Android Implementation of an Auto-Configuration Method for Wi-Fi based MANETs Using Bluetooth

Urbina Espinoza Alberto[1], Díaz-Ramírez Arnoldo[1], Carlos T. Calafate[2]

[1]*Instituto Tecnológico de Mexicali, Av. Tecnológico sn Col. Elías Calles, Mexicali, B.C., Mexico, 21376,*
a.urbinaes@gmail.com, adiaz@itmexicali.edu.mx
[2]*Universitat Poltècnica de València, Camino de Vera sn, Valencia, España, 46022*
calafate@disca.upv.es

Abstract: Without a fixed infrastructure, node configuration becomes a complex issue in *ad-hoc* networks. In this paper, we describe an implementation of a previously proposed auto-configuration method that uses Bluetooth to configure nodes joining an 802.11-based *MANET*. The implementation was done in devices using Android OS, since it is the most widely used operating system for mobile platforms. We introduce the proposed method's architecture, followed by implementation details of client and server modes. We present the results of the experiments performed, showing the efficiency of the model implemented in Android, and comparing it with its implementation in a Linux PC.

Key Words: MANET, Wi-Fi, Bluetooth, Android, Ad-Hoc

1. INTRODUCTION

We live in an era where communication technology has evolved significantly. Nowadays, and thanks to wireless technology, users are able to communicate and share their information from almost every place.
The problem with most of the wireless networks used today is that they need a fixed infrastructure to perform important tasks, such as packet routing.

Mobile Ad-hoc networks (MANETs) (Chlamtac et al., 2003) do not require any fixed infrastructure or a centralized administration, since the nodes of the network are able to auto-organize themselves using diverse topologies. Devices can be added or disconnected from the network in a dynamic way.
MANETs can be used in a lot of scenarios, from a business meeting, to scenarios where communication infrastructure was destroyed by natural disasters.

On the other hand, this flexibility makes MANET deployment a complex issue. One of the main problems when deploying a MANET is the configuration of nodes. Some solutions have been proposed, like PACMAN (Weniger, 2003) and the routing protocol OLSR (Clausen and Baccelli, 2005), which are oriented to the assignment of IP addresses. However, Wi-Fi based MANETs require more parameters for their configuration, such as the network mask, the SSID, the routing protocol, the channel or the encryption type.

The IEEE 802.15.1 standard, also known as Bluetooth, works with a reduced radio range (from 10 centimeters to 10 meters), and has a battery cost up to 97% lower than other wireless technologies.
Bluetooth's topology is based on the creation of piconets, that are sub-networks formed by a maximum of 8 nodes where one master device is always present, and up to 7 slave devices can join it. Scatternets are larger Bluetooth networks that are created when connecting two or more piconets through master nodes.

The Bluetooth protocol stack can be divided in three sections: the *Bluetooth Controller* (Hardware), the *Host Controller Interface*, and the *Bluetooth Host* (Software). The first and latter components are separated by the Host Controller Interface, as can be observed in Fig 1.

ISBN: 978-607-95534-5-6

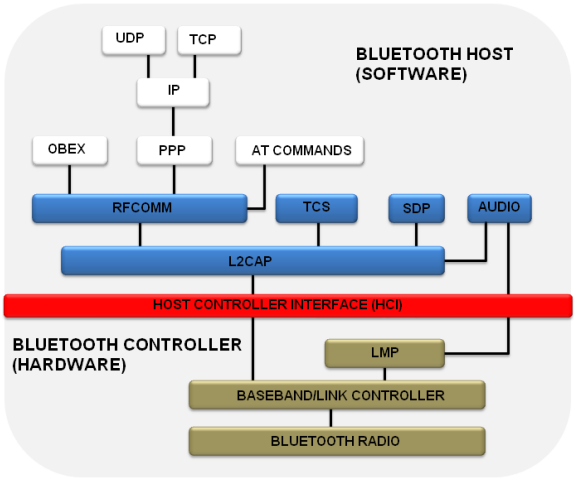

Fig. 1. Bluetooth Protocol Stack

In the higher level layer the L2CAP (Logical Link Control and Adaptation Protocol) protocol is made available, being used for several tasks like segmentation and reassembly of packets, providing good management for unidirectional transmission to other Bluetooth devices, etc. RFCOMM (Radio Frequency Communication), also located in the Bluetooth Host layer, is a simple group of transport protocols, built over L2CAP, that provides up to sixty simultaneous connections for Bluetooth devices emulating serial ports RS-232.

