Published by :
http://www.ijert.org

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 02, February-2020

Android Wi-Fi Direct Architecture: From Protocol Implementation to Formal Specification

Rémy Maxime Mbala[1], Jean Michel Nlong[2], Jean Robert Kala Kamdjoug[3]
[1] The University of Ngaoundéré, Ngaoundéré, Cameroon
[2] Cameroon-Congo Inter-State University, Sangmelima, Cameroon
[3] Catholic University of Central Africa, Yaoundé, Cameroun

***Abstract* –With the proliferation of Android smart phones, it should be possible to set up Mobile Ad-hoc Networks (MANETs) to share information anywhere, anytime. To achieve this goal, it is necessary to have a good wireless communication technology capable of responding to the characteristics of MANETs. Wi-Fi Direct could be a good candidate not only because it allows direct communication between devices without the need for a fixed infrastructure but also because it is present in most smart phones with at least Android 4.0. The study of Wi-Fi Direct has been the subject of several works that denounce the limitations of the technology in terms of construction of large-scale mobile networks. In this paper, we make a detailed technical study of the Wi-Fi Direct framework on Android Operating System to understand how it works and its implementation in order to really overcome their network restrictions. We also provide a formal specification of Wi-Fi Direct on Android by using Z and Object-Z specification languages. This study allows us to understand the functions and the classes of the framework and see at what level changes can be make to implement other solutions.**

Keywords: Wi-Fi Direct, Peer-to-Peer, Android, Ad-hoc Network, WifiP2pManager, Wpa_supplicant, Formal Specification, Z, Object-Z.

I. INTRODUCTION

Since the early announcement of Wi-Fi Direct specifications by the Wi-Fi Alliance[1], the promise of its advertised features and use-cases has motivated a lot of projects, in the aim of exploiting this technology for general network communications. Indeed, the deployment of large-scale, self-organized and self-managed mobile ad-hoc network is a key enabler of many field applications, ranging from military operations to catastrophe management, passing by game and infotainment. Wi-Fi Direct was intended to allow concrete peer-to-peer mobile ad-hoc network deployment, based on smart phones and other handheld equipments, and is embedded in almost all the smart phones sold these past five years.

An ad hoc network consists of a set of geographically distributed equipments, potentially mobile and sharing a radio channel. Equipments can play the role of source, destination or router for communications; they must be able to play these roles at any time, thus exhibiting a peer-to-peer network organization. An Ad Hoc network is opportunistic, self-created, self-organized and self configured [29]. Self-creation means that when the facilities are put together in a neighbourhood, the network is created in an "opportunistic" way such that each equipment is able to communicate with all the others to a hop or hop by hop. Self-organization means that only the interactions between devices are used to provide the necessary control and administrative functions to maintain the network. Nodes cooperate to organize themselves by distributing different roles if necessary. The auto-configuration implies that the nodes must be able to quickly perform certain functions to always respect the defined topology and allow for flexibility, robustness and transition to network wide.

The specifications of Wi-Fi Direct were developed by Wi-Fi Alliance consortium, and are published in [6]. This technology allows connecting two compatible devices for data transfer streaming. Being a modification of classical Wi-Fi, it offers the same performances in terms of range and bandwidth. Wi-Fi Direct differs from classical Wi-Fi in the sense that it directly connects two equipments, with no third party. Android smart phones from version 4.0 are natively equipped with this technology (the API is called Wi-Fi Peer-to-Peer [6]), and can opportunistically create Ad Hoc groups in a neighborhood for sharing information. With Wi-Fi Direct, the group formation process is divided into four phases: discovery, negotiation of the group owner, the WPS provisioning and configuration of addresses [6].

Taking advantage of these capabilities, several studies [2, 3, 8, 15, 26, 30] have been carried out to evaluate this technology and to study the feasibility of building a true mobile Ad Hoc network as presented above. These works are mainly experimental, and highlight the difficulties of putting into practice all the interesting features commercially announced. It then appears important to well circumscribe the purpose and the commercial aspects of the technology, and its concrete operational capacities. Given the lack of a native specification for Wi-Fi Direct, this work aims to study its operational functioning, formalize the use scenarios and to provide the formal specification by using Z and Object-Z specification languages, for Android.

Formal specification helps build more robust and maintainable software with fewer defects than systems developed without using formal methods [36]. We choose Z and Object-Z languages because they are the formal specification languages predominantly used in model based specification and they have an important feature over other languages: powerful semantic and calculus (predicate calculus and set theory), strong support of objects and its specification style corresponds directly to object oriented programming constructs, while UML-B weakly supports the concepts object and VDM++ does not have exact formal calculus [36, 37, 38].

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 02, February-2020

The remainder of this paper is organized as follows. Section II provides some background on Wi-Fi Direct standard and specification, along with advertised features and performances. In section III we present related works on the usability of Android Wi-Fi Direct stack, and the results and conclusions obtained. The overall architecture of the Android Wi-Fi Direct implementation is presented in Section IV, which formal specification is derived in Section V, along with a discussion on its capabilities. Section VI concludes the paper.

II. WI-FI DIRECT GROUP FORMATION PROCESS

In Wi-Fi Direct, the group formation is divided into four different phases namely discovery, Group Owner (GO) negotiation, Wi-Fi Protected Setup (WPS) provisioning and address configuration [2, 6].

a. Discovery: The Wi-Fi Direct device discovery process is used to search for and locate the devices that are intending to participate in a specific service. The device that turns on the Wi-Fi Direct mode initiates scan, listen, and search phases to find other WLAN-based D2D-enabled devices in the currently used frequency. Specifically, a Wi-Fi Direct device can first conduct 802.11 scanning defined in 802.11 standard [7] in order to identify surrounding Peer to Peer groups. The specification is based on the 2.4-GHz frequency used in 802.11b/g/n, but it can also be extended in the future to 5.0 GHz, which is used in 802.11a/n/ac. In the scanning process, the device periodically switches its channel and transmits a probe request message in each channel. If the channel of another WLAN-based D2D device is timely matched with the scanning device at a specific moment, the probe request can be received by another device, which can respond with a probe response. After scanning, the device enters the find phase, in which the default channel is listened to and the mainly used channels of the frequency (e.g. social channels of 2.4 GHz) are searched for. In the find phase, the devices alternates between listen and search states [7, 9]. A device in search state conducts active scanning sequentially at three social channels (channels 1, 6 and 11 of 2.4 GHz) by transmitting a probe response frames. In listen state, the device conducts idle listening at listen channel for a random duration, during which it replies to probe request frame by transmitting a probe response frame. The example in Fig. 1 shows how two devices with Wi-Fi Direct capabilities progress through their device discovery process to discover each other.

