Final Report: The Mobile DYSWIS Spring 2012

IRT Lab, Columbia University

Project Student
Jin Hyung Park
jp2105@columbia.edu

Mentor
Kyung Hwa Kim
kk2515@columbia.edu

Advisor
Henning Schulzrinne
hgs@cs.columbia.edu

May 10, 2012

Table of Contents

1. Introduction

1.1. Purpose

The main purpose of this project is to implement the mobile version of the DYSWIS application so that we can expand the network diagnostic into the mobile network platform. As a result, we can provide detailed information about the failure even if the local network is completely out of service.

1.2. Project Scope

The project scope is defined as follows:

1. Implement the base environment that allows a user to write the mobile DYSWIS module without modifying the original application

2. Implement Bluetooth server library that provides a communication interface between the mobile DYSWIS and the desktop DYSWIS

3. Implement the Android specific network diagnostic module, which can diagnose the Android C2DM(Cloud to Device Manager)

4. Implement common network diagnostic modules

1.3. Runtime Environment

The mobile DYSWIS is written in Java, HTML5, and JavaScript, and it runs under the Android OS 2.3.3 or the newer version and the PhoneGap[1] 1.5.0 or the newer version. Also, the mobile DYSWIS requires the Bluetooth 2.0 stack since it uses bluetooth communication between the mobile version and the desktop version of DYSWIS.

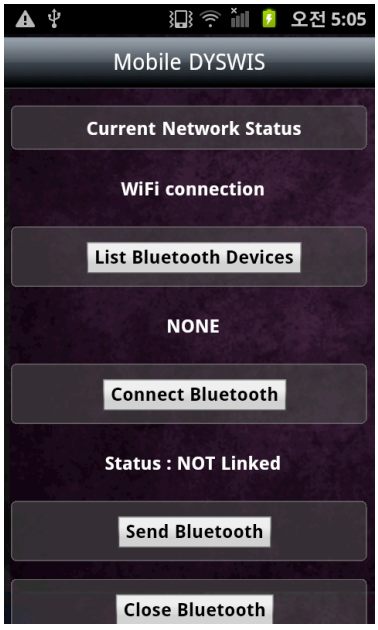


Figure 1.1 The Main Screen of the mobile DYSWIS

2. Application usage and features

2.1. Discover Bluetooth devices and connect to the Bluetooth

After launch, a bluetooth connection between he mobile version and the desktop version of DYSWIS must be established so that the mobile DYSWIS can send information to the desktop DYSWIS and receive information from the desktop DYSWIS to diagnose the network. After launching the desktop DYSWIS, you can discover it by tapping 'List Bluetooth Devices' button.

Figure 2.1 Discovering the desktop DYSWIS

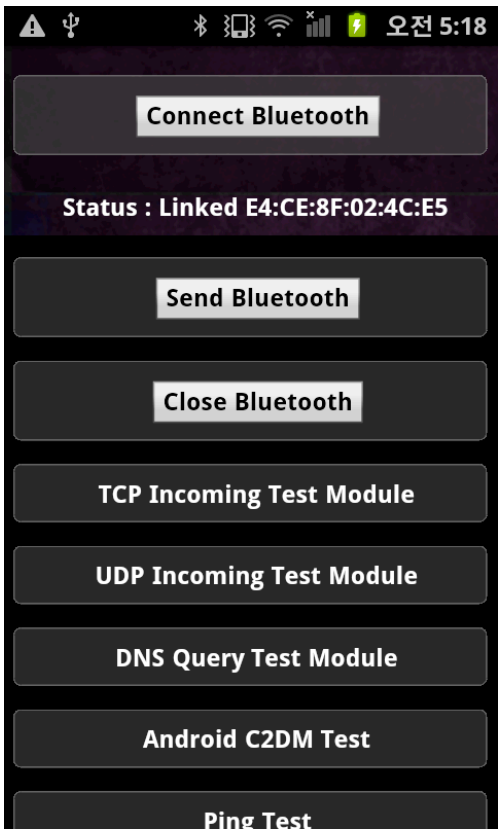
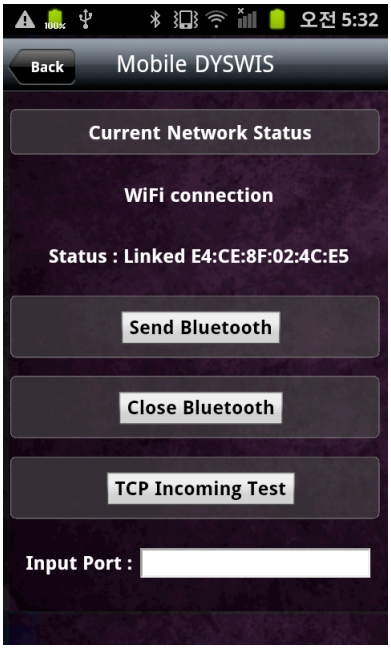


