# Pervasive Systems

Ioannis Chatzigiannakis

Sapienza University of Rome
Department of Computer, Control, and Management Engineering (DIAG)

Lecture 2:
Wi-Fi Direct





## P2P vs Wi-Fi Direct

- Wi-Fi Peer-to-Peer (P2P): technology, technical specification
- Wi-Fi Direct: marketing and certification





## Introduction

- Wi-Fi direct is new technology defined by the Wi-Fi alliance aimed at enhancing direct device to device communication without requiring a wireless access point.
- Wi-Fi direct builds upon the successful IEEE 802.11 infrastructure mode and lets devices negotiate who will take over the AP-like functionalities.
- Direct device to device connectivity was already possible in original IEEE 802.11 standard but it contains many drawbacks such as lack of power saving or extended QoS capabilities.
- Wi-Fi device to device communication space is 802.11z, also known as Tunneled Direct Link Setup(TDLS), which enables direct device to device communication but requires station to associated with the same AP.





## Technical overview

- In a typical Wi-Fi network, client scans and associate to wireless networks available, which are created and announced by Access Points (AP).
- Wi-Fi Direct is that these roles are specified as dynamic, and hence a Wi-Fi Direct device has to implement both the role of a client and the role of an AP.
- These roles are therefore logical roles that could even be executed simultaneously by the same device, this type of operation is called Concurrent mode.

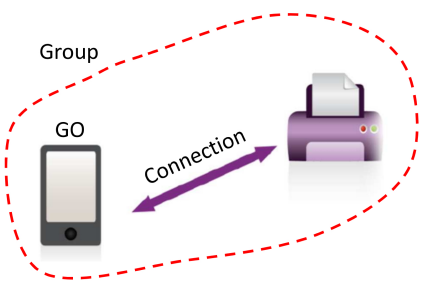
# Architecture

- Wi-Fi direct device communicate by establishing P2P group.
- The device implementing AP-like functionality in P2P group is referred to as the P2P Group Owner(P2P GO), and device acting as client are known as P2P clients.
- Once P2P group is established, other P2P clients can join the group as in a traditional Wi-Fi network.
- When the device act as both as P2P client and as P2P GO the device will typically alternate between the two roles by time-sharing the Wi-Fi interface.(Example: Laptop 2 in upper fig.)
- Like a traditional AP, a P2P GO announces itself through beacons, and has to support power saving for its associated clients.

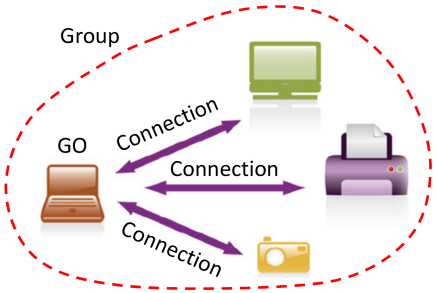




# How Wi-Fi Direct works?




One-to-one configuration



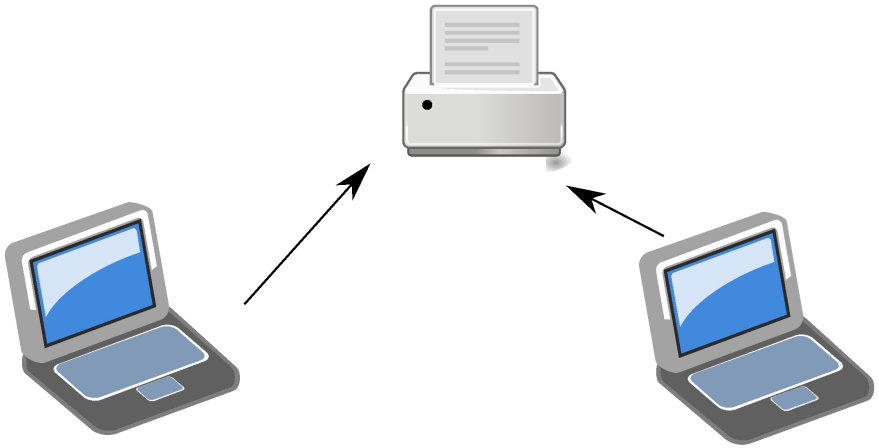

One-to-many configuration

As long as one device in a connection is Wi-Fi Direct-certified, you can connect all devices without a Wi-Fi home network or hotspot.