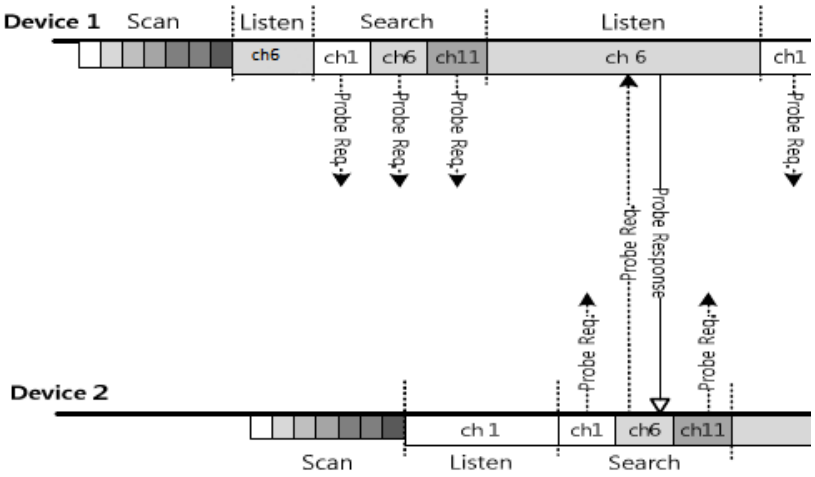

Fig. 1 : Device Discovery process in Wi-Fi Direct [9]

b. GO negotiation: Once the devices share the probe messages, the association process is initiated to establish Peer to Peer connection and select a Group Owner (GO). This is done by using a three-way handshake association messages in the channel that they used to share the probe messages. The role of Group Owner is given to a single device, which becomes responsible for transmission of periodical beacon messages for maintaining connection with group clients. To elect a GO, each node is given an intent value ranging from 0 to 15. This intent value is compared between the devices, and the device with the highest intent value becomes the GO[2] [2, 6, 15].

c. WPS provisioning: Wi-Fi Direct uses Wi-Fi Protected Setup (WPS) for initial set up and authentication. WPS is based on WPA-2 security and uses Advanced Encryption Standard (AES)-CCMP as cypher and randomly generated pre-shared key for mutual authentication [6]. The Group Owner takes the role of registrar; the key responsibilities of the registrar are granting and revoking network access for the client devices. It

[2]In case both P2P devices send equal GO Intent values, a tie breaker bit is used for decision and the device with tie breaker bit set to 1 shall become GO.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 02, February-2020

is also responsible for passing on the necessary information such as group ID, the operating channel, and the pre shared key to the client devices. WPS provisioning is also divided into two phases: in the first phase the security keys are generated by the internal registrar and in the second phase the devices disconnect and reconnect using the key that is generated in the first phase [2, 3, 6, 26].

d. **Address configuration:** Finally, DHCP is used to assign IP addresses to the Peer to Peer devices. The Group Owner acts as the DHCP server and assigns the clients IP addresses. In order to obtain an IP address, the client and the DHCP server exchange DHCP discover offer, request and acknowledgement messages [6].

III. RELATED WORKS ON THE WI-FI DIRECT

Several works were done on Wi-Fi Direct with different objectives. The works focus on experimental evaluation of the protocol's features as well as the development of novel services using Wi-Fi Direct. We present a number of these works in this section. The presentation is not exhaustive, it aims to showing the different lines of research on which the scientific community has looked and the difficulties that people have encountered while experimenting with Wi-Fi Direct. An extensive survey on Wi-Fi Direct technology and comprehensive review of state-of-the-art are presented in [1].

The works done in Wi-Fi Direct can be categorized into four areas according to the research lines and the expected objectives. Some are looking at the evaluation of the performances of the technology, others on the device discovery, the group formation and scalability for communication or on the energy management.

1. Performance evaluation

This is to evaluate the performance of Wi-Fi Direct according to certain parameters: the time of device discovery and group formation, the bandwidth actually used, the transmission distance, multimedia delivery, etc. In [2] and [3], authors presented performance evaluation of the device discovery and the group formation delay using standard, persistent and autonomous group formation mode. In these works, the autonomous mode takes about 1s to discover other equipment, while the discovery times in the standard mode and the persistent mode are in the ranges [1.48s, 15.35s] and [1.5s, 15.9s] respectively. For the group formation, times are 3 to 6 seconds in the autonomous mode and 5 to 9s in the standard and persistent mode. The performance evaluation of Wi-Fi Direct for multimedia delivery and streaming applications is presented in [4]. The authors choose the Round Trip Time (RTT) and bandwidth as performance metrics; they make a comparison between Wi-Fi Direct and Bluetooth networks. In addition they evaluate the impact of increasing number of P2P devices. The connection establishment overhead and delay were evaluated in [5] where authors developed "Electrical Business Card Communication over Wi-Fi Direct (EriCC-W)", a Wi-Fi Direct based data transfer system, to identify smart phones using gestures and transforms the gestures into frequency spectrum to authenticate devices. These works revealed that the Wi-Fi Direct has shorter transmission delays than the Bluetooth and can maintain connections over long distances (between 150 and 200 meters). Thus Wi-Fi Direct could be an interesting technology to share information in a certain range but the discovery and the group formation times still remain an obstacle especially for multimedia applications.

2. Device discovery

Empirical measurement results in [2, 3] show that device discovery accounts for the largest portion of the overall time required completing P2P group formation procedure. From these measurements, we can notice that the device discovery may take over ten seconds in the worst case. It implies that reducing device discovery time is most critical issue for satisfactory service start-up.

In order to tackle this problem, authors in [8] proposed Listen Channel Randomization (LCR) scheme to reduce device discovery delay. They developed an elaborate Markov model in order to analyze the legacy find phase based on the channel status of the three social channels. The proposed scheme was tested using NS3 simulations which revealed that LCR can significantly reduce the device discovery, yielding up to 72% delay reduction compared to the legacy find phase [8]. J. Feng and al. [10] sat up the experiments connecting two mobile devices using Wi-Fi Direct and performed the loss channel analysis. They also proposed a new loss model scheme based on Gilbert Elliot model which can facilitate the theoretical analysis when designing or optimizing wireless networks. In [9], authors proposed also a scheme to reduce the delay induced in the association process of Wi-Fi Direct. They designed a so called "client-aided search" of Group Owners where the mobile clients can share the information of Group Owners that they have associated with in the past. In this scheme, channel and location information of multiple Group Owners can be shared by client devices, which can reduce the delay in the forthcoming association process. The proposed solution was also tested using NS3 simulation.

3. Energy management

According to the Wi-Fi Direct specification [6], the Group Owner is the central device of the group, he is sometimes called Soft Access Point since it interconnects all devices present in the group and it manages all theirs communications. Because of this, it is important for the Group Owner to have built-in mechanisms to better manage energy since it is a mobile device. Two mechanisms are presented in [6]: Notice of Absence (NoA); the Group Owner announce time intervals where clients are not allowed to access to the communication channel regardless of whether they are in power saving mode or not, and Opportunistic Power Save (OppPS); it is about taking advantage of the sleep periods of P2P Clients. The OppPS and NoA are considered insufficient in services which are using periodic data transmissions such as video streaming, screen sharing, multi-user on-line gaming [1].

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 02, February-2020

Then several other solutions have been proposed to optimize these two first mechanisms.
Adaptive Single Presence Period (ASPP) and Adaptive Multiple Presence Periods (AMPP) [14] are the two first algorithms proposed to improving the performance of OppPS and NoA schemes. They allow a portable device implementing Wi-Fi Direct (e.g. a mobile phone) to offer access to an external network (e.g. a cellular network) while addressing the trade-off between performance and energy consumption in a configurable manner. The ASPP scheme can be used with both OppPS and NoA, whereas the AMPP scheme can be used only with NoA however AMPP can also improve the performance of ASPP. In ASPP scheme, the P2P GO computes a single presence period in each beacon interval based on the amount of traffic at the up-link of the P2P GO. The presence period is shortened if the up-link is congested and vice versa. In contrast to ASPP, the AMPP uses multiple presence periods in a beacon interval and hence it can be used only with NoA scheme. Algorithms running on P2P GO estimate the raw bandwidth on the up-link external network and accordingly adjust the number of presence periods [1]. The simulations show that ASPP and AMPP can manage to reduce the power consumption of Wi-Fi Direct devices acting as Access Points (50-90%) without introducing major user experience degradation.
In the same order or ideas, two others approaches to compute absence periods for energy saving using NoA scheme are proposed in [2]: A static policy, where the P2P GO advertises a fixed duration presence window after each Beacon frame and a dynamic policy, where the P2P GO adjusts the duration of the presence window based on the traffic conditions using Adaptive Single Presence Period (ASPP) algorithm. The results of this work are not very far from those of [14].
Dynamically Synchronized Power Management (DSPM) scheme is proposed in [12] to dynamically synchronize the active time slots with the data transmission time slot to further enhance the energy efficiency of OppPS and NoA. In [13], Dynamic Power Save (DPS) is proposed; A P2P Group Owner which implements DPS mechanism toggles between OppPS and NoA base on traffic characteristics. Therefore the Group Owner must be capable to decide and switch the power save mode if the applications in the network are changed. To dynamically adjust the parameters for OppPS and NoA schemes to improve the performance of Wi-Fi Direct network, H. Yoo and al. [11] proposed a Traffic Aware Parameter Tuning Scheme (TAPS). TAPS consumes less energy when the maximum values for OppPS and NoA are used whereas no improvement in throughput. However for improve throughput lowest values are used and TAPS consume more energy.

