Intro to wifi direct

Anupam Singh
Mobile Engineer : UrbanCompany
Co Host : Androidiots Podcast

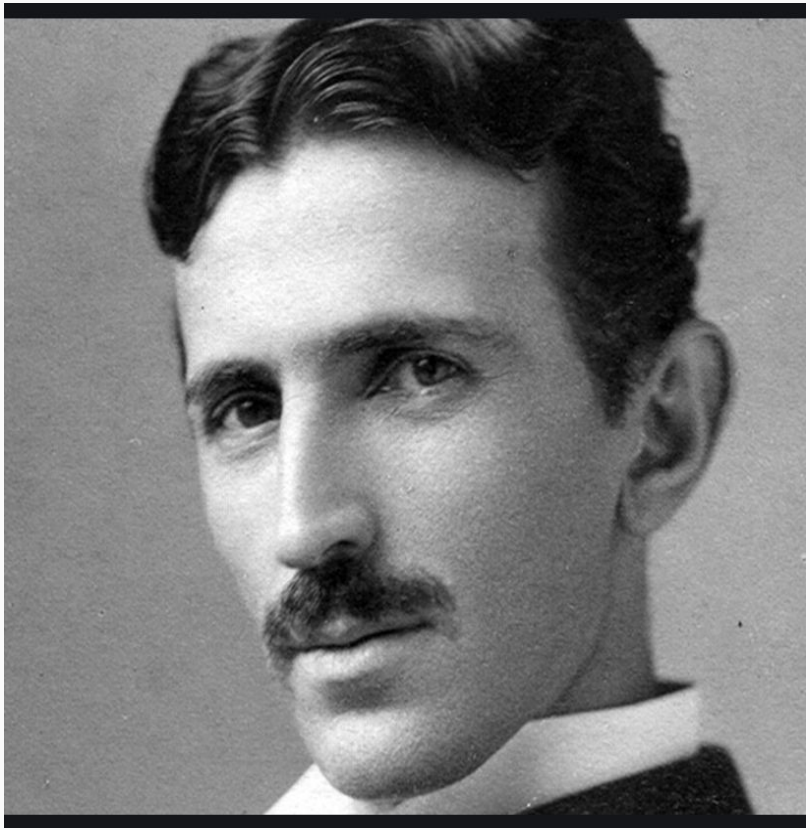
The World currently

- Communication on the basis of data transfer is the bedrock of contemporary world
- Every software system out there requires some sort of data transfer to work
- With the explosion of usage multimedia the data storage was backed by cloud
- They need one thing to work for sure ??

The INTERNET (Img Source : The Internet)

Bijli (The Electricity) : Are you sure?