An important property of Bluetooth is that it supports the concept of services, and includes a Service Discovery Protocol (SDP). SDP allows devices to discover what services each other support, and what parameters to use to connect to them. Each service is identified by a Universally Unique Identifier (UUID), where official services are assigned a short form UUID (16 bits rather than the full 128).

In this paper, we introduce an auto-configuration method for IEEE 802.11 based MANETs using Bluetooth, and discuss its implementation in mobile devices using the Android operating system. Additionally, we present the experimental results of the evaluation of our implementation.

With the intention of showing that the auto-configuration method is suitable for most mobile devices, the implementation was done using Bluetooth and the Android OS. Android is a Linux-based operating system running in 72% of the mobile devices worldwide according to results from the third quarter of 2012. It is distributed as open source software. It was developed by Google that releases its code under the Apache License. This permissive licensing allows the software to be freely modified and distributed, as well as the creation of applications to extend device's functionality.

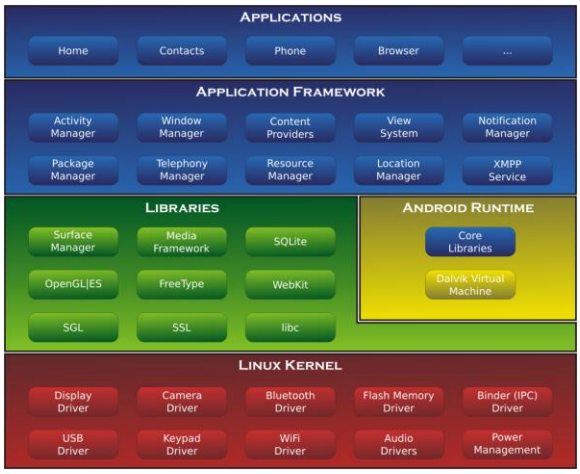

Fig. 2. Android System Architecture

The rest of the document is organized as follows. We introduce the auto-configuration method for MANETs in Section 2. The implementation details of this method are discussed in Section 3. In Section 4, we show the results of the evaluation of our implementation. Finally, in Section 5 we present the conclusions and discuss future work.

2. AUTO-CONFIGURATION METHOD

The original MANET auto-configuration solution was proposed by the Computer Networks Group from the Technical University of Valencia (Cano-Reyes et al. 2006), and it consists in the use of two types of nodes. The first one is the server node, that must have registered the MANET_autoconf service, and that is in charge of assigning different IP addresses to each device using the same Wi-Fi parameters. The other type of node is the client node. These devices want to gain access to the MANET, and for this purpose they will retrieve their configuration parameters from the server node. Fig. 3 shows the architecture of the configuration method.

The Bluetooth technology is only used for parameter retrieval. Once the client node gets the configuration parameters, the Bluetooth interface is turned off and the Wi-Fi interface is activated Thereafter, the rest of the parameters are configured locally, allowing the client to gain access to the MANET.

To connect with the server node, the client must first perform an inquiry operation to discover nearby Bluetooth devices and, once discovered, search for the MANET_autoconf service in each of these devices using the Service Discovery Protocol. Once the server node is discovered, the client must establish an L2CAP or RFCOMM connection with the server.