4. Group formation

The technical specification states that Wi-Fi Direct can permit to build a large-scale wireless network by interconnecting groups, but it does not specify how it is possible. So several works have been done to find out how this can be done. Most of these works focused on the experimental evaluation of basic standard features (with a limited number of nodes), others trying to overcome Wi-Fi Direct limitations encountered through hacks and/or by rooting the devices. Related works can be divided depending on their main optimization target: (i) selection of the best GO, (ii) autonomous group formation (bypassing the user's authorization and intra group communication), and (iii) inter-group communication.

- **Group Owner selection**

As we said above, the Group Owner election is performed in GO Negotiation phase where devices negotiate together to which will act as Soft Access Point by declaring their intents. Several studies have been made in order to propose more optimal solutions for the choice of the GO.
WD2 [15] aimed at automatically selecting Group Owner based on the Received Signal Strength Indication (RSSI) measurements. Each device collects the RSSI reading from nearby devices and the Group Owner Intent is computed based on such collected measurements. Their prototype experiments indicate that WD2 increases the average throughput by 45% over conventional Wi-Fi Direct.
For Menegato et al. [16], the device who act as GO should change dynamically, and the choice of a new GO should be based on the residual energy of the candidates. With the same idea, [17] proposed an Efficient Multi-group formation and Communication (EMC) which exploits the battery specifications to compute the rank which will permit the devices to qualify potential Group Owners. In [18], authors proposed Wi-Fi Direct Group Manager (WFD-GM), a middleware layer protocol for the configuration of Wi-Fi Direct groups to enable to select the best Group Owner and the creation of opportunistic network. They combine several features such as the level and the capacity of the battery and the number of discovered equipments, to evaluate the suitability of a node to act as Group Owner in a specific context. Three other different approaches to choose Group Owner are presented in [19]: the device with the highest ID in the surroundings, the device that has the shortest average distance from the other nodes, the node with less mobility with respect to its neighbors. Authors in [20] proposed a combined metric approach to select the Group Owner based on several parameters such as the mobility degree, the distance, the residual energy of the battery, and the number of discovered neighbors. The parameters are normalized and weighted to compute Intent Values of each device. An election algorithm is then used to select the P2P Group Owner.
The most part of the results of these works are presented by simulations and they could permit to select the optimal Group Owner but could also increase the group formation times for more complex algorithms.

- **Intra group communication**

The main objective of the research in this part is to improve the communication protocol in a group, for example by modifying the initial topology. Typically, the topology used is similar to that of clusters where the Group Owner plays almost the same roles as the Cluster Head [16]. Once the Group Owner is elected, it becomes the central device through which all communications in the group are routed.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 02, February-2020

Devices are all connected to the Group Owner and not between them.

The problem with this topology is that the Group Owner is the central equipment of the group, and if it leaves the group for some reasons (mobility, battery down, etc.), the group is destroyed and the negotiations must start again to elect a new one.

Thus, A. A Shahin et al. [21] introduced a framework that enables devices in one Wi-Fi Direct group to communicate by managing the topology changes in the group as well as the data exchange between devices. In [22], authors introduced the concept nominating Backup Group Owner that can replace the Group Owner, if he leaves the group, and permit to acceleration the group reconstitution. Lombera et al. [23] proposed iTrust, a peer management over Wi-Fi Direct that enable peers to form a mobile ad hoc network for decentralized information sharing while Park et al. [24] proposed DirectSpace, a framework for collaboration between devices, which provides a mean for sharing workspaces users over Wi-Fi Direct. [16], [17], [18] and [19] also presented mechanisms for managing intra group communications. These proposed frameworks were validated through the implementation of chat applications over multiple Android based devices and show that the initial topology of Wi-Fi Direct groups, as define in his specification, can be changed to manage intra group communications better.

- **Inter group communication**

Inter group communication is not introduced in Wi-Fi Direct specification. It is said that it is possible to interconnect multiple groups; the specification does not clearly address this issue. Several researches proposed solutions in order to have a large-scale network by interconnecting several groups.

According to [6], the group members are all connected to the Group Owner and cannot belong in many groups. The Group Owner is the only device that can belong to several groups; in this case it will be owner in one group and client in the other.

There are several solutions in the literature which tackle this problem. In [25], Duan et al. proposed a method for establishing multi-group communication in Wi-Fi Direct by letting the Group Owner to connect as a legacy client in another group using WLAN interface. In the same logic, C. Casseti et al. [26] exploit tunneling in the transport layer to overcome the physical limitations that prevent multiple groups' interconnection. In the defined topology, the Group Owner uses his P2P interface to connect to devices in the same group and his WLAN interface to connect to another Group Owner in another group. It is also the basis of the ideas of the authors in [27] who use the multicast to permit the communication between groups. In [28], authors implemented a proactive routing protocol using off-the-shelf smart phones to enable efficient message delivery over a multi-hop mobile ad hoc network.

IV. ARCHITECTURE OF THE ANDROID WI-FI DIRECT IMPLEMENTATION

In this section we make a detailed technical study of the Wi-Fi Direct framework on Android Operating System to understand how it works and its implementation with the aim of improving it and to offer a mobile large-scale wireless network infrastructure. So we have downloaded the source code of Android from the 4.0 version Ice-Cream Sandwich (API level 14) and higher from Android Open Source Project (AOSP) [31] and we study the connectivity and the connection system of Wi-Fi Peer to Peer on Android. This study leads us to understand the functions and the classes of the framework of the wireless peer-to-peer on Android and see at what level changes can be make to implement other solutions like [32] who has dive into Android networking by adding Ethernet connectivity.

