Wi-Fi Peer-to-Peer

In this document

See also

- Creating P2P Connections with Wi-Fi

Wi-Fi peer-to-peer (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 (Android's Wi-Fi P2P framework complies with the Wi-Fi Alliance's Wi-Fi Direct™ certification program). Using these APIs, you can discover and connect to other devices when each device supports Wi-Fi P2P, then communicate over a speedy connection across distances much longer than a Bluetooth connection. This is useful for applications that share data among users, such as a multiplayer game or a photo sharing application.

The Wi-Fi P2P APIs consist of the following main parts:

- Methods that allow you to discover, request, and connect to peers are defined in the `WifiP2pManager` class.
- Listeners that allow you to be notified of the success or failure of `WifiP2pManager` method calls. When calling `WifiP2pManager` methods, each method can receive a specific listener passed in as a parameter.
- Intents that notify you of specific events detected by the Wi-Fi P2P framework, such as a dropped connection or a newly discovered peer.

You often use these three main components of the APIs together. For example, you can provide a `WifiP2pManager.ActionListener` to a call to `discoverPeers()`, so that you can be notified with the `ActionListener.onSuccess()` and `ActionListener.onFailure()` methods. A `WIFI_P2P_PEERS_CHANGED_ACTION` intent is also broadcast if the `discoverPeers()` method discovers that the peers list has changed.

API Overview

The `WifiP2pManager` class provides methods to allow you to interact with the Wi-Fi hardware on your device to do things like discover and connect to peers. The following actions are available:

Table 1. Wi-Fi P2P Methods

Method	Description
`initialize()`	Registers the application with the Wi-Fi framework. This must be called before calling any other Wi-Fi P2P method.
`connect()`	Starts a peer-to-peer connection with a device with the specified configuration.
`cancelConnect()`	Cancels any ongoing peer-to-peer group negotiation.
`requestConnectInfo()`	Requests a device's connection information.

`createGroup()`	Creates a peer-to-peer group with the current device as the group owner.
`removeGroup()`	Removes the current peer-to-peer group.
`requestGroupInfo()`	Requests peer-to-peer group information.
`discoverPeers()`	Initiates peer discovery
`requestPeers()`	Requests the current list of discovered peers.

`WifiP2pManager` methods let you pass in a listener, so that the Wi-Fi P2P framework can notify your activity of the status of a call. The available listener interfaces and the corresponding `WifiP2pManager` method calls that use the listeners are described in the following table:

Table 2. Wi-Fi P2P Listeners

Listener interface	Associated actions
`WifiP2pManager.ActionListener`	`connect()`, `cancelConnect()`, `createGroup()`, `removeGroup()`, and `discoverPeers()`
`WifiP2pManager.ChannelListener`	`initialize()`
`WifiP2pManager.ConnectionInfoListener`	`requestConnectInfo()`
`WifiP2pManager.GroupInfoListener`	`requestGroupInfo()`
`WifiP2pManager.PeerListListener`	`requestPeers()`

The Wi-Fi P2P APIs define intents that are broadcast when certain Wi-Fi P2P events happen, such as when a new peer is discovered or when a device's Wi-Fi state changes. You can register to receive these intents in your application by creating a broadcast receiver that handles these intents:

Table 3. Wi-Fi P2P Intents

Intent	Description
`WIFI_P2P_CONNECTION_CHANGED_ACTION`	Broadcast when the state of the device's Wi-Fi connection changes.
`WIFI_P2P_PEERS_CHANGED_ACTION`	Broadcast when you call `discoverPeers()`. You usually want to call `requestPeers()` to get an updated list of peers if you handle this intent in your application.
`WIFI_P2P_STATE_CHANGED_ACTION`	Broadcast when Wi-Fi P2P is enabled or disabled on the device.
`WIFI_P2P_THIS_DEVICE_CHANGED_ACTION`	Broadcast when a device's details have changed, such as the device's name.

Creating a Broadcast Receiver for Wi-Fi P2P Intents

A broadcast receiver allows you to receive intents broadcast by the Android system, so that your application can respond to events that you are interested in. The basic steps for creating a broadcast receiver to handle Wi-Fi P2P intents are as follows:

1. Create a class that extends the `BroadcastReceiver` class. For the class' constructor, you most likely want to have parameters for the `WifiP2pManager`, `WifiP2pManager.Channel`, and the activity that this broadcast receiver will be registered in. This allows the broadcast receiver to send updates to the activity as well as have access to the Wi-Fi hardware and a communication channel if needed.
2. In the broadcast receiver, check for the intents that you are interested in `onReceive()`. Carry out any necessary actions depending on the intent that is received. For example, if the broadcast receiver receives a `WIFI_P2P_PEERS_CHANGED_ACTION` intent, you can call the `requestPeers()` method to get a list of the currently discovered peers.