ISBN: 978-607-95534-5-6

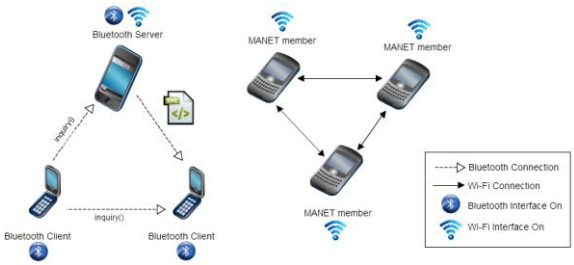

Fig. 3 Auto-configuration System Architecture

The device discovery performed by the client requires the server node to be visible. Thus, it is important to check this before doing the inquiry. The server will also be listening to any incoming L2CAP or RFCOMM connection, but considering the limitations of Bluetooth technology, the server can only attend up to 7 incoming connections simultaneously.

When server and client nodes establish a successful connection, the server proceeds to generate a customized XML file containing the client's parameters, and then it sends the file back to the client. The XML file is created by the server according to each device being configured. Therefore, each XML file created will contain a different IP address but the same routing protocol, SSID, mask, encryption type, encryption key and channel, and of course, specifying that the network will be of the ad hoc type. In Fig. 4 we can observe an example of an XML file sent by the server node.

In this method, the server node is responsible of the IP address assignment, a problem that in a centralized network is solved by a DHCP server. Also, it is important to point out that the IP addresses are assigned in increasing order, and that the server node must maintain a registry of the connected clients at all times. Once the client has received the XML file with the configuration parameters, it disconnects from the server and turns off the Bluetooth interface, while the Wi-Fi interface is activated. Deactivating an interface when it is not needed reduces the energy consumption of the battery, as well as the interference between Bluetooth and Wi-Fi technologies. With the Wi-Fi interface activated, the client node configures locally the parameters assigned by the server in the XML file, and gets connected to the MANET

The only device that has both interfaces activated is the server node, and only when the server wants to be part of the MANET and simultaneously serve more clients. The server node can turn off the Bluetooth interface when joining the MANET, but in this case new nodes can no longer gain access to the network.

```
<?xml version="1.0" encoding="UTF-8"?>
<Android_BlueWi>
    <!-- MANET Autoconfiguration Service -->
    <!-- Client-side parameters-->
    <Client_device>
        <mode>adhoc</mode>
        <SSID>MyAndroidMANET</SSID>
        <channel>1</channel>
        <encryption>NONE</encryption>
        <encryption-key>NONE</encryption-key>
        <ip-address>192.168.1.2</ip-address>
        <ip-netmask>255.255.255.0</ip-netmask>
        <rt-protocol>OLSR</rt-protocol>
    </Client_device>
</Android_BlueWi>
```

Fig 4. Example of an XML file with the client configuration.

3. IMPLEMENTATION

We implemented the proposed method using two different operating environments. The first implementation was done in a personal computer equipped with the Ubuntu 12.04 Linux distribution. The applications were developed using the Java programming language.

Java PC implementation was done using the NetBeans IDE and the BlueCove library that allows the use of the Bluetooth stack. BlueCove runs on any JVM, starting from version 1.1 or newer.

Since version 2.1, BlueCove has been distributed under the Apache Software License. BlueZ support was added in BlueCove version 2.0.3 as an additional GPL licensed module. BlueCove provides Java API for Bluetooth JSR 82. Fig. 5 shows the relationship between BlueCove and the Bluetooth stack. BlueZ is the Linux official Bluetooth Stack. It was originally developed by Qualcomm, and it is distributed under the General Public License. BlueZ is part of Linux kernel since version 2.4, and it has been proven to perform efficiently (Nakasima et al., 2008).

The second implementation was done in devices equipped with the Android operating system, version 4.0.4. The applications were also developed using the Java programming language.

Android includes in their SDK an API for the development of applications that use Bluetooth and Wi-Fi. Android's Bluetooth API supports the development of applications that perform inquiries on nearby devices, using Service Discovery Protocol and RFCOMM for creating connections to other Bluetooth devices. The SDK also includes libraries to write and read XML files in a fast and easy way.