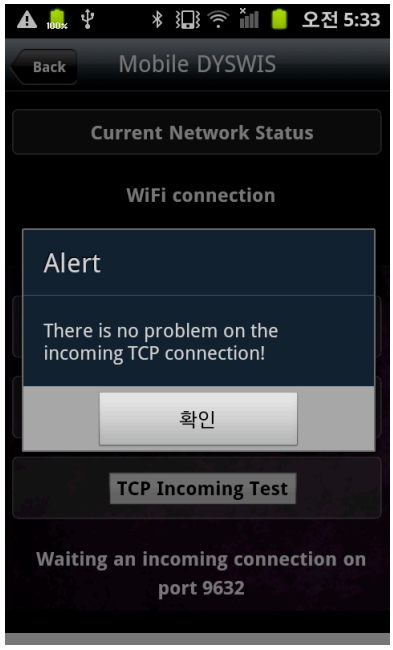
Figure 2.2 The connected result

Figure 2.1 shows the discovery results. By tapping 'Select Device' button, you can set the Bluetooth address of the desktop DYSWIS as the target. Then, tap 'Connect Bluetooth' button to connect to the specified desktop DYSWIS. When the connection is established successfully, the status label will appear as in Figure 2.2.

2.2. Network Diagnostic modules

The mobile DYSWIS has 6 modules. One module tests the android native network function(C2DM). Others are for testing the normal network environments. TCP incoming, UDP incoming, Android C2DM, and Route speed tests are required the desktop DYSWIS, and DNS Query and Ping tests can be run independently.

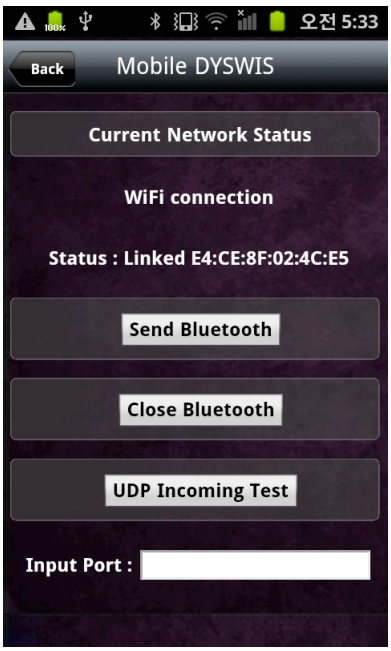


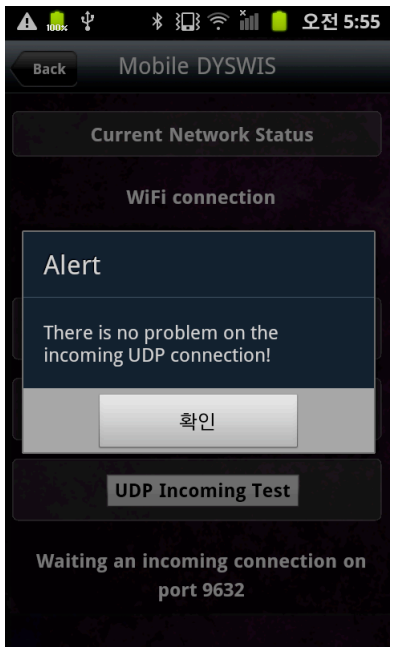


2.2.1. TCP Incoming Test Module

Figure 2.3 - TCP Incoming Test Module

To test TCP incoming connection, a user input the port to open with TCP at first. When the user touch the 'TCP Incoming Test' button, the mobile DYSWIS opens the given TCP port, and send the request to the desktop DYSWIS to connect into the mobile DYSWIS. If the mobile DYSWIS receives the expected data, the test will be passed.