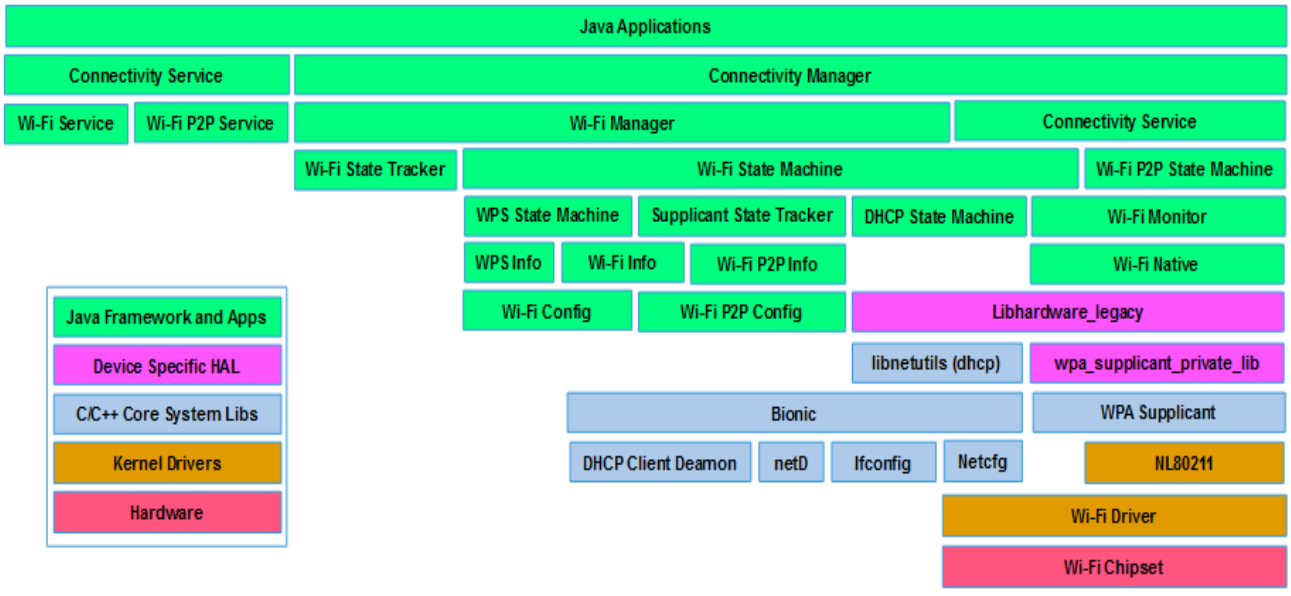

Fig. 2: Layer structure of Wi-Fi and Wi-Fi Peer to Peer on Android [32]

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 02, February-2020

The Android operating system is divided into five main layers [33]: the kernel, in which are developing the hardware drivers; the Hardware Abstraction Layer (HAL), which provides interfaces that expose device hardware capabilities to the higher-level java API framework; Native C/C++ libraries, which permits to built many core Android system components and services; the java API framework, which allows developers to write applications for users and systems Apps, which provides a set of core apps for email, SMS messaging, calendars, contacts, etc.

As shown in Figure 2, the Wi-Fi Direct connectivity system is spread into several layers of the Android system: The hardware level where the chipset reside; the kernel that contains the Wi-Fi drivers; system libraries among which is the WPA supplicant; the Hardware Abstraction Layer which are the C++ functions that allow applications to communicate with hardware; and the java framework which provides the Software Development Kit (SDK). The package that contains the Wi-Fi Direct framework is the Wi-Fi P2P Manager which uses the same low level functions that the standard Wi-Fi package (Wi-Fi Manager). By further studying the Wi-Fi P2P Manager package, we came up with the following figure 3 which shows the architecture of Wi-Fi Peer-to-Peer mode in Android and interactions between the different functions in these levels and how it works.

Typically, we can divide the architecture of Wi-Fi Direct in three parts or levels: the Software Development Kit (SDK), the Java Native Interface (JNI) and the kernel space. The SDK and the JNI are part of user space because users can access and custom them for it specific uses.