**GO is short for Group Owner**

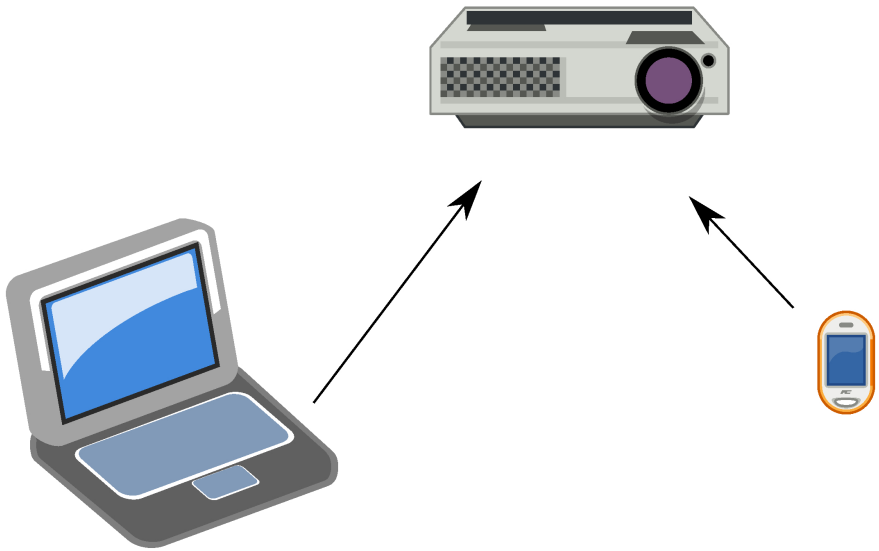


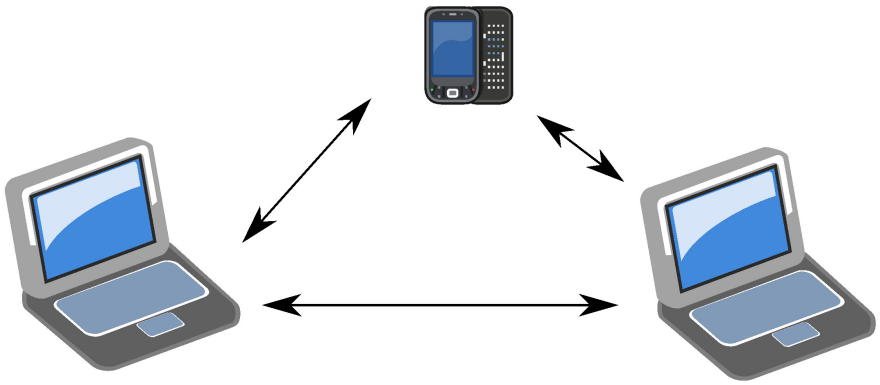






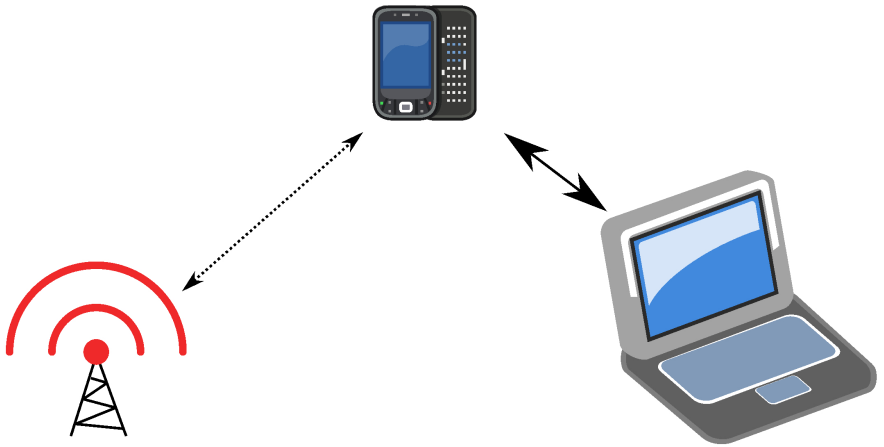
















## Notes

- Only the P2P GO is allowed to cross-connect the devices in its P2P group to an external network.
- This connection must be done at network layer, typically implemented using Network Address Translation(NAT).
- Wi-Fi direct does not allow transferring the role of P2P GO within the group.
- If P2P GO leaves the P2P group then the group is break down, and has to re-established.