ISBN: 978-607-95534-5-6

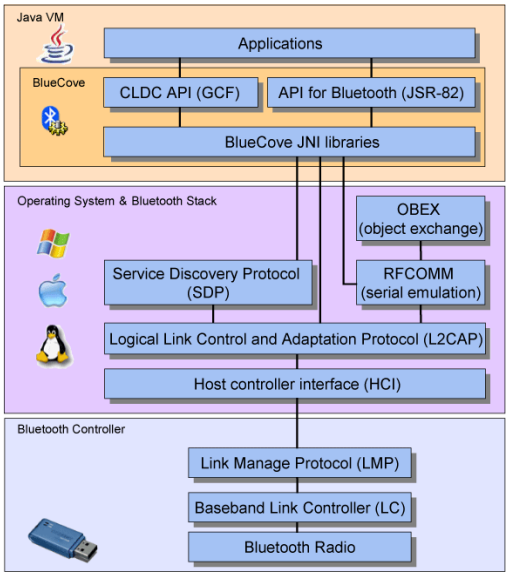

Fig. 5. BlueCove acting as interface between Java and the operative system Bluetooth stack.

The configuration parameters and the MANET's member list are saved and loaded from XML files, making it faster and more efficient than other storage options like SQLite databases. When the application starts its execution it turns on the device's Bluetooth adapter, sets the discoverable option on, and registers the Manet_autoconf service.

CONFIGURATION, MANET MEMBERS and JOIN MANET are the three tasks that the server application can perform. All of them are easy to access by opening the application menu. The CONFIGURATION section from the server application is where the parameters for the MANET are stored. Here, the user can modify the IP address range that will be assigned, as well as the different Wi-Fi parameters, as shown in Fig. 6.

The MANET MEMBERS section shows the user the list of the devices connected to the MANET, the Bluetooth addresses of the client devices, the name and the assigned IP in the Wi-Fi MANET. This information is stored in an XML file. Fig. 7 shows how members are stored in the XML file, and how they are displayed on the screen to the user. Note how IP addresses are assigned in ascending order.

The last option, JOIN MANET, prompts a question to the server device user to determine whether he wants to keep the Bluetooth signal on. It is important to remember that if the Bluetooth signal is turned off, no more clients can be added to the MANET. Once the user selects the option, a loading screen appears while the application completes the process of adding the server node to the MANET.

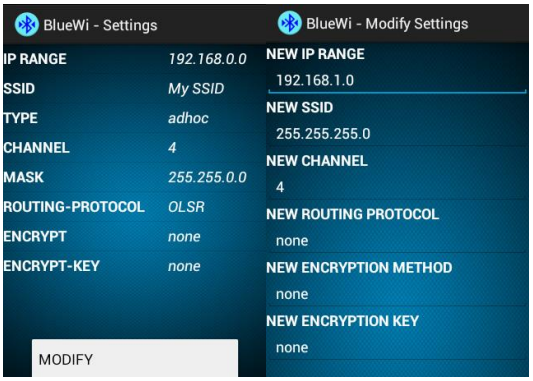

Fig. 6. Configuration interface for the server app.

Once this is done, another message is shown to the user confirming that he is now connected to the network. Fig. 8 shows the GUI for this option.

The client application is simpler than the server one. When the client application starts its execution, it prompts an authorization request for enabling the Bluetooth interface. If the user accepts it, the application automatically starts the device inquiry. When the inquiry is completed, a list of the devices found with the *MANET_autoconf* service registered is shown to the user (envisioning a scenario with two or more servers that are able to attach to a nearby ad hoc network). The user can select a server from the list to connect. The application creates an RFCOMM channel to communicate with the server, and to retrieve the configuration information. The parameters are delivered to the client in an XML file and, after this, the Bluetooth interface of the client node is turned off. The Wi-Fi interface is activated, and then the parameters received in the XML file are used to configure the client node.

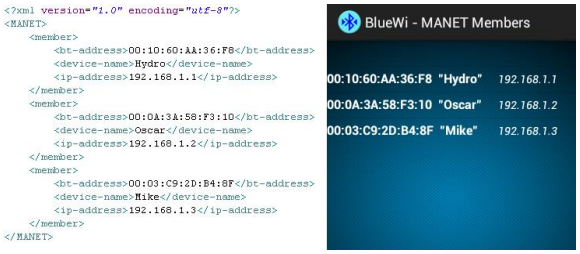

Fig. 7. XML file information displayed in app screen.