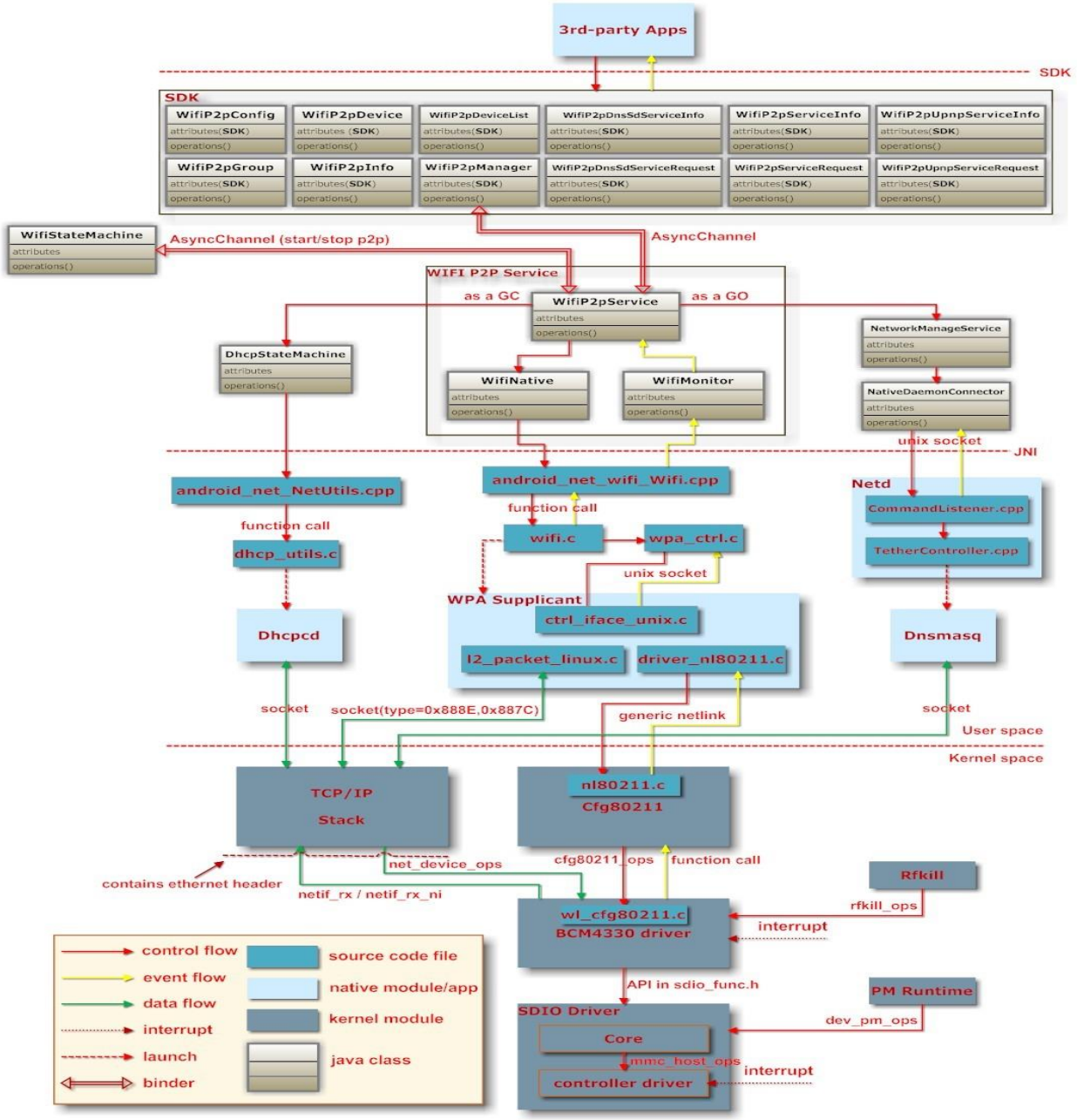

Fig. 3: The interactions between the components of Wi-Fi Direct on Android

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 02, February-2020

As shown the legend in figure 3, the yellow arrows represent the flow of events from the kernel to the upper layers while the red arrows represent the flow of control that go from the upper layers to the kernel and the green arrows represent the flow of data. The SDK layer is the java framework that is split into two main parts: the SDK which allows programmers to write applications and the WIFI P2P Service which manages the services. In the SDK, *WiFiP2pManager*[3] is the main class that should first be called by the program and that uses other classes to get specific information. For example, it will use the *WifiP2pConfig* class to get Wi-Fi P2p configuration after setting up a connection; the *WifiP2pDevice* class to get information of a Wi-Fi P2p device; *WifiP2pInfo* class to get connection information about a P2p group. The *WifiP2pManager* class interacts with the *WifiP2pService* via the *AsyncChannel* binder. *AsyncChannel* binder is an asynchronous channel between two handlers which may be in the same process as with the *WifiStateMachine* class or in another process as with the *WifiP2pManager* class. The *WifiP2pService*[4] class includes a state machine to perform Wi-Fi P2P operations. Applications communicate with this service to issue device discovery and connectivity requests through the *WifiP2pManager* class. The state machine (*WifiStateMachine* class) communicates with the Wi-Fi driver through wpa_supplicant and handles the event responses through *WifiMonitor* class. So *WifiMonitor* class is used to monitor wpa_supplicant status change and notify Android framework. It listens for events from the wpa_supplicant, and passes them to the state machine for handling. *WifiNative* class provides native calls for start/stop the supplicant daemon and sending requests (various commands) to the supplicant daemon. The *WifiStateMachine* class[5] tracks the state of Wi-Fi connectivity, all events handling and all changes in connectivity. It extends the *StateMachine* class[6] which defines a Hierarchical State Machine (HSM). The HSM processes messages and arranges states hierarchically. In the HSM, when the transition is made, the common ancestor state that is closest to current state is firstly found; then exit from current state and all ancestors state up to but not include the closest common ancestor, finally enter all of new states below the closest common ancestor down to the new state. The figure 4 presents the state for P2P operations defined in the *WifiP2pService* file. On Android, if the device supports P2P operations, the initial state will be the P2pDisabledState state and the P2pNotSupportedState state on the other hand. Assume that we want to transit from WaitForUserActionState state to UserAuthorizingInvitationState State, since the closest common ancestor for the two states is the DefaultState state, WaitForUserActionState.exit(), P2pDisabledState.exit(), P2pEnabledState.enter(), GroupCreatingState.enter(), and UserAuthorizingInvitationState.enter() commands will be called in sequence.

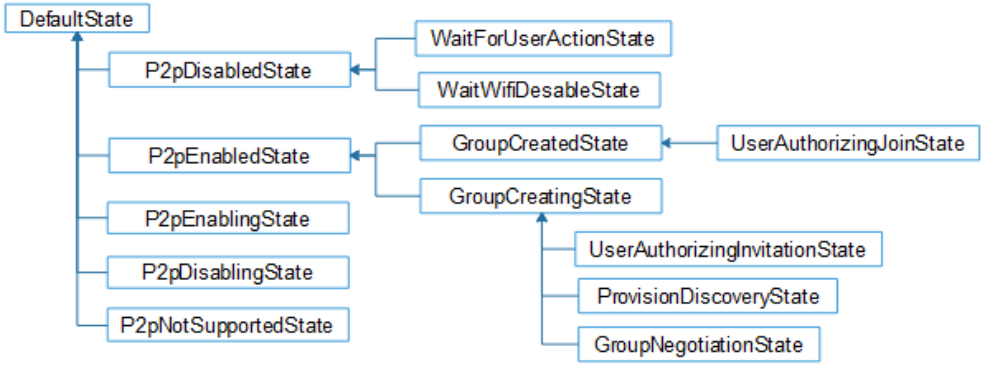

Fig. 4: P2P state machine defined in *WifiP2pService.java* file

The *WifiSateMachine* class supports state machine for Soft Ap and Client operations while *WifiP2pService* handles state machine for p2p operation. These two classes co-ordinate to ensure that only one operation exists at a certain time. The *WifiP2pService* controls also the state machines that interact with the native DHCP client (*DhcpStateMachine* class) or the network manager service (*NetworkManagementService* class) depending on whether the device is a Group Client or the Group Owner respectively. The *DhcpStateMachine*[7] class extends also the *StateMachine* class and defines the state machine that interacts with the native DHCP client and can communicate to a controller. It permits to wakeup or renewal the DCHP using the native DCHP client and provides notification right before DHCP request or renewal is started. The *NetworkManagementService*[8] is an interface for android framework to access to the network interface in the kernel. It works with *Netd* to give commands to network driver. The *NetworkManagementService* communicates with *Netd* (*NativeDeamonConnector*[9]) via sockets for sending commands. To handle the requests, *NativeDeamonConnector* wraps commands as command objects.

[3]Defined in frameworks/base/wifi/java/android/net/wifi/p2p/ within AOSP source tree
[4]Defined in frameworks/base/wifi/java/android/net/wifi/p2p
[5] Defined in frameworks/base/wifi/java/android/net/wifi/ within AOSP source tree.
[6] In frameworks/base/core/java/java/com/android/internal/util/stateMachine.java
[7]In frameworks/base/core/java/android/net/DhcpStateMachine.java
[8]In frameworks/base/services/java/com/android/server/NetworkManagementService.java
[9]Inframeworks/base/services/java/com/server/NativeDeamonConnector.java

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 02, February-2020

The Java Native Interface (JNI) is an interface software library that allows Java SDK codes to call or to be called by native applications (i.e., hardware and OS specific programs), or with Native C/C++ libraries. The JNI code written in *android_net_wifi_Wifi.cpp*[10] communicates with HAL layer by function calls. The HAL layer code is written in the *wifi.c* file which communicates with wpa_supplicant over control interface (*wpa_ctrl.c*) by using UNIX sockets. The wpa_supplicant accesses the wireless driver in the kernel via the file *nl80211.c*; but this depends on the driver implementation. So if the wireless driver is implemented using NL80211 interface, the device can use wpa_supplicant_8 in Android.
The following sections provide details on the operations of the Wi-Fi Direct in the SDK and in the wpa_supplicant.

1. Wi-Fi Direct API in the Software Development Kit (SDK)

The Android API level 14 and higher incorporates the opportunity for applications to discover, connect and communicate by the use of Wi-Fi Direct. *WifiP2pManager* is the primary class, which is composed of the following three main parts: Listeners, Request methods, Intent actions.

- **Listeners**

The message passing for Wi-Fi Direct in Android is asynchronous and the API specifies listener callback methods that are responsible for reacting to requests from the application. The following five different interfaces represent the various listeners: *ActionListener*, *ChannelListener*, *ConnectionInfoListener*, *GroupInfoListener* and *PeerListListener*. Each of these interfaces has callback methods which are triggered when a response is sent. The *ActionListener*'s callback methods inform whether the operation was successful or not. In case of a failure, the callback will convey a constant to point out the reason. This reason can be one of the following: *ERROR* denotes that the reason was due to an internal failure, *P2P_UNSUPPORTED* indicates that Wi-Fi Direct is not supported by the current device and *BUSY* means that the framework is busy, and therefore unable to serve the request. The *ChannelListener*'s callback will be triggered if the channel gets disconnected from the framework. The remaining listeners are triggered when some specific requested information is available.

- **Request methods**

The API has defines nine different request methods. Some of them are required to be implemented, while others are optional. By using these methods, the application will be able to request the operating system to perform specific actions. Each of them will trigger asynchronous message requests and they should therefore be able to react when responses are sent. This is why each method includes a listener for callbacks.
In order to implement Wi-Fi Direct functionality in an application, a registration to the Wi-Fi framework is required. This is realized by executing the *initialize*. All other request methods in the Wi-Fi Direct API depend on this registration. Hence, this must be the first Wi-Fi Direct operation to be performed. The figure 4 following shows a proposed state machine of the initialization process. When the application enters the initialized state it should be able to discover other peer devices.

