App Development for Smart Devices
CS 495/595 - Fall 2011

Lec #17: Networking II

Tamer Nadeem
Dept. of Computer Science
Objective

• Bluetooth

• Network Connectivity and WiFi

• Presentation
  - Mobile Crowdsensing: Current State and Future Challenges
    • Presenter: Sheekha Khetan
  - Virtual Compass: relative positioning to sense mobile social interactions
    • Presenter: Pierre-Olivier
Network Connectivity
Network Connectivity

• Different network options (Wi-Fi, GPRS, 3G) have different characteristics (speed, reliability, cost) in accessing Internet.

• Your applications should be able to know and manage these connections to ensure they run efficiently and responsively.

• Android networking is principally handled via the `ConnectivityManager`.

  • `WifiManager` lets you monitor and control the Wi-Fi connectivity.

• Android broadcasts Intents that describe changes in network connectivity and offers APIs to control network settings.
Connectivity Manager

• Use `ConnectivityManager` service to
  ➤ Monitor the connectivity state
  ➤ Set your preferred network connection
  ➤ Manage connectivity failover.

```java
String service = Context.CONNECTIVITY_SERVICE;
ConnectivityManager connectivity =
    (ConnectivityManager) getSystemService(service);
```

• Need to enable read and write network state access permissions.

```xml
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.CHANGE_NETWORK_STATE"/>
```

More Details on ConnectivityManager:
Background Data Transfer

• User sets preference for background data transfers.
  ➤ Enable/Disable background data transfers: Settings ➪ Accounts & sync settings ➪ Background data setting.
  ➤ This value is enforced at the application level

• Obtain the background data setting through calling `getBackgroundDataSetting` on the `ConnectivityManager`.
  ```java
  boolean backgroundEnabled = connectivity.getBackgroundDataSetting();
  ```

• If the background data setting is disabled
  ➤ Your application should transfer data only when it is active and in the foreground.
  ➤ User explicitly requests that your application not transfer data when it is not visible and in the foreground.
Background Data Transfer

• If your application requires background data transfer, notify users and offer to go to the settings page to enable it.

• When user changes the background data preference, the system sends a broadcast Intent with the `ConnectivityManager.ACTION_BACKGROUND_DATA_SETTING_CHANGED` action.

• Register a new Broadcast Receiver that listens for this Intent

```java
class BroadcastReceiver:
    @Override
    public void onReceive(Context context, Intent intent) {
        // Do something when the background data setting changes.
    }

class MainClass:
    public static void main(String[] args) {
        registerReceiver(new BroadcastReceiver(), new IntentFilter("android.net.conn.CONNECTIVITY_CHANGE"));
    }
```
Monitoring Network Details

- ConnectivityManager provides info on the available connections

- Use `getActiveNetworkInfo` or `getNetworkInfo` methods to return `NetworkInfo` object

  ➤ To find the connection status, network type, and detailed state information of the returned network

```java
// Get the active network information.
NetworkInfo activeNetwork = connectivity.getActiveNetworkInfo();
int networkType = networkInfo.getType();
switch (networkType) {
    case (ConnectivityManager.TYPE_MOBILE) : break;
    case (ConnectivityManager.TYPE_WIFI) : break;
    default: break;
}

// Get the mobile network information.
int network = ConnectivityManager.TYPE_MOBILE;
NetworkInfo mobileNetwork = connectivity.getNetworkInfo(network);
NetworkInfo.State state = mobileNetwork.getState();
NetworkInfo.DetailedState detailedState = mobileNetwork.getDetailedState();
```

Controlling Hardware Radios

- When an application requests an Internet connection, Android attempts to connect using the preferred network.

- Use `getNetworkPreference` and `setNetworkPreference` to find the current, and set the preferred, network.

```java
int networkPreference = connectivity.getNetworkPreference();
connectivity.setNetworkPreference(NetworkPreference.PREFER_WIFI);
```

- Use `setRadio` to control the availability of the network types.

```java
connectivity.setRadio(NetworkType.WIFI, false);
connectivity.setRadio(NetworkType.MOBILE, true);
```
Monitoring Network Connectivity

• To monitor network connectivity create, listen for broadcasted `ConnectivityManager.CONNECTIVITY_ACTION` Intents. Include several extras to additional details on the change.
  ➤ `EXTRA_IS_FAIlOVER` - true if the current connection is the result of a failover from a preferred network.
  ➤ `EXTRA_NO_CONNECTIVITY` - device is not connected
  ➤ `EXTRA_REASON` - string describing why the connection failed.
  ➤ `EXTRA_NETWORK_INFO` - returns NetworkInfo object with details
  ➤ `EXTRA_OTHER_NETWORK_INFO` - returns NetworkInfo object populated with details for the possible failover network connection.
  ➤ `EXTRA_EXTRA_INFO` - contains additional network-specific extra connection details.
WiFi
WiFi Manager

• The **WifiManager** represents the Android Wi-Fi Connectivity Service. Used to:
  ➤ Configure Wi-Fi network connections
  ➤ Manage the current Wi-Fi connection
  ➤ Scan for access points
  ➤ Monitor changes in Wi-Fi connectivity.