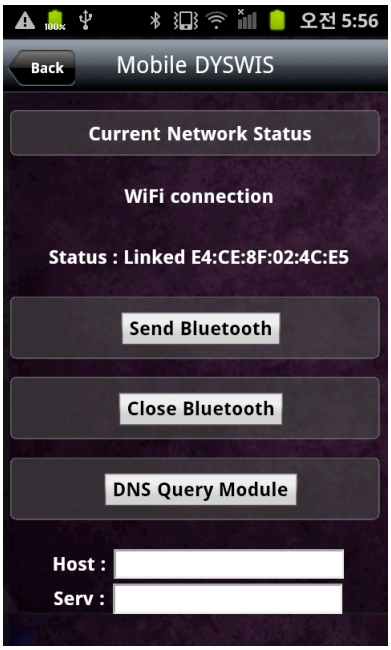


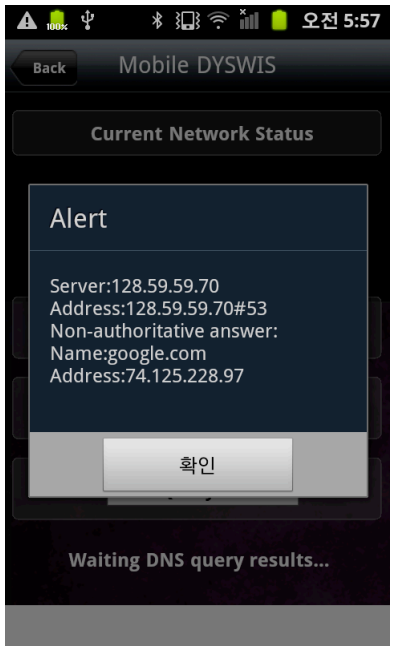


2.2.2. UDP Incoming Test Module

Figure 2.4 - UDP Incoming Test Module

The UDP incoming test is same as the TCP incoming test. The test will be started when the user inputs the port and touches the ‘UDP Incoming Test’ button.

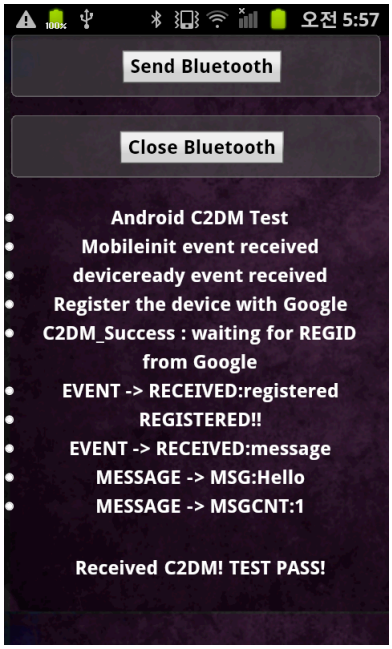




2.2.3. DNS Query Test Module

Figure 2.5 - DNS Query Module

The DNS Query Module can be used to test whether the mobile device can connect the DNS server, and get the correct DNS information. At the ‘Host’ filed, a user should input the host name to verify, and the user should input the DNS server IP address at the ‘Serv’ field. The DNS Query result will be showed at the alert pop-up window.



2.2.4. Android C2DM Test

Figure 2.6 - The Android C2DM Test Module

The Android C2DM test module can test whether the mobile device can receive the push notification correctly or not. This test is needed because lots of Android applications are using this C2DM service, but sometimes users cannot receive the push notification information of each application. As a result, users have questions why their mobile devices cannot receive their push notification information. This module tests Google's C2DM service, and reports the result whether the mobile device can receive the push notification correctly. If the test is passed, it shows the success message in the window.