Thanks : Tesla

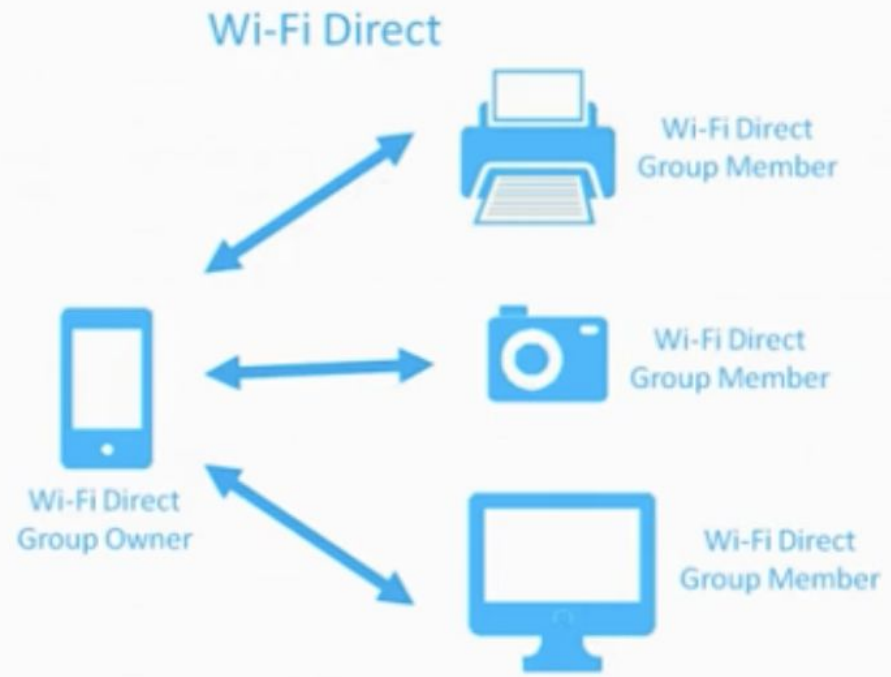


But why are we here?

- Goal of Internet ：Connected world
- What is there is no internet to connect?
- In comes Wifi Direct

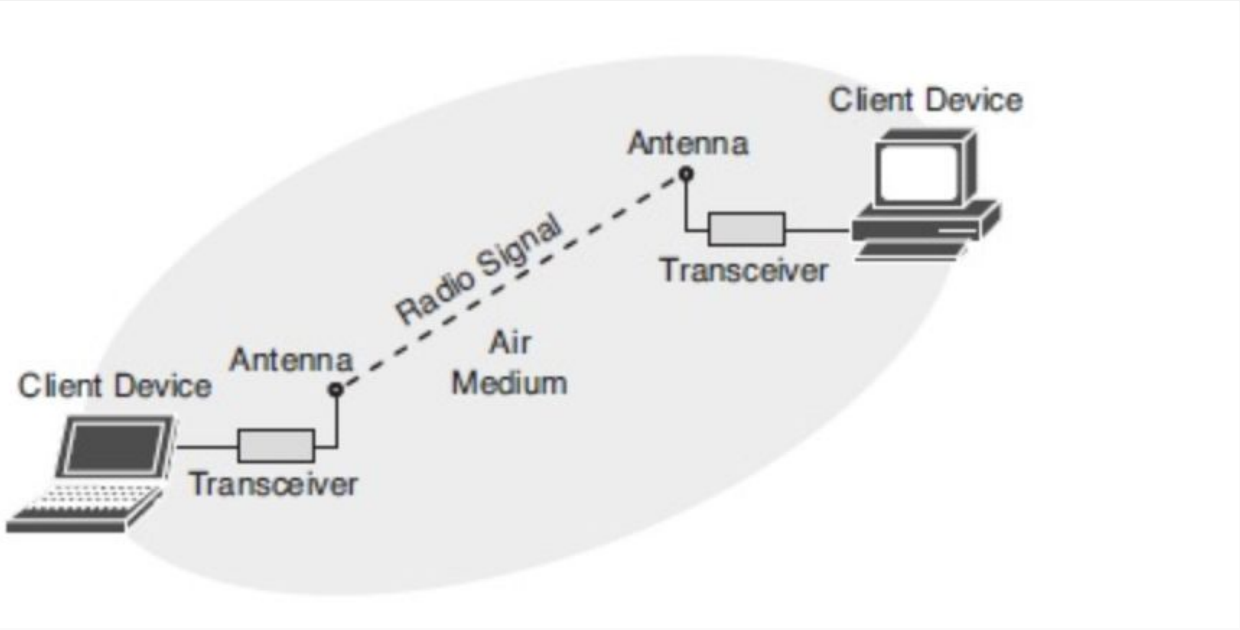
What is Wifi Direct?

- WiFi Direct is a certification of the WiFi Alliance, which has more than 600 members worldwide, with companies such as LG, Microsoft or Dell within it.
- WiFi Direct is seen by many as a kind of the second generation of WiFi. It allows compatible devices that do not have their Internet connection to establish a connection with others that do, the one who has the support and creates the group becomes the group owner, and acts as a router

