEXT

THIS LESSON TEACHES YOU HOW TO

1. Set Up Application Permissions
2. Set Up the Broadcast Receiver and Peer-to-Peer Manager
3. Initiate Peer Discovery
4. Fetch the List of Peers
5. Connect to a Peer

YOU SHOULD ALSO READ

- Wi-Fi Peer-to-Peer

Set Up Application Permissions

In order to use Wi-Fi P2P, add the CHANGE_WIFI_STATE, ACCESS_WIFI_STATE, and INTERNET permissions to your manifest. Wi-Fi P2P doesn't require an internet connection, but it does use standard Java sockets, which require the INTERNET permission. So you need the following permissions to use Wi-Fi P2P.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.android.nsdchat"
    ...

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

Set Up a Broadcast Receiver and Peer-to-Peer Manager

To use Wi-Fi P2P, you need to listen for broadcast intents that tell your application when certain events have occurred. In your application, instantiate an IntentFilter and set it to listen for the following:

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.

```
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);

    ...
}
```

At the end of the onCreate() method, get an instance of the WifiP2pManager, and call its initialize() method. This method returns a WifiP2pManager.Channel object, which you'll use later to connect your app to the Wi-Fi P2P framework.

```
@Override

Channel mChannel;

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

Now create a new BroadcastReceiver class that you'll use to listen for changes to the System's Wi-Fi P2P state. In the onReceive() method, add a condition to handle each P2P state change listed above.

```
@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 not, alert
        // the Activity.
        int state = intent.getIntExtra(WifiP2pManager.EXTRA_WIFI_STATE, -1);
        if (state == WifiP2pManager.WIFI_P2P_STATE_ENABLED) {
            activity.setIsWifiP2pEnabled(true);
        } else {
            activity.setIsWifiP2pEnabled(false);
        }
    } else if (WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION.equals(action)) {

        // The peer list has changed!  We should probably do something about
        // that.

    } else if (WifiP2pManager.WIFI_P2P_CONNECTION_CHANGED_ACTION.equals(action)) {

        // Connection state changed!  We should probably do something about
        // that.

    } else if (WifiP2pManager.WIFI_P2P_THIS_DEVICE_CHANGED_ACTION.equals(action)) {
        DeviceListFragment fragment = (DeviceListFragment) activity.getFragmentManager()
                .findFragmentById(R.id.frag_list);
        fragment.updateThisDevice((WifiP2pDevice) intent.getParcelableExtra(
                WifiP2pManager.EXTRA_WIFI_P2P_DEVICE));

    }
}
```

Finally, add code to register the intent filter and broadcast receiver when your main activity is active, and unregister them when the activity is paused. The best place to do this is the onResume() and onPause() methods.

```
/** 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);
}
```

Initiate Peer Discovery

To start searching for nearby devices with Wi-Fi P2P, call discoverPeers(). This method takes the following arguments:

- The WifiP2pManager.Channel you received back when you initialized the peer-to-peer mManager
- An implementation of WifiP2pManager.ActionListener with methods the system invokes for successful and unsuccessful discovery.

```
mManager.discoverPeers(mChannel, new WifiP2pManager.ActionListener() {

        @Override
        public void onSuccess() {
            // Code for when the discovery initiation is successful goes here.
            // No services have actually been discovered yet, so this method
            // can often be left blank.  Code for peer discovery goes in the
            // onReceive method, detailed below.
        }

        @Override
        public void onFailure(int reasonCode) {
            // Code for when the discovery initiation fails goes here.
            // Alert the user that something went wrong.
        }
});
```

Keep in mind that this only *initiates* peer discovery. The discoverPeers() method starts the discovery process and then immediately returns. The system notifies you if the peer discovery process is successfully initiated by calling methods in the provided action listener. Also, discovery will remain active until a connection is initiated or a P2P group is formed.

Fetch the List of Peers

Now write the code that fetches and processes the list of peers. First implement the WifiP2pManager.PeerListListener interface, which provides information about the peers that Wi-Fi P2P has detected. The following code snippet illustrates this.

```
    private List peers = new ArrayList();
    ...

    private PeerListListener peerListListener = new PeerListListener() {
        @Override
        public void onPeersAvailable(WifiP2pDeviceList peerList) {

            // Out with the old, in with the new.
            peers.clear();
            peers.addAll(peerList.getDeviceList());

            // If an AdapterView is backed by this data, notify it
            // of the change.  For instance, if you have a ListView of available
            // peers, trigger an update.
            ((WiFiPeerListAdapter) getListAdapter()).notifyDataSetChanged();
            if (peers.size() == 0) {
                Log.d(WiFiDirectActivity.TAG, "No devices found");
                return;
            }
        }
    }
```

Now modify your broadcast receiver's onReceive() method to call requestPeers() when an intent with the action WIFI_P2P_PEERS_CHANGED_ACTION is received. You need to pass this listener into the receiver somehow. One way is to send it as an argument to the broadcast receiver's constructor.

```
public void onReceive(Context context, Intent intent) {
    ...
    else if (WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION.equals(action)) {

        // Request available peers from the wifi p2p manager. This is an
        // asynchronous call and the calling activity is notified with a
        // callback on PeerListListener.onPeersAvailable()
        if (mManager != null) {
            mManager.requestPeers(mChannel, peerListListener);
        }
        Log.d(WiFiDirectActivity.TAG, "P2P peers changed");
    }...
}
```

Now, an intent with the action WIFI_P2P_PEERS_CHANGED_ACTION intent will trigger a request for an updated peer list.

Connect to a Peer

In order to connect to a peer, create a new WifiP2pConfig object, and copy data into it from the WifiP2pDevice representing the device you want to connect to. Then call the connect() method.

```
@Override
public void connect() {
    // Picking the first device found on the network.
    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() {
            // WiFiDirectBroadcastReceiver will notify us. Ignore for now.
        }

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

The WifiP2pManager.ActionListener implemented in this snippet only notifies you when the *initiation* succeeds or fails. To listen for *changes* in connection state, implement the WifiP2pManager.ConnectionInfoListener interface. Its onConnectionInfoAvailable() callback will notify you when the state of the connection changes. In cases where multiple devices are going to be connected to a single device (like a game with 3 or more players, or a chat app), one device will be designated the "group owner".

```
@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.
    }
}
```

Now go back to the onReceive() method of the broadcast receiver, and modify the section that listens for a WIFI_P2P_CONNECTION_CHANGED_ACTION intent. When this intent is received, call requestConnectionInfo(). This is an asynchronous call, so results will be received by the connection info listener you provide as a parameter.

```
...
} 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);
    }
    ...
```

NEXT : USING WI-FI P2P FOR SERVICE DISCOVERY ›