The following code shows you how to create a typical broadcast receiver. The broadcast receiver takes a `WifiP2pManager` object and an activity as arguments and uses these two classes to appropriately carry out the needed actions when the broadcast receiver receives an intent:

```
/**
 * A BroadcastReceiver that notifies of important Wi-Fi p2p events.
 */
public class WiFiDirectBroadcastReceiver extends BroadcastReceiver {

    private WifiP2pManager mManager;
    private Channel mChannel;
    private MyWiFiActivity mActivity;

    public WiFiDirectBroadcastReceiver(WifiP2pManager manager, Channel channel,
            MyWifiActivity activity) {
        super();
        this.mManager = manager;
        this.mChannel = channel;
        this.mActivity = activity;
    }

    @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
        }
    }
}
```

Creating a Wi-Fi P2P Application

Creating a Wi-Fi P2P application involves creating and registering a broadcast receiver for your application, discovering peers, connecting to a peer, and transferring data to a peer. The following sections describe how to do this.

Initial setup

Before using the Wi-Fi P2P APIs, you must ensure that your application can access the hardware and that the device supports the Wi-Fi P2P protocol. If Wi-Fi P2P is supported, you can obtain an instance of `WifiP2pManager`, create and register your broadcast receiver, and begin using the Wi-Fi P2P APIs.

1. Request permission to use the Wi-Fi hardware on the device and also declare your application to have the correct minimum SDK version in the Android manifest:

   ```
   <uses-sdk android:minSdkVersion="14" />
   <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" />
   ```

2. Check to see if Wi-Fi P2P is on and supported. A good place to check this is in your broadcast receiver when it receives the `WIFI_P2P_STATE_CHANGED_ACTION` intent. Notify your activity of the Wi-Fi P2P state and react accordingly:

```
@Override
public void onReceive(Context context, Intent intent) {
    ...
    String action = intent.getAction();
    if (WifiP2pManager.WIFI_P2P_STATE_CHANGED_ACTION.equals(action)) {
        int state = intent.getIntExtra(WifiP2pManager.EXTRA_WIFI_STATE, -1);
        if (state == WifiP2pManager.WIFI_P2P_STATE_ENABLED) {
            // Wifi P2P is enabled
        } else {
            // Wi-Fi P2P is not enabled
        }
    }
    ...
}
```

3. In your activity's `onCreate()` method, obtain an instance of `WifiP2pManager` and register your application with the Wi-Fi P2P framework by calling `initialize()`. This method returns a `WifiP2pManager.Channel`, which is used to connect your application to the Wi-Fi P2P framework. You should also create an instance of your broadcast receiver with the `WifiP2pManager` and `WifiP2pManager.Channel` objects along with a reference to your activity. This allows your broadcast receiver to notify your activity of interesting events and update it accordingly. It also lets you manipulate the device's Wi-Fi state if necessary:

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

4. Create an intent filter and add the same intents that your broadcast receiver checks for:

```
IntentFilter mIntentFilter;
...
@Override
protected void onCreate(Bundle savedInstanceState){
    ...
    mIntentFilter = new IntentFilter();
    mIntentFilter.addAction(WifiP2pManager.WIFI_P2P_STATE_CHANGED_ACTION);
    mIntentFilter.addAction(WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION);
    mIntentFilter.addAction(WifiP2pManager.WIFI_P2P_CONNECTION_CHANGED_ACTION);
    mIntentFilter.addAction(WifiP2pManager.WIFI_P2P_THIS_DEVICE_CHANGED_ACTION);
    ...
}
```

5. Register the broadcast receiver in the `onResume()` method of your activity and unregister it in the `onPause()` method of your activity:

```
/* register the broadcast receiver with the intent values to be matched */
@Override
protected void onResume() {
    super.onResume();
    registerReceiver(mReceiver, mIntentFilter);
}
/* unregister the broadcast receiver */
@Override
protected void onPause() {
    super.onPause();
    unregisterReceiver(mReceiver);
}
```

When you have obtained a `WifiP2pManager.Channel` and set up a broadcast receiver, your application can make Wi-Fi P2P method calls and receive Wi-Fi P2P intents.

You can now implement your application and use the Wi-Fi P2P features by calling the methods in `WifiP2pManager`. The next sections describe how to do common actions such as discovering and connecting to peers.

Discovering peers

To discover peers that are available to connect to, call `discoverPeers()` to detect available peers that are in range. The call to this function is asynchronous and a success or failure is communicated to your application with `onSuccess()` and `onFailure()` if you created a `WifiP2pManager.ActionListener`. The `onSuccess()` method only notifies you that the discovery process succeeded and does not provide any information about the actual peers that it discovered, if any:

```
mManager.discoverPeers(channel, new WifiP2pManager.ActionListener() {
    @Override
    public void onSuccess() {
        ...
    }

    @Override
    public void onFailure(int reasonCode) {
        ...
    }
});
```

If the discovery process succeeds and detects peers, the system broadcasts the `WIFI_P2P_PEERS_CHANGED_ACTION` intent, which you can listen for in a broadcast receiver to obtain a list of peers. When your application receives the `WIFI_P2P_PEERS_CHANGED_ACTION` intent, you can request a list of the discovered peers with `requestPeers()`. The following code shows how to set this up:

```
PeerListListener myPeerListListener;
...
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, myPeerListListener);
    }
}
```

The `requestPeers()` method is also asynchronous and can notify your activity when a list of peers is available with `onPeersAvailable()`, which is defined in the `WifiP2pManager.PeerListListener` interface. The `onPeersAvailable()` method provides you with an `WifiP2pDeviceList`, which you can iterate through to find the peer that you want to connect to.