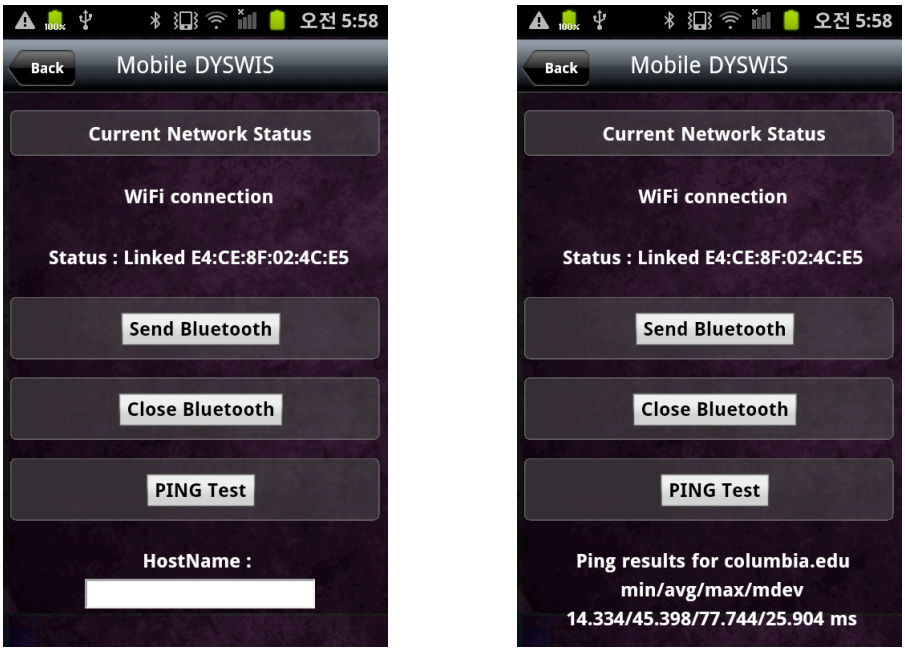


Figure 2.7 - The Ping Test Module

2.2.5. Ping Test

The Ping module implements a normal ICMP ping application. When a user inputs a host name to check the ICMP ping at the 'HostName' field, it will show the ICMP ping results with the minimum speed, average speed, and the maximum speed. The user can easily determine the network speed of the mobile device with this module. Also, this module can be used to test the network connectivity.

2.2.6. Route Test

The Route test module checks the TCP packets of the round trip time to the given hosts. When a user starts the Route test module, the mobile DYSWIS sends the request to the desktop DYSWIS to receive a random host in the desktop DYSWIS P2P network. The mobile DYSWIS sends the requests five times to the desktop DYSWIS. Whenever the mobile DYSWIS receives the server address to check the round trip time, the mobile DYSWIS sends the TCP check packet, and records the results. After finishing checking all five servers, the mobile DYSWIS shows the results with the average, minimum, and maximum round trip times.

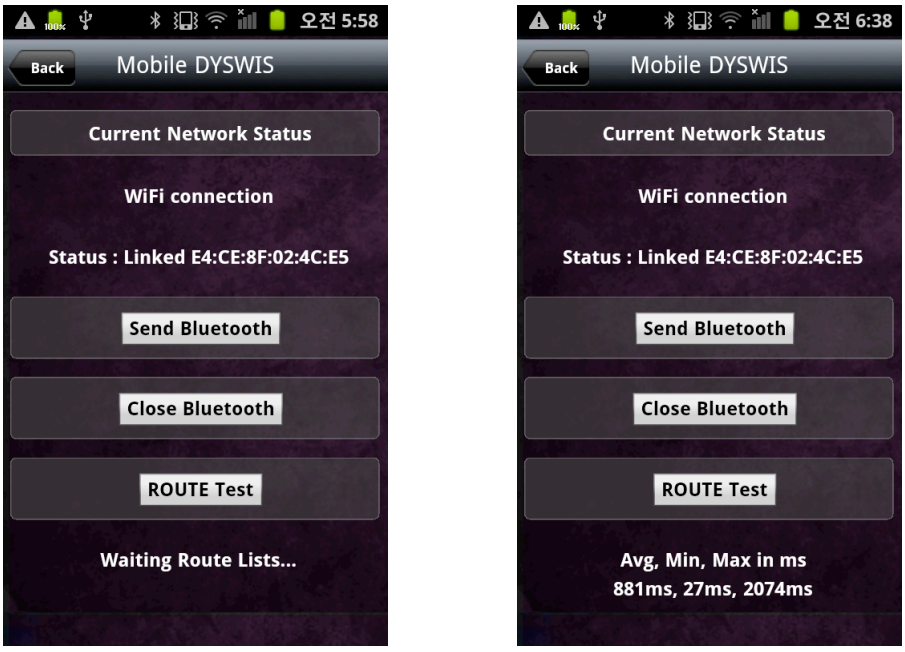


Figure 2.8 - The Route Test Module

3. Implementation Details

3.1. How can make the modules of the mobile DYSWIS be written in JavaScript

3.1.1. Background

The desktop DYSWIS is implemented on the OSGi framework. This enables a user to write the modules for the desktop DYSWIS without modifying the original DYSWIS. In this sense, the mobile DYSWIS should have the features that can write the modules separately; however, the Dalvik VM[2], which is the Java virtual machine for the Android, does not allow loading Java classes because each class in the Dalvik should be signed when the compile time. As a result, there is no way to insert the class files or JAR files during running the application.