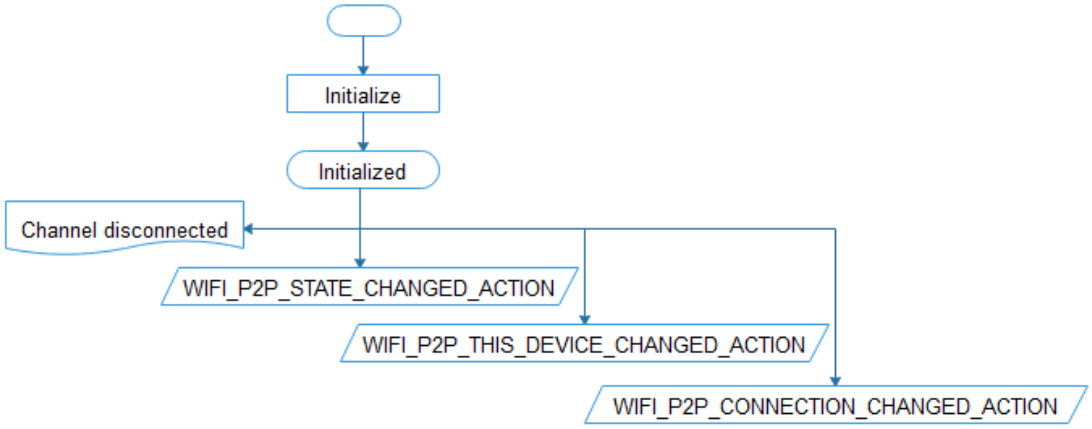

Fig. 5: The initialization process

To be able of finding peer devices, the application must execute the *discover_Peers* method. This operation initiates a peer discovery, which involves sending a request to the framework to scan for available peer devices. If the request is successfully achieved, the discovery procedure will stay active until a P2P group is formed or a successful connection request is initiated. When the application knows that peer devices are discovered, it can request for the current list of devices from the framework by calling the *requestPeers* method. The figure 5 following shows a proposed state machine of the discovery process with a suggested sequence of the related request methods

[10]In frameworks/base/core/jni/android_net_wifi_Wifi.cpp

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 02, February-2020

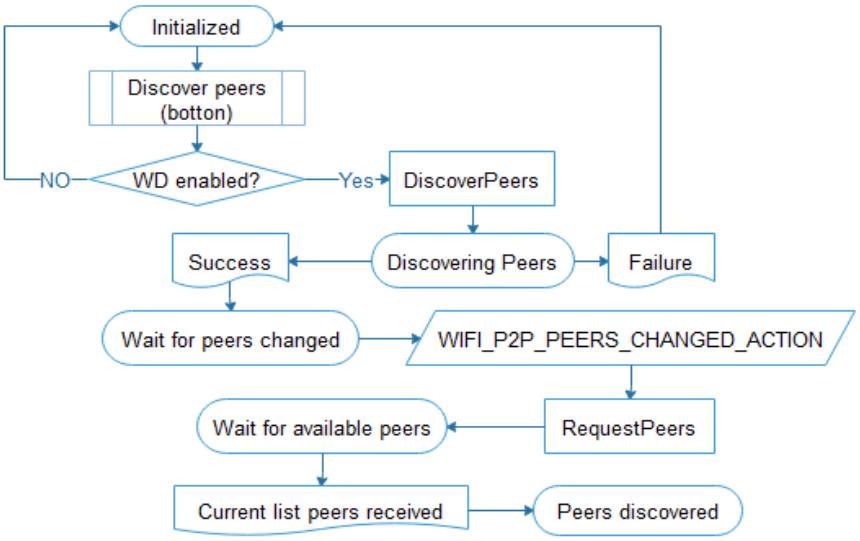

Fig. 6: The discover peers process

When the current list of peer devices has been received and the application enters the peers discovered state, a connection request can be initiated with one of the devices in the list. This is done by executing the *connect* method. If the current device is not already part of a P2P group, this request will initiate a group negotiation with the peer device. A group negotiation is required in order to decide which device that is going to act as the group owner. If the current device is already part of a group, an invitation to join this group is sent. If an ongoing group negotiation ought to be cancelled, the *cancelConnect* method must be executed. Upon a successful group negotiation and when the application knows that the connection has been changed, it can detect if network connectivity exists. If so, a request for connection info can be inquired by executing the *requestConnectionInfo* method. By doing so, the application will be able to attain the following details: if a group has been formed; the group owner's IP address; if the current device is the group owner.

If a group has been formed, a request for group info can be inquired by executing the *requestGroupInfo* method. The following information will be received by the application: the list of client devices that are currently part of the P2P group; the name of the interface the group is using; the Service Set Identifier (SSID) of the group; the details of the group owner in a *WifiP2pDevice* object; the group's passphrase; if the current device is the group owner.

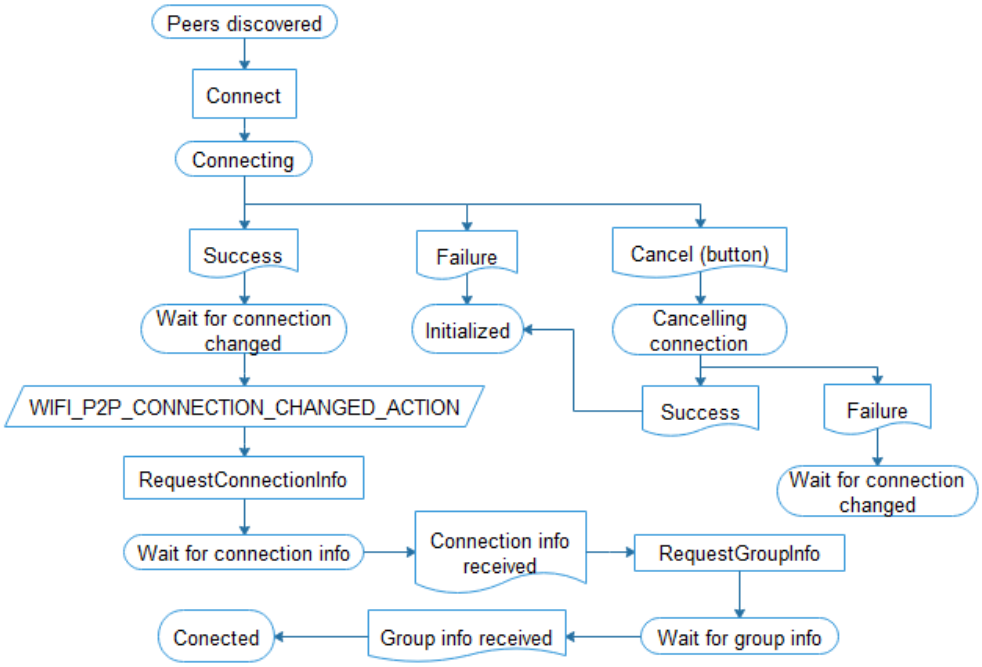

Fig. 7: The connection process

The *createGroup* method causes the current device to create an empty P2P group with itself acting as the group owner. This method is only intended to be used in circumstances where the peer devices are legacy equipment, and will normally not be used in ordinary Wi-Fi Direct operations.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 02, February-2020

In order to perform a disconnection request to a connected group, the *removeGroup* method must be executed. By using the method's callback listener, the application will be able to know whether the request was successful or not. The *removeGroup* method should be able of being triggered in the following states: wait for connection changed, wait for connection info, wait for group info, connected.

- **Intents and intent actions**

An Android Intent is an abstract description of an operation to be performed. To achieve this operation, broadcast receivers with intent filters need to be registered at the application. The following intent actions are relevant for Wi-Fi Direct: *WIFI_P2P_STATE_CHANGED_ACTION, WIFI_P2P_PEERS_CHANGED_ACTION, WIFI_P2P_CONNECTION_CHANGED_ACTION, WIFI_P2P_THIS_DEVICE_CHANGED_ACTION*.

