Termite WiFi Direct API

Developers Guide

2014/15

Nuno Santos

1. Objectives

This document provides a brief description of the Termite WiFi Direct API.

2. Termite API Guide

In order for an application to make use of the Termite framework, the application must be packaged with the Termite API library and make use of its services. Several steps must be done to integrate and use the Termite API in an application. We now explain these steps, and refer the reader to the SimpleChat application shipped with the Termite package. Most of the examples used in this section were taken from SimpleChat.

Handle Simulated WiFi Direct Events

Throughout the lifecycle of a simulation, the Termite API fires events that can be captured by the application using a broadcast receiver. The Termite API specifies four events:

`WIFI_P2P_STATE_CHANGED_ACTION`

> This event is triggered whenever the Termite service is launched or terminated. Along with the service it is possible to learn the current state of the service. This can be done by obtaining additional state information located in an extra flag `EXTRA_WIFI_STATE`, and testing whether its value corresponds to `WIFI_P2P_STATE_ENABLED`.

`WIFI_P2P_PEERS_CHANGED_ACTION`

> This event is fired whenever there are changes in the set of devices placed within the local device's WiFi range. The function that is invoked passes the current list of devices and it is possible to inspect several details about it using the functions described below.

`WIFI_P2P_NETWORK_MEMBERSHIP_CHANGED_ACTION`

> This event is triggered by membership changes in any of the groups that the current device belongs to. These changes include nodes that join a group or leave a group. The node that triggers the event does not necessarily have to be a peer to the notified device (but it must be reachable to the GO of the group). Note that a device could belong to multiple groups.

`WIFI_P2P_GROUP_OWNERSHIP_CHANGED_ACTION`

> This event happens whenever the ownership state of a group changes: either a node becomes a GO of a group, or a node ceases to be a GO of a group.

The application must take the initiative to specify how to handle all or part of these events using a broadcast receiver. The structure of a broadcast receiver that handles these events looks like this:

```
public class SimWifiP2pBroadcastReceiver extends BroadcastReceiver {
    ...
    @Override
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
        if (SimWifiP2pBroadcast.WIFI_P2P_STATE_CHANGED_ACTION.equals(action)) {
            int  state  =  intent.getIntExtra(SimWifiP2pBroadcast.EXTRA_WIFI_STATE,  -
1);
            if (state == SimWifiP2pBroadcast.WIFI_P2P_STATE_ENABLED) {
                    ...
            } else {
                    ...
            }
        }  else  if  (SimWifiP2pBroadcast.WIFI_P2P_PEERS_CHANGED_ACTION.equals(action))
{
             ...
        } else if (SimWifiP2pBroadcast.WIFI_P2P_NETWORK_MEMBERSHIP_CHANGED_ACTION.
              equals(action)) {
             SimWifiP2pInfo ginfo = (SimWifiP2pInfo) intent.getSerializableExtra(
                          SimWifiP2pBroadcast.EXTRA_GROUP_INFO);
             ...
        } else if (SimWifiP2pBroadcast.WIFI_P2P_GROUP_OWNERSHIP_CHANGED_ACTION.
              equals(action)) {
             SimWifiP2pInfo ginfo = (SimWifiP2pInfo) intent.getSerializableExtra(
                          SimWifiP2pBroadcast.EXTRA_GROUP_INFO);
             ...
        }
    }
}
```

Initialize the Termite API

Before using the specific services of the Termite API, the application must perform a sequence of initialization steps:

1. Update the manifest to launch the Termite service:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    ...
    <application
        ...
        <service
android:name="pt.inesc.termite.wifidirect.service.SimWifiP2pService" />
    </application>
</manifest>
```

2. Initialize the Termite Socket Manager:

```
// initialize the Termite API
SimWifiP2pSocketManager.Init(getApplicationContext());
```

3. Register the events you wish the application be notified of:

```
IntentFilter filter = new IntentFilter();
filter.addAction(SimWifiP2pBroadcast.WIFI_P2P_STATE_CHANGED_ACTION);
filter.addAction(SimWifiP2pBroadcast.WIFI_P2P_PEERS_CHANGED_ACTION);
filter.addAction(SimWifiP2pBroadcast.WIFI_P2P_NETWORK_MEMBERSHIP_CHANGED_ACTION);
filter.addAction(SimWifiP2pBroadcast.WIFI_P2P_GROUP_OWNERSHIP_CHANGED_ACTION);
SimWifiP2pBroadcastReceiver receiver = new SimWifiP2pBroadcastReceiver(this);
registerReceiver(receiver, filter);
```

4. Bind the Termite Service. This is done as follows:

```
Intent intent = new Intent(v.getContext(), SimWifiP2pService.class);
            bindService(intent, mConnection, Context.BIND_AUTO_CREATE);
```

In the binding process, it is necessary to pass two callbacks, which are invoked if the service has been correctly connected, or otherwise. These callbacks are implemented by a ServiceConnection class in methods `onServiceConnected` and `onServiceDisconnected`, as illustrated below:

```
private SimWifiP2pManager mManager = null;
private Channel mChannel = null;
private Messenger mService = null;
...
private ServiceConnection mConnection = new ServiceConnection() {
// callbacks for service binding, passed to bindService()

       @Override
       public void onServiceConnected(ComponentName className, IBinder service) {
              mService = new Messenger(service);
              mManager = new SimWifiP2pManager(mService);
              mChannel   =   mManager.initialize(getApplication(),   getMainLooper(),
null);
              ...
       }