3.1.2. Hybrid Application

The solution for the problem mentioned at the section 3.1.1 is that the mobile DYSWIS is written as the hybrid application. The hybrid application means the application is written in HTML5 and JavaScript so that the application can be run all platforms if the platform has the web browser supports HTML5 and JavaScript.[3] By writing the application as the hybrid application, we can write each module in HTML5 and JavaScript. As a result, we can separately write the modules for the mobile DYSWIS without modifying the original application; however, this hybrid application requires the native layer plugins to enable JavaScript to use native APIs, for example, Bluetooth, TCP, or UDP sockets. In this project, I implemented those native layer plugins to use Bluetooth, TCP, UDP, and ICMP in JavaScript.

3.2. How to write the module for the mobile DYSWIS

3.2.1. Android Native Plugin Part

To support JavaScript to use the Android's native APIs, we have to implement some plugins so that those plugins provide the new APIs for JavaScript. The blue arrow of the Figure 3.1 shows what layers we should implement.

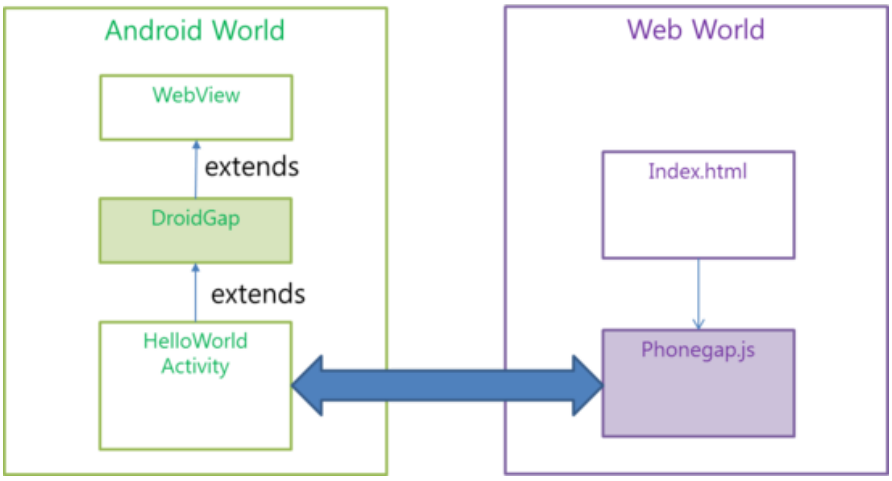


Figure 3.1 - The native plugin bridge to support the native APIs in JavaScript

For this project, I implemented 7 plugins. These plugins will be used again later to extend the mobile DYSWIS. Table 3.1 shows what plugins I implemented, and those purposes.

Plugin Name	Purpose
BTSocket	Supports the Bluetooth
C2DMReceiver	Supports the C2DM service
PGDNS	Supports the DNS Query commands
PGTCPSocket	Supports the TCP socket
PGUDPSocket	Supports the UDP socket
PGTCPRoute	Supports getting the TCP round time
PGPing	Supports generating the ICMP ping

Table 3.1 - The plugins of the mobile DYSWIS to support JavaScript

3.2.2. HTML5 + JavaScript Part

To write the module for the mobile DYSWIS in JavaScript, we should follow next steps.

3.2.2.1. Implement the native Android plugin

At first, we need to implement the native Android plugin. For example, if we want to use the TCP socket in JavaScript, we should wrap APIs related to the TCP socket. Also, the native plugin uses JSON(JavaScript Object Notation) for the data between the native application and the JavaScript layer. This means that we need to make JSON data, not normal binary data. For example, when we receives some bytes from the TCP socket, we have to encode those bytes into JSON. As a result, the native plugin should contain JSON library.