## Key Mechanisms

| Key Mechanisms | Madatory | Optional |
|---|---|---|
| Device Discovery | X | |
| – Service Discovery | | X |
| Group Formation | X | |
| – Invitation | | X |
| – Client Discovery | X | |
| Power Management | | |
| – P2P-PS, P2P-WMM-PS | X | |
| – Notice of Absence | X | |
| – Opportunistic Power Save | X | |

## Device Discovery

- Identify other Wi-Fi Direct devices and establish a connection
  - If the target is not in a Group, a new Group is formed. GO is negotiated.
  - If the target is already part of a Group, the searching device may attempt to join the existing Group.
- Wi-Fi Protected Setup (WPS) is used to obtain credentials and authenticate the searching device.





## Service Discovery

- Advertise the higher layer applications to other Wi-Fi Direct devices
  - Even before a connection is formed,
  - For example: Wi-Fi printer can advertise its printer services to other Wi-Fi devices.
- Implementation is vendor-specific.





## Terminology

- P2P Group
- P2P Device
- P2P Group Owner (GO)
- P2P Client
- “legacy” device

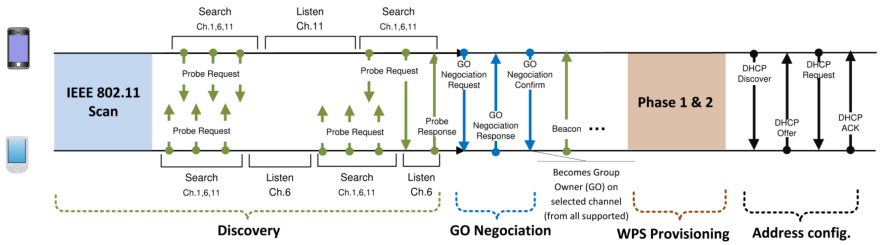




## Group Formation

- Three types of group formation techniques are Standard, Autonomous and Persistent cases.
- Group Formation procedure involves two phases-
  - Determination of P2P Group owner
    - Negotiated - Two P2P devices negotiate for P2P group owner based on desire/capabilities to be a P2P GO.
    - Selected - P2P group Owner role established at formation or at an application level
  - Provisioning of P2P Group
    - Establishment of P2P group session using appropriate credentials
    - Using Wi-Fi simple configuration to exchange credentials.

# Standard Group Formation

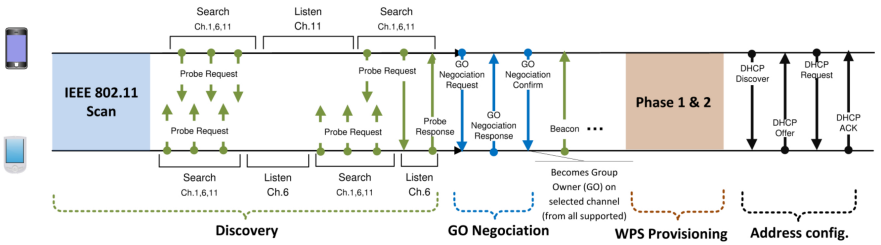



- for speed: only on social channels 1, 6, 11,
- probe request/response mechanism,
- search: device scans (on social channels),
- listen: device listens for probe requests.