       @Override
       public void onServiceDisconnected(ComponentName arg0) {
              mService = null;
              mManager = null;
              mChannel = null;
              ...
       }
};
```

This code also shows the final steps that must be done in order to make full use of the Termite API, namely the instantiation and initialization of the `SimWifiP2PManager` class, which will provide the interface between the application and the Termite Service.

Probing the Network

Before establishing a TCP connection, it is necessary to know which devices are part of the network and learn their IPs. This and other information about the simulated WiFi network can be retrieved by obtaining instances of classes `SimWifiP2pInfo` and `SimWifiP2pDeviceList`. There are two ways to obtain these classes:

1. Notified by Termite whenever there are changes in the network: In this case, instances of these classes are passed to the listeners of the broadcast receiver (see above).

2. Explicitly requested by the application: In particular, calling the methods `requestPeers` and `requestGroupInfo` of the `SimWifiP2pManager` class. The first method obtains the list of devices that are within the WiFi range of the device where the call is made, irrespective of whether or not the devices belong to a P2P group; the second method obtains various information about the WiFi groups of the calling device and about the devices that are connected to the same groups. Both methods return immediately and the results are propagated to the application using listener classes, which looks like this:

```
mManager.requestPeers(mChannel, (PeerListListener) SimpleChatActivity.this);
...
@Override
public void onPeersAvailable(SimWifiP2pDeviceList peers) {
...
}
```

And this is what the listener looks like for for the second callback:

```
mManager.requestGroupInfo(mChannel, (GroupInfoListener) SimpleChatActivity.this);
...
@Override
public void onGroupInfoAvailable(SimWifiP2pDeviceList devices,
       SimWifiP2pInfo groupInfo) {
       ...
}
```

Callbacks and broadcast receiver methods return their results in two classes: `SimWifiP2pInfo`, and `SimWifiP2pDeviceList`. These are some of the most relevant methods:

`[SimWifiP2pInfo] String getDeviceName()`

> Returns the name of the device, identification that corresponds to the device name contained in the topology file.

`[SimWifiP2pInfo] Set<String> getDevicesInNetwork()`

> Returns the name of the devices that are in the same network of the calling device. It will include the name of the devices registered in all the P2P groups of which the home device is client or group owner of.

`[SimWifiP2pInfo] boolean askIsConnected()`

> Tells whether the local device is connected to a WiFi network, i.e. is group owner of a P2P group, or client of one or multiple P2P groups.

`[SimWifiP2pInfo] boolean askIsGO()`

Tells whether the local device is the group owner of a P2P group.

`[SimWifiP2pInfo] boolean askIsClient()`

Tells whether the local device is the client of a P2P group.

`[SimWifiP2pInfo] boolean askIsConnectionPossible(String deviceName)`

Tells whether it is possible to set up a TCP connection between the local device and a remote device.

This class uses the names of the devices as IDs. To translate this information to address information, which is necessary for setting up TCP connections, we must use the instances of the `SimWifiP2pDeviceList`, which contain device descriptors (class `SimWifiP2pDevice`).

`[SimWifiP2pDeviceList] Collection<SimWifiP2pDevice> getDeviceList()`

Returns the list of devices. This list includes different sets of devices depending on the callback method that returns the instance of this class. The instance returned by the `requestPeers` callback refers only to devices that are within the range of the local host. The instance returned by the `requestGroupInfo` callback includes all the devices of the simulation scenario.

`[SimWifiP2pDeviceList] SimWifiP2pDevice getByName(String deviceName)`

Yields the device descriptor based on the device name or null if the device descriptor was not found in the list.

The device descriptor contains address information that is necessary for connecting the devices using sockets:

`[SimWifiP2pDevice] String getVirtIp()`

Returns the virtual IP of the device referred in the device descriptor.

`[SimWifiP2pDevice] String getVirtPort()`

Returns the virtual port of the device referred in the device descriptor.

The virtual devices will be used for setting up TCP connections.

Setting Up TCP Connections

Once the service is active, it is possible to set up TCP connections. For that, the server and the client parts must use two wrappers of the typical socket interface. These wrappers have been tailored for Termite: `SimWifiP2pSocket`, and `SimWifiP2pSocketServer`. The server side looks like this:

```
private SimWifiP2pSocketServer mSrvSocket = null;
...
try {
      mSrvSocket = new SimWifiP2pSocketServer(10001);
```

```
       SimWifiP2pSocket sock = mSrvSocket.accept();
       ...
       sockIn = new BufferedReader(new InputStreamReader(sock.getInputStream()));
       String s = sockIn.readLine();
       ...
} catch (IOException e) {
       e.printStackTrace();
}
```

The client side looks like this:

```
SimWifiP2pSocket mCliSocket = null;
...
try {
       mCliSocket = new SimWifiP2pSocket("192.168.0.2",10001);
} catch (UnknownHostException e) {
       return "Unknown Host:" + e.getMessage();
} catch (IOException e) {
       return "IO error:" + e.getMessage();
}
...
try {
       mCliSocket.getOutputStream().write("Hello World\n");

} catch (IOException e) {
       Log.d("Error reading socket:", e.getMessage());
}
```

These classes do two things: (i) translate the virtual addresses to real addresses, and (ii) simulate the communication paths expressed in the topology of the spontaneous network. Therefore, the IP address that is fed to the client corresponds to the virtual IP address expressed in the topology script for the targeted destination. Likewise for the virtual port. To reflect the topology changes of the network, whenever the connections between peers are not possible, the connection will fail: the connect method will produce an exception when starting a connection, of a `IOException` will be generated when reading from the socket if the connection is active and is broken down.

Termination of the Termite Service

Disabling the Termite service will prevent any further simulation to take place: no more updates are received from the console, no more broadcast events are triggered, and all active connections will be torn down. To disable the Termite service, unbind the service:

```
unbindService(mConnection);
```

In SimpleChat, this is done in the “WiFi-Off” button. The service can be re-enabled in the future.