3.2.2.2. Implement the factory class for the native plugin

After creating the native plugin class, we need to write the factory class for the native plugin class. This factory class determines which native object will be passed to JavaScript. In this manner, we can keep TCP or UDP connections at the JavaScript. Normally, objects of JavaScript of one page will be removed when the page is refreshed; however, we cannot keep the TCP connection if the TCP object of JavaScript is released. As a result, we use the factory class to manage objects.

3.2.2.3. Implement the JavaScript class for the native plugin class

If there are ready to use the native plugin and the JavaScript class, we can write the program in JavaScript. The code 3.2 shows how we use TCP in JavaScripts

```
this.startTestTCP = function(port) {
    this.port = port;
    console.log("Start TCP Incoming Test with Port = " + this.port);
    // Remove the port form on the screen, and replace it to waiting TCP connection
PORT
    var htmlText = "Waiting an incoming connection on port " + this.port + "<br />";
    document.getElementById("tcpincoming_result").innerHTML = htmlText;

    // To open and bind the given port,
    // we should use "0.0.0.0:port:ture" for the PGTCPSocket's initilizing mehtod
    tcpsocket = new PGTCPSocket("0.0.0.0:"+this.port+":true");
    tcpsocket.onopen = function() {
        alert('connected');
    };
    tcpsocket.onmessage = function(msg) {
        var data = msg.data;
        console.log("Incoming TCP data " + data);
        alert('There is no problem on the incoming TCP connection!');
        btsocketCommand = "TcpIncomingTest:Result:OK";
        SendBTCommand();
        document.getElementById("tcpincoming_result").innerHTML = formCode;
    };
    tcpsocket.onclose = function() {
        alert('close');
    };
}
```

Code 3.2 - The part of tcpincoming.js

As we can see in Code 3.2, we can read TCP data from the JavaScript event, which is the ‘onmessage’ event in this case. When the native TCP plugin receives data, it will notify to JavaScript by sending the JavaScript event. There are cons and pros. The native plugin automatically creates the TCP and streamer objects so that we can simply use the TCP communication, and we can write another TCP modules without modifying the native application. To write a module without modifying the application is one of this project goal; however, this TCP socket depends on the native Android plugin, there is no way to modify the TCP options if the native plugin does not wrap the original APIs that we want.

In sum, we can write modules of the mobile DYSWIS separately by using HTML5 and JavaScript. Of course, it requires us to write the native plugin to support JavaScript; however, it will give us more benefits after implementing the native plugins; for example, writing modules without modifying the application, supporting another mobile platforms, and so on. In this project, I chose writing the mobile DYSWIS in JavaScript to support the independent module programming environment.

4. The diagnostic Scenario

To diagnostic the network problems on the mobile, I designed the diagnostic test scenario by the top-down approach. Figure 4.1 shows the flow chart of this scenario.