To be able to initiate a discovery process and connect with peer devices, the Wi-Fi Direct mode must be enabled on the current device. This information will be broadcasted by the Android system. By registering a broadcast receiver with an intent filter that contains the *WIFI_P2P_STATE_CHANGED_ACTION* intent action, the application can be informed if the Wi-Fi Direct mode is enabled on the current device. Every time the users turn on or off the Wi-Fi Direct mode, intent will be broadcasted causing the application to always be updated on the Wi-Fi Direct mode's status.

If the *WIFI_P2P_PEERS_CHANGED_ACTION* intent action is added to the filter, the application can be notified when peers are discovered. This will most likely occur after a peer discovery process has been initiated. However, the application will always be ready to receive this type of notification as long as the broadcast receiver is registered.

In order for the application to detect any changes in its Wi-Fi connectivity, the *WIFI_P2P_CONNECTION_CHANGED_ACTION* intent action must be added to the filter. By adding the *WIFI_P2P_THIS_DEVICE_CHANGED_ACTION* intent to the filter, the application will be informed when a change in the device's P2P properties has occurred. This could for instance happen when the devices status changes from being available to be connected. When the intent action is triggered, the user interfaces of the application should be updated in order to give back information of the device's current status to the users.

2. Wi-Fi Direct API in the wpa supplicant

In wpa_supplicant, P2P module API[11] consists of functions for requesting operations and for providing event notifications. Similar set of callback functions are configured with *p2p_config* data structure to provide callback functions that P2P module can use to request operations and to provide event notifications. In addition, there are number of generic helper functions that can be used for P2P related operations.

These are the main functions for an upper layer management entity to request P2P operations:

- *p2p_find():* start P2P find for device discovery phase;
- *p2p_stop_find():* for stopping find in the device discovery phase;
- *p2p_listen():* start Listen state for specified duration. This function can also be used to request the P2P module to keep the device discoverable on the listen channel for an extended set of time;
- *p2p_connect()*: start the Group Owner negotiation for the group formation;
- *p2p_reject()*: explicitly block connection attempts and reject the peer device;
- *p2p_prov_disc_req()*: send provision discovery request;
- *p2p_sd_request()*: schedule a service discovery query;
- *p2p_sd_cancel_request():* cancel a pending service discovery query;
- *p2p_sd_response()*: send response to a service discovery query;
- *p2p_sd_service_update()*: need to be called whenever there is a change in availability of the local services. It indicates a change in local services;
- *p2p_invite()*: invite a P2P device into a group.

These are the main callback functions for P2P module to provide event notifications to the upper layer management entity:

- *p2p_config::dev_found()*: is used to notify that a new P2P device has been found during a search state or listen state;
- *p2p_config::go_neg_req_rx()*: is used to notify that a P2P device is requesting group owner negotiation and notify of a receive Group Owner negotiation request;
- *p2p_config::go_neg_completed()*: notify that Group Owner negotiation has been completed;
- *p2p_config::sd_request()*: indicate the reception of a service discovery request;
- *p2p_config::sd_response()*: indicate the reception of a service discovery response;
- *p2p_config::prov_disc_req()*: indicate the reception of a Provision discovery request frame that the P2P module accepted;
- *p2p_config::prov_disc_resp()*: indicate reception of a provision discovery response frame for a pending request schedule with *prov_disc_req();*
- *p2p_config::invitation_process()*: can be used to implement persistent reconnect by allowing automatic restating for persistent groups without user interaction. It is an optional callback for processing invitation;
- *p2p_config::invitation_received()*: is used to indicate sending of an Invitation response for a received Invitation request;

[11]Defined in /external/wpa_supplicant_8/src/p2p/p2p.hwithin AOSP source tree.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 02, February-2020

- *p2p_config::invitation_result()*: indicate result of an invitation procedure started by calling *p2p_invite().*

The P2P module uses following functions to request lower layer driver operations:

- *p2p_config::p2p_scan()*: request a p2p scan or search operation to be completed;
- *p2p_config::send_probe_resp()*: transmit a probe response frame. It is used to reply to probe request frames that were indicated with a call *to p2p_probe_req_rx();*
- *p2p_config::send_action():* transmit an action frame;
- *p2p_config::send_action_done()*: notify that Action frame sequence was completed;
- *p2p_config::start_listen()*: start listen state. Once the listen state has started, *p2p_listen_cb()* must be called to notify the p2p module
- *p2p_config::stop_listen():* stop listen state. It can be used a listen state operation that was previously requested with *start_listen().*

Events from lower layer driver operations are delivered to the P2P module with following functions:

- *p2p_probe_req_rx()*: report reception of a probe request frame;
- *p2p_rx_action()*: report received action frame;
- *p2p_scan_res_handler()*: indicates a p2p scan results;
- *p2p_scan_res_handled()*: indicate end scan results;
- *p2p_send_action_cb()*: is used to indicate the result of an Action frame transmission that was requested with struct *p2p_config::send_action()* callback*;*
- *p2p_listen_cb()*: indicate the start of a requested Listen state.

In addition to the per-device state, the P2P module maintains per-group state for group owners. This is initialized with a call to *p2p_group_init()* when a group is created and deinitialized with *p2p_group_deinit().* The upper layer GO management entity uses following functions to interact with the P2P per-group state:

- *p2p_group_notif_assoc()*: notification of P2P client association with Group Owner*;*
- *p2p_group_notif_disassoc():* notification of P2P client disassociation from Group Owner;
- *p2p_group_notif_formation_done():* notification of completed group formation;
- *p2p_group_match_dev_type()*: match device types in group with requested type.

In the following we present how these functions are put together to guarantee the functionalities defined in the specification.

- **Device Discovery**

Device Discovery is a mandatory procedure to be supported by all P2P devices. It enables P2P devices to quickly find each other and form a connection. It provided medium access scheme, channels management, time duration on each channel and neighbours discovery. A P2P Device runs the Device Discovery procedure to detect the presence of other P2P Devices to which the connection will be attempted in its wireless range. Two distinct phases are using namely *Scan* and *Find*. Device Discovery uses *Probe Request (PREQ)* and *Probe Response (PRESP)* frames to exchange device information.

Before beginning the *Scan* phase, the global P2P module context is initialized by calling the *p2p_init()* function. This function permits to keep a copy of the configuration data in the *p2p_config* structure. The initialization permit to configure some parameters like the list of supported channels (*p2p_channels* structure), the maximum number of discovered peers to remember (*max_peers*), whether concurrent operations are supported (*concurrent_operations*), the discoverable interval, etc. After the initialization, the P2P Device begins the scan phase. The scan phase may be used to find other P2P Devices or P2P Groups and to locate the best potential Operating Channel to establish a P2P Group. The *p2p_scan()* function is used to request a p2p scan or search operation to be completed. Type arguments specify which type of scan is to be done: *P2P_SCAN_FULL* indicates that all channels are to be scanned; *P2P_SCAN_SOCIAL* indicates that only the social channels should be scanned; *P2P_SCAN_SPECIFIC* request a scan of a single channel that the frequency is known; *P2P_SCAN_SOCIAL_PLUS_ONE* request a scan of all the social channels plus one extra channel that the frequency is known. The full scan type is used for the initial scan because the P2P Device performs traditional Wi-Fi scan through all supported channels in the *p2p_channels* structure defined in IEEE std 802.11 – 2012 Annex J [7] in order to collect information about the surrounding devices. The scan is processing by sending or receiving Probe Request or Probe Response frames; this is done by the using of *p2p_probe_req_rx()* and *send_probe_resp()* functions respectively. The results of the scan will be reported by calling *p2p_scan_res_handler()*, and then calling *p2p_scan_res_handled()* to indicate that all scan results have been indicated and to terminate the process. The following figure (fig.7) presents the scan phase process on Android.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 02, February-2020