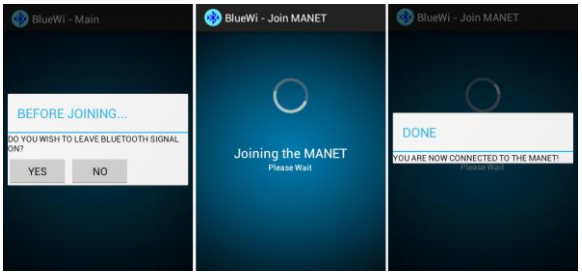

Fig. 8. Server node joining the MANET

ISBN: 978-607-95534-5-6

The configuration of the Wi-Fi parameters of the nodes was not straightforward. The Android API wireless library does not include a complete support for modifying the Wi-Fi parameters. In contrast, in a Linux distribution like Ubuntu, it is possible to use the set of tools to manipulate wireless extensions provided by the wireless-tools package, as well as commands like *ifconfig*. To modify the Wi-Fi parameters in Android, the device must be rooted, which means allowing the execution of privileged commands to the user running the Android mobile operating system. Rooting enables all the user-installed applications to run privileged commands that are typically not available when the device is not rooted. In other words, if the device is not rooted, the application would be unable to perform commands like *ifconfig*.

The main problem when implementing the proposed method in Android was that this operating system has no default support for wireless tools, even if it is rooted. Also, *ifconfig* is the only command that can be used, and it only allows modifying the IP address.

However, as mentioned previously, we need to configure more parameters in order to correctly create an ad hoc network. These parameters are configured using the *iwconfig* command from the wireless-tools package. To include the *iwconfig* command it is necessary to perform a cross compilation. A cross compiler is a compiler capable of creating executable code for a platform other than the one on which the compiler is running. This way, *iwconfig* command was included in the Android operative system with a cross compilation made in a PC using the Linux operative system.

4. EXPERIMENTAL RESULTS

Two sets of experiments were defined to evaluate the efficiency of the proposed method, and to compare the performance of its implementation in Linux and Android. The experiments performed for Linux where performed in netbooks with a dual-core processor working at 1.66GHz and 1GB of Ram Memory, in the other side, for Android experiments nodes where tablets with Android version 4.0.4, dual-core processor at 1GHz and 1GB of Ram Memory, this hardware was chosen due to its really similar features in CPU and memory. In the results, we compared the time elapsed between the instant when the connection is established, and the instant when the parameters are configured, and the device becomes part of the MANET.

The first scenario included one server node and one client node. The distance between nodes was experiment-depended, in the range from 1 to 12 meters. It is important to note that 10 meters is the maximum distance defined by the Bluetooth standard for an efficient communication. Fig. 9 shows the results obtained from this set of experiments. We can notice that, in the first 10 meters, the time needed to define the configuration parameters and send them back to the client in the RFCOMM connection was almost the same. However, beyond that distance, the communication through the Bluetooth channel experienced a notable increase in time, and sometimes was not even completed.

In the second set of experiments, we defined a fixed distance of 5 meters between devices, and we varied the number of client nodes attempting to retrieve their configuration parameters. The number of devices varied from 1 to 9. Since the Bluetooth standard defines that, in a piconet, a master node can only attend 7 simultaneous connections; the two extra devices had to wait in a queue until a new connection can be established. This way, we can also test what happens when the number of nodes is larger than the maximum piconet size. Fig. 10 shows the results obtained from the second set of experiments. We can observe that the time needed to configure the client nodes is almost constant when the number of nodes is less or equal than 7. However, when the number of client nodes is greater than 7, the server node cannot attend all the clients concurrently due the Bluetooth standard limit for piconet size. Once a device disconnects from the server, another one is attended, causing a little gap in the average configuration time for all the nodes.