# Standard Group Formation

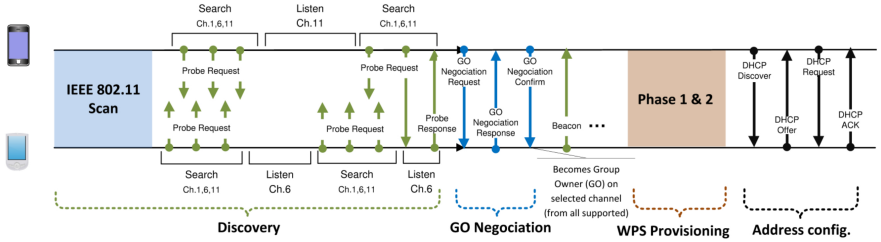



- GO negotiation,
- provisioning (WSC),
- autonomous P2P group.





# Standard Group Formation

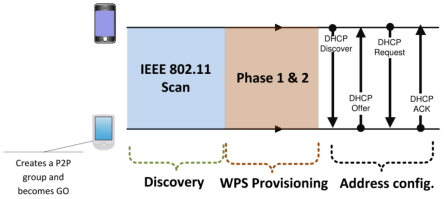



- In this case the P2P device have to discover each other, and then negotiate which device will act as P2P GO.
- Its start by performing a traditional Wi-Fi scan, by means of which they can discover existent groups and Wi-Fi network.
- To prevent conflicts when two devices declare the same GO Intent, a tie-breaker bit is included in the GO Negotiation Request, which is randomly set every time a GO Negotiation Request is sent.



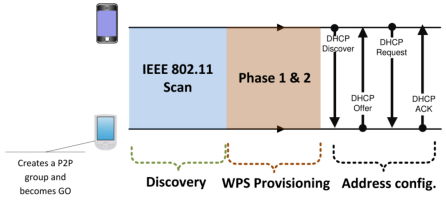


# Autonomous Group Formation




- Discovery,
- provisioning (WSC),
- autonomous P2P group.

# Autonomous Group Formation

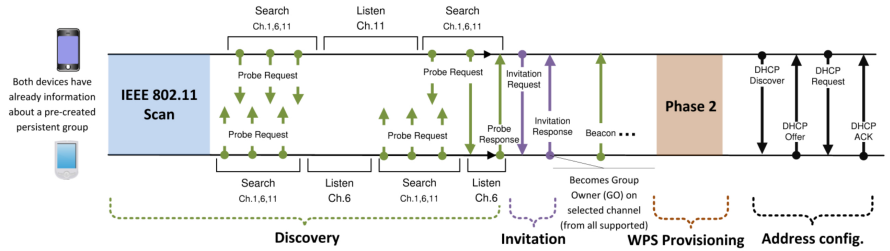



- A P2P device may autonomously create a P2P group, where it immediately becomes the P2P GO, by sitting on a channel and starting a beacon.
- Other devices can discover the established group using traditional scanning mechanisms.
- As compared to previous case, the discovery phase is simplified in this case as the device establishing the group does not alternate between states, and indeed no GO negotiation phase is required.





# Persistent Group Formation




- In this process, P2P device can declare a group as persistent, by using flag in the P2P capabilities attribute present in beacon frames.
- After the discovery phase, if a P2P device recognizes to have formed a persistent group with the corresponding peer in the past, any of the two P2P devices can use the Invitation Procedure to quickly re-instantiate the group.





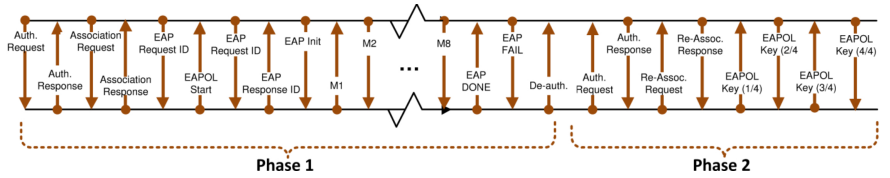
# Security

- Wi-Fi Direct devices are required to implement Wi-Fi Protected Setup (WPS) to support a secure connection with minimal user intervention.
- WPS allows establishing a secure connection by introducing a PIN in the P2P Client, or pushing a button in the two P2P Devices.
- Following WPS terminology, the P2P GO is required to implement an internal Registrar, and the P2P Client is required to implement an Enrollee.
- The operation of WPS is composed of two parts. In the first part, the internal Registrar is in charge of generating and issuing the network credentials, i.e., security keys, to the Enrollee
- In the second part, the Enrollee (P2P Client) disassociates and reconnects using its new authentication credentials.