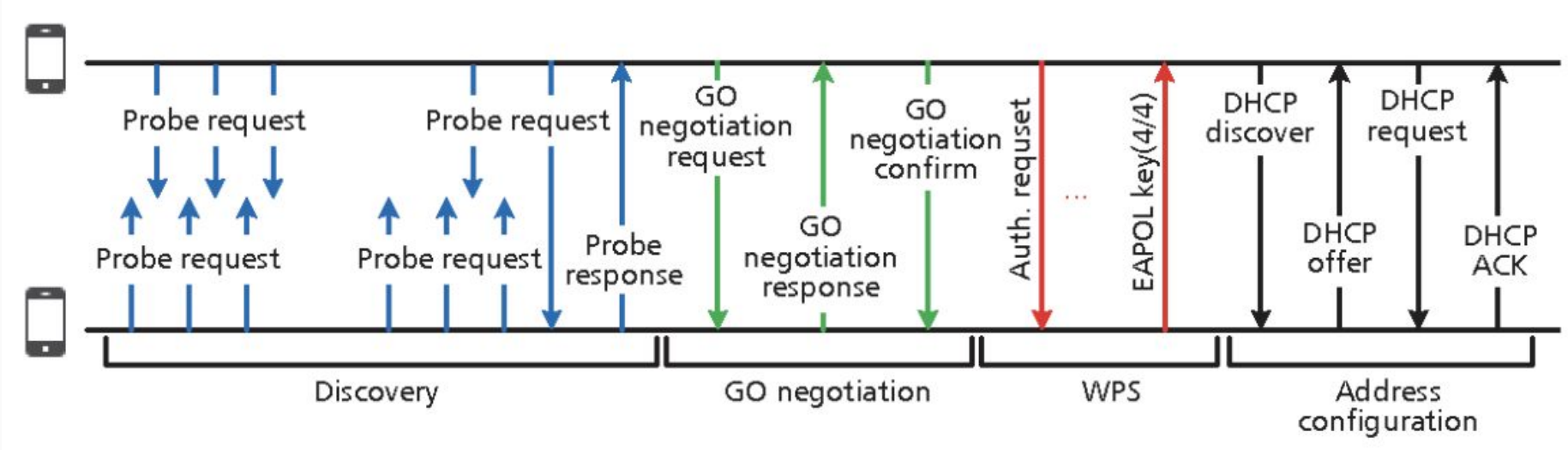


- Range 240-400m, data speed depends of flavor of Wifi (a, g,n)
- Security : WPA2
- Battery consumption for the host is high

How is my data transferred ?



Wifi Direct Communication Chain




It's a Match!
You and Dale have liked each other.

Wifi Direct Api Components

- `WifiP2pManager` : Contains Methods that allow you to discover, request, and connect to peers.

- ***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.

Wifi Direct APP Components

- Creating and registering a broadcast receiver for your application
- Discovering peers
- Connecting to a peer
- Transferring data to a peer.

Part 1 : Creating and registering a broadcast receiver for your application

Step 1: Create BroadCast Receiver

In Broadcast Receiver is used 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.

```
/**
 * A BroadcastReceiver that notifies of important Wi-Fi p2p events.
 */
class WiFiDirectBroadcastReceiver(
        private val manager: WifiP2pManager,
        private val channel: WifiP2pManager.Channel,
        private val activity: MyWifiActivity
) : BroadcastReceiver() {

    override fun onReceive(context: Context, intent: Intent) {
        val action: String = intent.action
        when (action) {
            WifiP2pManager.WIFI_P2P_STATE_CHANGED_ACTION -> {
                // Check to see if Wi-Fi is enabled and notify appropriate activity
            }
            WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION -> {
                // Call WifiP2pManager.requestPeers() to get a list of current peers
            }
            WifiP2pManager.WIFI_P2P_CONNECTION_CHANGED_ACTION -> {
                // Respond to new connection or disconnections
            }
            WifiP2pManager.WIFI_P2P_THIS_DEVICE_CHANGED_ACTION -> {
                // Respond to this device's wifi state changing
            }
        }
    }
}
```

Step 2: Add Permissions

Wi-Fi P2P doesn't require an internet connection, but it does use standard Java sockets, which requires the INTERNET permission

```
<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.ACCESS_FINE_LOCATION" />
<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 3 : Check if Wifi P2P is supported on the device

Remember the BroadCast Listener We created ?
Lets pay him a visit again