It is important to note that, in both sets of experiments, the Linux implementation required less time to establish connections and to configure nodes than the Android implementation. This is expected due to the lightly superior hardware performance of the Linux-based PCs when compared to Android platforms

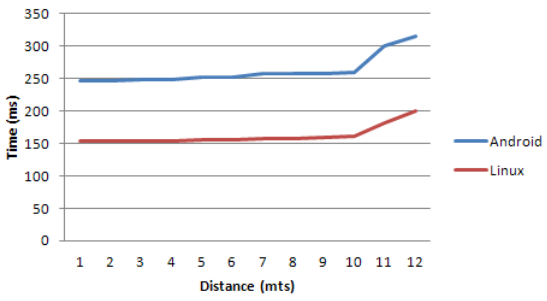

Fig. 9. Results obtained from the first set of experiments.

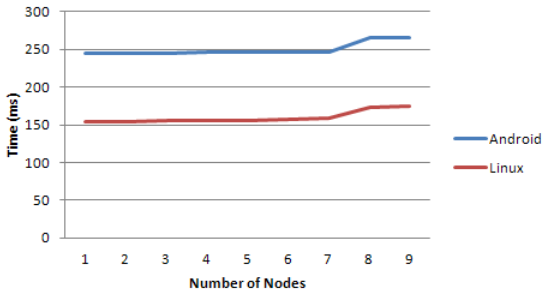

Fig. 10. Results obtained from the second set of experiments

5. CONCLUSIONS AND FUTURE WORK

Mobile Ad-hoc networks do not require any fixed infrastructure or a centralized administration, allowing nodes to be added or disconnected from the network in a dynamic way. However, this flexibility makes MANET deployment a complex issue. One of the main problems when deploying a MANET is the configuration of the different node parameters, such as the IP address or the Wi-Fi configuration details.

In this paper, we presented the implementation of a previously proposed method to solve the auto-configuration problem of a Wi-Fi based ad hoc network.

We implemented the proposed method in mobile devices equipped with the Android operating system. We discussed implementation details and conducted a set of experiments in order to validate our implementation. Additionally, we also implemented the proposed method in a personal computer equipped with the Linux operating system, and compared its performance with the Android implementation. We showed that the auto-configuration method solves the problem of setting the node's network parameters in a MANET. Also, we were able to implement it efficiently in mobile devices running the Android operating system. Finally, we showed that, in the Linux implementation, nodes were configured in less time compared to the Android implementation, although this difference is not significant enough to hinder its adoption in Android devices to make it widely available.

As future work, we plan to implement the auto-configuration method in other mobile platforms.

REFERENCES

Android SDK| Developers webpage.

http://developer.android.com/sdk/index.html.

Bluetooth Official Web Site:

http://www.bluetooth.com

BlueZ official website:

http://www.bluez.org

BlueCove Java Library for Bluetooth: *http://code.google.com/p/bluecove.*

I. Chlamtac, M. Conti, and J. Liu. Mobile ad hoc networking: imperatives and challenges. *Ad Hoc Networks*, 1(1):13-64, July 2003.

José Cano-Reyes, Eduardo Burgoa, Carlos T. Calafate, Juan-Carlos Cano, and Pietro Manzoni. An auto-configuration method for ieee 802.11 based MANETs using Bluetooth. In *Proceedings of the XVII Jornadas del Paralelismo, 2006.*

K. Weniger. Passive duplicate address detection in mobile ad hoc networks. In *Proceedings of the IEEE Wireless Communications and Networking* volume 3, pages 1504-1509, March 2003.

Sukey Nakasima, Francisco Reyna, Arnoldo Díaz Ramírez, and Carlos T. Calafate. First experiences with BlueZ. *Research in Computer Science*, 39:97-114, 2008.

Thomas Heide Clausen and Emmanuel Baccelli. A simple address auto-configuration mechanism for olsr. In *Proceedings of IEEE International Symposium on Circuits and Systems*, pages 2971-2974, May 2005.

Wireless Tools for Linux:

http://hpl.hp.com/personal/linux/tools.html.

ISBN: 978-607-95534-5-6