• Use **Context.WIFI_SERVICE** constant to access WifiManager

```java
String service = Context.WIFI_SERVICE;
WifiManager wifi = (WifiManager) getSystemService(service);
```

• Permission to access/change the Wi-Fi state

```xml
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE"/>
```

More Details on WifiManager:
MANAGING YOUR WI-FI

• Enable/Disable Wi-Fi hardware using the `setWifiEnabled` method

• Request current Wi-Fi state using the `getWifiState` or `isWifiEnabled` methods

```java
if (!wifi.isWifiEnabled())
    if (wifi.getWifiState() != WifiManager.WIFI_STATE_ENABLING)
        wifi.setWifiEnabled(true);
```

• WifiManager provides low-level access to the Wi-Fi network configurations.
  ➤ Full control over each Wi-Fi configuration setting to completely replace the native Wi-Fi management application if required.
Monitoring Wi-Fi Connectivity

- The Wi-Fi Manager broadcasts one of the following Intents with the change in connectivity status:

  ➤ **WIFI_STATE_CHANGED_ACTION** - moves between enabling, enabled, disabling, disabled, and unknown. Includes two extras **EXTRA_WIFI_STATE** and **EXTRA_PREVIOUS_STATE** for the new and previous states.

  ➤ **SUPPLICANT_CONNECTION_CHANGE_ACTION** - the connection state with the active supplicant (access point) changes. It is fired when a new connection is established or an existing connection is lost, using the **EXTRA_NEW_STATE** Boolean extra, which returns true in the former case.
Monitoring Wi-Fi Connectivity

• Continue the broadcasts Intents:
  
  ➤ **NETWORK_STATE_CHANGED_ACTION** - the Wi-Fi connectivity state changes. Two extras: **EXTRA_NETWORK_INFO** includes a NetworkInfo object that details the current network state, and **EXTRA_BSSID** includes the BSSID of the access point you’re connected to.

  ➤ **RSSI_CHANGED_ACTION** - monitor the current signal strength of the connected Wi-Fi network. Includes extra **EXTRA_NEW_RSSI** that holds the current signal strength.

    o To use this signal strength you should use the **calculateSignalLevel** static method on the Wi-Fi Manager to convert it to an integer value on a scale you specify.
Monitoring Active Connection Details

• Use the `getConnectionInfo` method on the WifiManager to find information on the active connection’s status.

  ➤ The returned `WifiInfo` object includes the SSID, BSSID, Mac address, and IP address of the current access point, as well as the current link speed and signal strength.

```java
WifiInfo info = wifi.getConnectionInfo();
if (info.getBSSID() != null) {
    int strength = WifiManager.calculateSignalLevel(info.getRssi(), 5);
    int speed = info.getLinkSpeed();
    String units = WifiInfo.LINK_SPEED_UNITS;
    String ssid = info.getSSID();
    String cSummary = String.format("Connected to %s at %s%. Strength %s/5", ssid, speed, units, strength);
}
```

More Details on WifiInfo:
Scanning for Hotspots

• Wi-Fi Manager can conduct access point scans using the `startScan` method.
  ➤ An Intent with the `SCAN_RESULTSAVAILABLEACTION` action will be broadcast to asynchronously announce that the scan is complete and results are available.
  ➤ Call `getScanResults` to get those results as a list of `ScanResult` objects.
  ➤ Each Scan Result includes the details retrieved for each access point detected, including link speed, signal strength, SSID, and the authentication techniques supported.

• Following example shows how to initiate a scan for access points that displays a Toast indicating the total number of access points found and the name of the access point with the strongest signal.
Example - Scanning for Hotspots

// Register a broadcast receiver that listens for scan results.
registerReceiver(new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        List<ScanResult> results = wifi.getScanResults();
        ScanResult bestSignal = null;
        for (ScanResult result : results) {
            if (bestSignal == null ||
                WifiManager.compareSignalLevel(bestSignal.level, result.level) < 0)
                bestSignal = result;
        }

        String toastText = String.format("%s networks found. %s is the strongest.",
                                           results.size(), bestSignal.SSID);

        Toast.makeText(getApplicationContext(), toastText, Toast.LENGTH_LONG);
    }
}, new IntentFilter(WifiManager.SCAN_RESULTS_AVAILABLE_ACTION));

// Initiate a scan.
wifi.startScan();
Creating Wi-Fi Network Configurations

• Create and register a configuration to connect to a Wi-Fi network.
• Network configurations are stored as WifiConfiguration objects.
• The following is a non-exhaustive list of some of the public fields available for each Wi-Fi configuration:
  • ➤ BSSID - The BSSID for an access point
  • ➤ SSID - The SSID for a particular network
  • ➤ networkId - unique identifier to identify this network configuration
  • ➤ priority - priority when ordering list of access points to connect to
  • ➤ status - The current status of this network connection, which will be one of the following: WifiConfiguration.Status.ENABLED, WifiConfiguration.Status.DISABLED, or WifiConfiguration.Status.CURRENT
Creating Wi-Fi Network Configurations

- The configuration object also contains the supported authentication techniques, as well as the keys used previously to authenticate with this access point.

- The `addNetwork` method lets you specify a new configuration to add to the current list.

- The `updateNetwork` lets you update a network configuration by passing in a `WifiConfiguration` with a network ID and the values you want to change.

- You can also use `removeNetwork`, passing in a network ID, to remove a configuration.

- To persist any changes made to the network configurations, you must call `saveConfiguration`. 
Managing Wi-Fi Configurations

• Once connected, use WiFi Manager to interrogate the active network connection to get additional details of its configuration and settings.

• Use `getConfiguredNetworks` for current network configurations list
  ➤ The list of `WifiConfiguration` objects returned includes the network ID, SSID, and other details for each configuration.

• To use particular network configuration, use the `enableNetwork` method, passing in the network ID to use and specifying true for the `disableAllOthers` parameter

```java
// Get a list of available configurations
List<WifiConfiguration> configurations = wifi.getConfiguredNetworks();
// Get the network ID for the first one.
if (configurations.size() > 0) {
    int netID = configurations.get(0).networkId;
    // Enable that network.
    boolean disableAllOthers = true;
    wifi.enableNetwork(netID, disableAllOthers);
}
```
Questions?