```
override fun onReceive(context: Context, intent: Intent) {
...
val action: String = intent.action
when (action) {
    WifiP2pManager.WIFI_P2P_STATE_CHANGED_ACTION -> {
        val state =
intent.getIntExtra(WifiP2pManager.EXTRA_WIFI_STATE, -1)
        when (state) {
            WifiP2pManager.WIFI_P2P_STATE_ENABLED -> {
                // Wifi P2P is enabled
            }
            else -> {
                // Wi-Fi P2P is not enabled
            }
        }
    }
}
}
```

Step 4: Initialize Wifi manager

- Broadcast listener is set up now we need a View/Activity/Fragment to start the communication
- We say create an instance of WifiP2P Manager
- We Create a communication Channel which connects the app to the wifi p2p framework.
- We register our broadcast listener with the channel

```
val manager: WifiP2pManager? by
lazy(LazyThreadSafetyMode.NONE) {
    getSystemService(Context.WIFI_P2P_SERVICE) as
WifiP2pManager?
}

var channel: WifiP2pManager.Channel? = null
var receiver: BroadcastReceiver? = null

override fun onCreate(savedInstanceState: Bundle?) {
    ...

    channel = manager?.initialize(this, mainLooper, null)
    channel?.also { channel ->
        receiver = WiFiDirectBroadcastReceiver(manager,
channel, this)
    }

}
```

Step 5: Create Intent Filter

- ***Intents*** that notify you of specific events detected by the Wi-Fi P2P framework, such as a dropped connection or a newly discovered peer.
- Add the same intents that your broadcast receiver checks for.

```
val intentFilter = IntentFilter().apply {
    addAction(WifiP2pManager.WIFI_P2P_STATE_CHANGED_ACTION)
    addAction(WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION)

addAction(WifiP2pManager.WIFI_P2P_CONNECTION_CHANGED_ACTION)

addAction(WifiP2pManager.WIFI_P2P_THIS_DEVICE_CHANGED_ACTION
)
}
```

Step 6: Register/UnRegister the broadcast receiver

- Pass in the Intent filters created in step 5

```
/* register the broadcast receiver with the intent values to
be matched */
override fun onResume() {
    super.onResume()
   //create a new instance
   receiver = WifiDirectBroadCastReciever(manager,channel,
this);
    receiver?.also { receiver ->
        registerReceiver(receiver, intentFilter)
    }
}

/* unregister the broadcast receiver */
override fun onPause() {
    super.onPause()
    receiver?.also { receiver ->
        unregisterReceiver(receiver)
    }
}
```

Part 2 : Discovering Peers

Step 1 : Use discover peer func

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

```
manager?.discoverPeers(channel, object :
WifiP2pManager.ActionListener {

    override fun onSuccess() {
        ...
    }

    override fun onFailure(reasonCode: Int) {
        ...
    }
})
```

In your Activity/Fragment.

Step 2 : List the peer after discovery

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()`***.

```
override fun onReceive(context: Context, intent: Intent) {
    val action: String = intent.action
    when (action) {
        ...
        WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION -> {
            manager?.requestPeers(channel) { peers:
WifiP2pDeviceList? ->
                // Handle peers list
            }
        }
        ...
    }
}
```

In your BroadCast Listener

Part 3 : Connect to Peers from the LIST

Only step : Use discover peer func

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`.

```
val device: WifiP2pDevice = ...
val config = WifiP2pConfig()
config.deviceAddress = device.deviceAddress
channel?.also { channel ->
    manager?.connect(channel, config, object :
WifiP2pManager.ActionListener {

        override fun onSuccess() {
            //success logic
        }

        override fun onFailure(reason: Int) {
            //failure logic
        }
}
)}

IN your Fragment
```

Part 4 : Transfer Data

Only step : Lot of code

No More Code

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.
5. Remember the Radio Waves

References

- Code : https://github.com/Androidiots/xchange
- Read：https://developer.android.com/training/connect-devices-wirelessly/wifi-direct

No Questions
&
Thanks