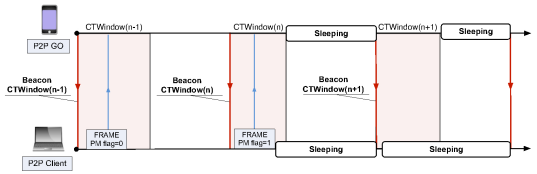


# WPS Provisioning




- 2 phase process.

# Power Saving

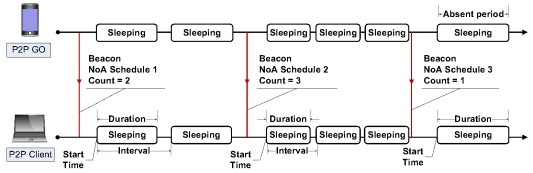



- Wi-Fi Direct defines two new power saving mechanisms: the Opportunistic Power Save protocol and the Notice of Absence (NoA) protocol.
- Opportunistic Power Save protocol (OPS) allows a P2P GO to save power when all its associated clients are sleeping.
- The P2P Group Owner can only save power when all its clients are sleeping.





# Power Saving




- Notice of absence protocol (NoA) allows a P2P GO to announce time intervals, referred to as absence periods, where P2P Clients are not allowed to access the channel.
- P2P GO defines a NoA schedule using four parameters:
  - Duration that specifies the length of each absence period
  - Interval that specifies the time between consecutive absence periods
  - Time that specifies the start time of the first absence period after the current Beacon frame
  - Count that specifies how many absence periods will be scheduled during the current NoA schedule.





# Benefits

- Mobility & Portability: Wi-Fi Direct-certified devices connect anytime, anywhere.
- Immediate Utility: Users have the ability to create direct connections with the very first Wi-Fi Direct-certified device they bring home. For example, a new laptop certified for Wi-Fi Direct can create direct connections with the existing legacy Wi-Fi devices in the users home.
- Ease of Use: Wi-Fi Direct devices have features that allow users to identify available devices and services before establishing a connection.
- Simple Secure Connections: Wi-Fi Protected Setup makes it simple to create security-protected connections between devices. Users in most cases will be able to connect at the push of a button.





# References

1. IEEE 802.11-2013 Standard, Device-To-Device communication with Wi-Fi direct: Overview and experimentation, 2007.
2. Wi-Fi Alliance, P2P Technical Group, Wi-Fi Peer-to-Peer (P2P) Technical Specification v1.0, December 2009.
3. Wi-Fi Alliance, Wi-Fi Protected Setup Specification v1.0h, Dec. 2006.
4. IEEE 802.11z-2010 - Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications Amendment 7: Extensions to Direct-Link Setup (DLS).

# Introduction

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.
- that notify you of specific events detected by the Wi-Fi P2P framework, such as a dropped connection or a newly discovered peer.





# API Overview

| 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

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.





# 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() |

## Wi-Fi P2P Intents

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 (see next slide).





## Wi-Fi P2P Listeners

| 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. |





## Wi-Fi P2P Listeners

| Intent | Description |
|---|---|
| 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.

# Creating a Broadcast Receiver for Wi-Fi P2P Intents

The basic steps are as follows:

- 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.
- 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.





# Creating a Broadcast Receiver for Wi-Fi P2P Intents

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.





```
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.





# Initial setup

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" />
```





# Initial setup

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
        }
    }
    ...
}
```





# Initial setup

- 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.

# Initial setup

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





# Initial setup

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);
    ...
}
```





# Initial setup

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);
}
```





# Initial setup

- 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.





# Discovering peers

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

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





# Discovering peers

- 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.





# Discovering peers

```
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);
    }
}
```

# Discovering peers

- 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.





# Connecting to peers

```
//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 as follows:

- Create a ServerSocket. Waits for a connection from a client on a specified port and blocks until it happens.
- Create a client Socket. The client uses the IP address and port of the server socket to connect to the server device.
- 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.
- 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.