Connecting to peers

When you have figured out the device that you want to connect to after obtaining a list of possible peers, call the `connect()` method to connect to the device. This method call requires a `WifiP2pConfig` object that contains the information of the device to connect to. You can be notified of a connection success or failure through the `WifiP2pManager.ActionListener`. The following code shows you how to create a connection to a desired device:

```
//obtain a peer from the WifiP2pDeviceList
WifiP2pDevice device;
WifiP2pConfig config = new WifiP2pConfig();
config.deviceAddress = device.deviceAddress;
mManager.connect(mChannel, config, new ActionListener() {

    @Override
    public void onSuccess() {
        //success logic
    }

    @Override
    public void onFailure(int reason) {
        //failure logic
    }
});
```

Transferring data

Once a connection is established, you can transfer data between the devices with sockets. The basic steps of transferring data are as follows:

1. Create a `ServerSocket`. This socket waits for a connection from a client on a specified port and blocks until it happens, so do this in a background thread.
2. Create a client `Socket`. The client uses the IP address and port of the server socket to connect to the server device.
3. Send data from the client to the server. When the client socket successfully connects to the server socket, you can send data from the client to the server with byte streams.
4. The server socket waits for a client connection (with the `accept()` method). This call blocks until a client connects, so call this is another thread. When a connection happens, the server device can receive the data from the client. Carry out any actions with this data, such as saving it to a file or presenting it to the user.

The following example, modified from the Wi-Fi P2P Demo sample, shows you how to create this client-server socket communication and transfer JPEG images from a client to a server with a service. For a complete working example, compile and run the Wi-Fi P2P Demo sample.

```
public static class FileServerAsyncTask extends AsyncTask {

    private Context context;
    private TextView statusText;

    public FileServerAsyncTask(Context context, View statusText) {
        this.context = context;
        this.statusText = (TextView) statusText;
    }

    @Override
    protected String doInBackground(Void... params) {
        try {

            /**
             * Create a server socket and wait for client connections. This
             * call blocks until a connection is accepted from a client
             */
            ServerSocket serverSocket = new ServerSocket(8888);
            Socket client = serverSocket.accept();

            /**
             * If this code is reached, a client has connected and transferred data
             * Save the input stream from the client as a JPEG file
             */
            final File f = new File(Environment.getExternalStorageDirectory() + "/"
                    + context.getPackageName() + "/wifip2pshared-" + System.currentTimeMillis()
                    + ".jpg");

            File dirs = new File(f.getParent());
            if (!dirs.exists())
                dirs.mkdirs();
            f.createNewFile();
            InputStream inputstream = client.getInputStream();
            copyFile(inputstream, new FileOutputStream(f));
            serverSocket.close();
            return f.getAbsolutePath();
        } catch (IOException e) {
            Log.e(WiFiDirectActivity.TAG, e.getMessage());
            return null;
        }
    }

    /**
     * Start activity that can handle the JPEG image
     */
    @Override
    protected void onPostExecute(String result) {
        if (result != null) {
            statusText.setText("File copied - " + result);
            Intent intent = new Intent();
            intent.setAction(android.content.Intent.ACTION_VIEW);
            intent.setDataAndType(Uri.parse("file://" + result), "image/*");
            context.startActivity(intent);
        }
    }
}
```

On the client, connect to the server socket with a client socket and transfer data. This example transfers a JPEG file on the client device's file system.

```
Context context = this.getApplicationContext();
String host;
int port;
int len;
Socket socket = new Socket();
byte buf[]  = new byte[1024];
...
try {
    /**
     * Create a client socket with the host,
     * port, and timeout information.
     */
    socket.bind(null);
    socket.connect((new InetSocketAddress(host, port)), 500);

    /**
     * Create a byte stream from a JPEG file and pipe it to the output stream
     * of the socket. This data will be retrieved by the server device.
     */
    OutputStream outputStream = socket.getOutputStream();
    ContentResolver cr = context.getContentResolver();
    InputStream inputStream = null;
    inputStream = cr.openInputStream(Uri.parse("path/to/picture.jpg"));
    while ((len = inputStream.read(buf)) != -1) {
        outputStream.write(buf, 0, len);
    }
    outputStream.close();
    inputStream.close();
} catch (FileNotFoundException e) {
    //catch logic
} catch (IOException e) {
    //catch logic
}

/**
 * Clean up any open sockets when done
 * transferring or if an exception occurred.
 */
finally {
    if (socket != null) {
        if (socket.isConnected()) {
            try {
                socket.close();
            } catch (IOException e) {
                //catch logic
            }
        }
    }
}
```