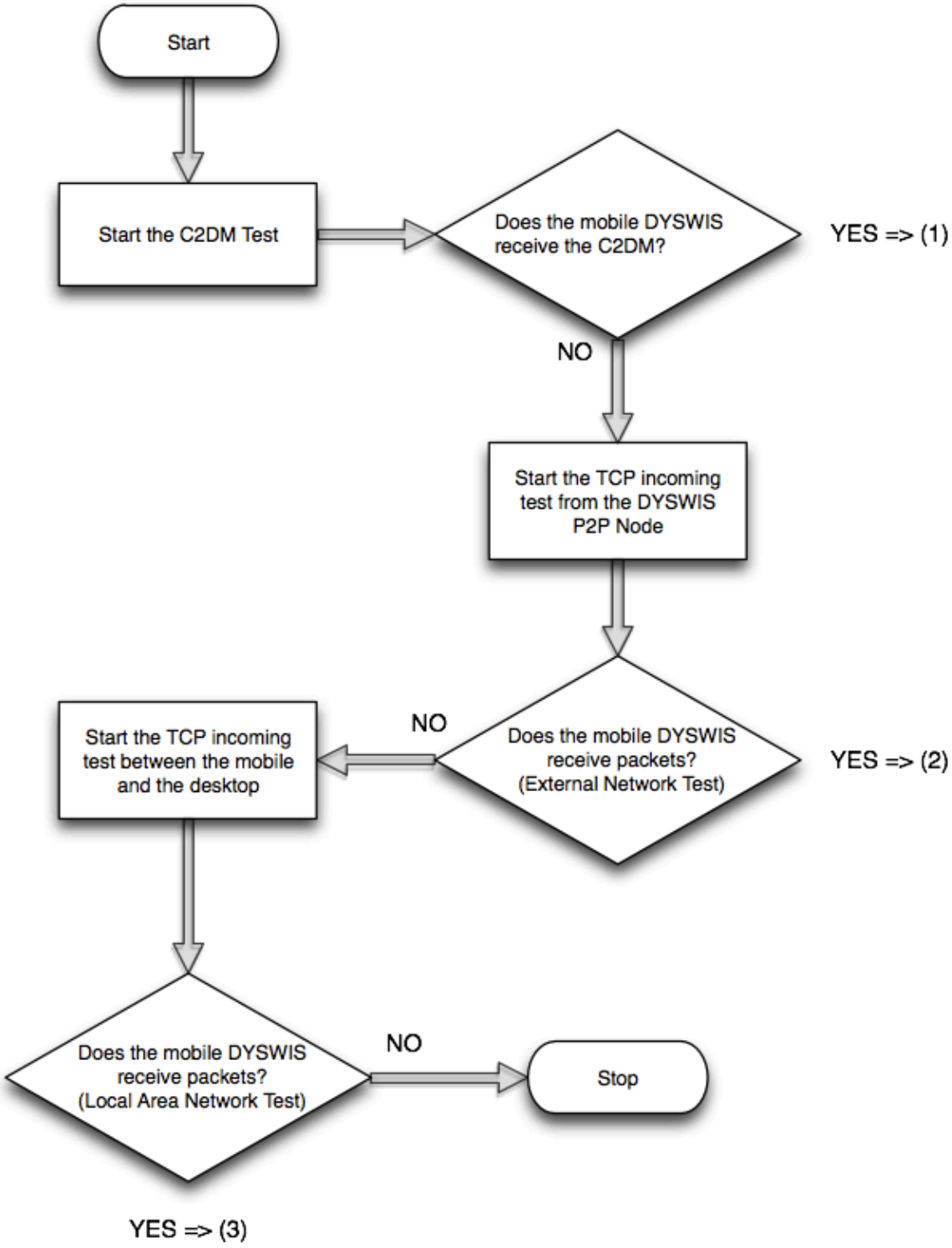


Figure 4.1 - Android C2DM Test Scenario

The purpose of this scenario is to show users whether their C2DM functions work fine. The most Android application are using the C2DM services; however, sometimes the user cannot receive the C2DM notification from Google. In this case, the user may wonder why they cannot receive C2DM notifications. So, first, the desktop DYSWIS requests the C2DM notification to Google. The case of (1) means we can receive the C2DM notification. If we receives the C2DM notification, our network connectivity is good. If we cannot receive the C2DM notification, we run the next test. The second test is waiting the incoming TCP connection from the desktop DYSWYS over P2P. If we can accept the TCP incoming connection, there is no problem on the internet. For the final step, we'll test the local area network. In the case of (3), the local network works fine; however, if we cannot receive the packet, the local area network has some problems. In this case, we may assume that the router or the wireless AP have problems.

5. Future Work Ideas

5.1. Support another mobile platforms

The current mobile DYSWIS supports only the Android platform. The mobile DYSWIS is written in HTML5 and JavaScript, so we can implement the mobile DYSWIS for the other platforms if we implement the native plugins for those platform. The PhoneGap, which I used the library for the hybrid application, supports iOS and Windows Mobile. This means that the mobile DYSWIS can also support those platforms.

5.2. Make the desktop DYSWIS diagnose the mobile DYSWIS

Currently, the mobile DYSWIS uses the power of the desktop DYSWIS; however, the desktop DYSWIS also uses the power of the mobile DYSWIS whenever the desktop DYSWIS cannot use their networks. This is because the mobile DYSWIS has a 3G network, so the mobile device does not lose their network connectivity in common. This property can give help to the desktop DYSWIS to diagnose its network.

Appendix A: References

[1] PhoneGap - PhoneGap is an HTML5 app platform that allows you to author native applications with web technologies and get access to APIs, http://phonegap.com/

[2] Dalvik VM - Dalvik is the process virtual machine (VM) in Google's Android operating system, http://en.wikipedia.org/wiki/Dalvik_(software)

[3] The Hybrid Application - http://blog.brightcove.com/en/2011/11/html5-and-rise-hybrid-apps