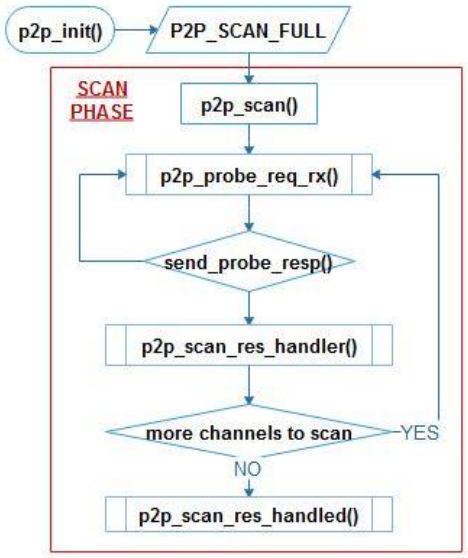

Fig. 8: Scan process

Once the *Scan* phase is completed, the device enters into the *Find* phase. In the *Find* phase, the P2P Device alternates between two states: *Search* and *Listen*. The find phase is initiated by the calling of the *p2p_find()* function, where the device will choose the type of the find: *P2P_FIND_START_WITH_FULL* indicates that the find will be done on all channels present in the p2p_channels structure; *P2P_FIND_ONLY_SOCIAL* indicates that the find will be done only on the social channels (channels 1, 6 and 11 in the 2,4 GHz band) and *P2P_FIND_PROGRESSIVE* indicates that the find will be done progressively in some channels that the frequencies are known. The *p2p_listen_in_find()* function is called to pick a random dwell time for the listen and the *p2p_channel_to_freq()* function permit to choose a listen channel and to convert it in the equivalent frequency. The duration of each Listen phase shall be a random integer of 100 TU intervals. This random number shall be no greater than the *maxDiscoverableInterval* (*max_disc_int*) value and no less than the *minDiscoverableInterval* (*min_disc_int*). On Android the formula to compute the Listen duration is:

$$TU = (r \% ((max\_disc\_int - min\_disc\_int) + 1) + min\_disc\_int) \times 100$$

The defaults values of *max_disc_int* and *min_disc_int* are 3 and 1 respectively; these values are fixed in the initialization phase. The *start_listen()* function initiate the listen with the frequency converted with *p2p_channel_to_freq()* function and ***(1024 x TU) / 1000*** in parameters. In the *Listen* phase (*p2p_listen()* function) the P2P Device may receive Probe Request (*p2p_probe_req_rx()*), send Probe Response (*send_probe_resp()*), receive a service discovery request (*sd_request()*), or send a service discovery response (*p2p_sd_response()*) on the listen channel chosen previously.
P2P Devices in the *Search* phase shall invoke *p2p_search()* function to initiate that phase and shall make a P2P Scan with one of these 3 types of scan: *P2P_SCAN_SPECIFIC* where the device scan only the known listen frequency of the peer during Group Owner Negotiation start, *P2P_SCAN_SOCIAL_PLUS_ONE* where the device only scan the known listen frequency of the peer during Invite start or *P2P_SCAN_SOCIAL* where the device only scan the socials frequencies to discover new devices. In the Search phase, one or more *Probe Request* frames could be transmitted and the device shall not reply to these frames.
Service Discovery is an optional procedure. The procedure starts after the Device Discovery and prior to the Group Formation procedure. It allows a P2P Device to connect to other P2P Devices by sending a service discovery request (*p2p_sd_request()* function) and receiving a service discovery response (*sd_response()* function) only if the latter device offers the intended service. The figure 8 following presents the *Find* phase process on Android.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 02, February-2020

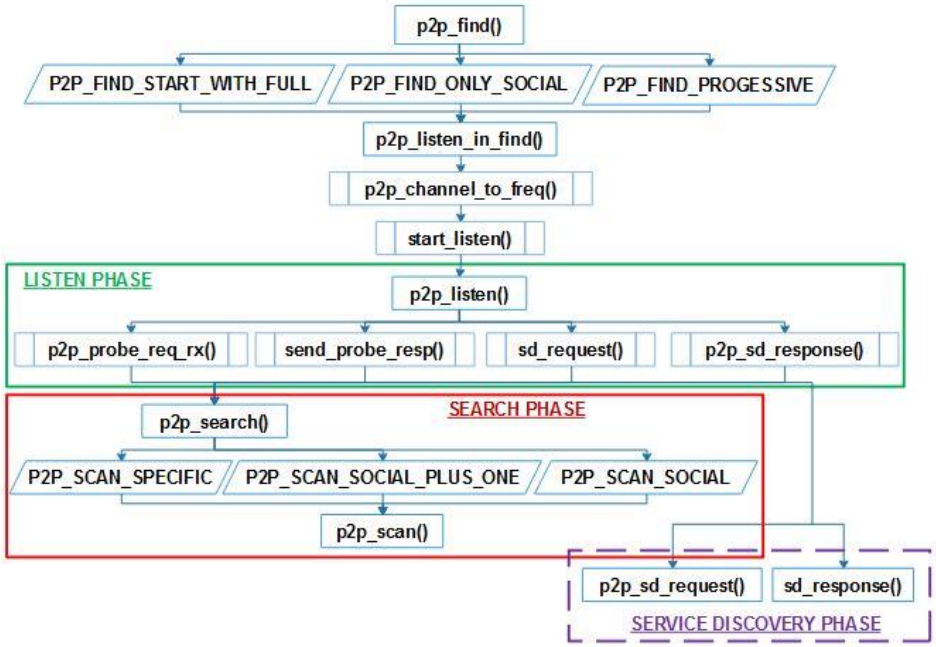

Fig. 9: Find phase process

- **Group Formation**

Following a successful Device Discovery, P2P Devices can establish the P2P Group by using Group Formation Procedures. These procedures provide self organization and topology organization mechanisms to build a group. During the Group Formation, the device that will act as *Group Owner (GO)* is determined in *GO Negotiation* phase in order to define the topology used. The *Invitation* phase permits to invite others devices into the persistent group. On Android, two types of P2P Group Formation schemes are possible: Standard Group Formation and Persistent Group Formation. To start the group formation, the P2P Device invoke the *p2p_connect()* function in which the *p2p_stop_find()* function is executed in order to stop the Find phase. After that, the *p2p_go_neg_start()* function is called to initiate the *Group Owner Negotiation* phase to decide which device will become the owner of the group in case of standard group formation; or the *p2p_invite_start()* function to begin the *Invitation* phase in case of persistent group formation.

The GO Negotiation is a three way handshake and the *p2p_connect_send()* function is used to initiate this handshake. During the handshake, the two devices exchange their intent to become GO by processing to the *GO Negotiation Request* (*p2p_process_go_neg_req()* function) or to the *GO Negotiation Response* (*p2p_process_go_neg_resp()*). In these exchanges, the devices send to each other a randomly chosen numeric value called "*intent value*". The Intent value ranges from 0 to 15, and it measures the desire of the P2P Device to be the P2P GO. The P2P Device sending the higher Intent value shall become GO. In case bath P2P devices send equal GO Intent values, a tie breaker bit is used for decision and the device with tie breaker bit set to 1 shall become GO. This mechanism is achieve in the *p2p_go_det()* function inside the *GO Negotiation Request* or the *GO Negotiation Response* processes. The GO Negotiation Request or Response frames are send using the *go_neg_req_rx()* function. Once the two devices are agree, the *GO Negotiation Confirmation* is process using *p2p_process_go_neg_conf()* function in order to configure the GO, to send back the confirmation and to complete the *GO Negotiation* phase.

In Persistent Group Formation, a P2P Device sends an invitation to another P2P Device in order to instantiate the P2P Group, this is done by using the *p2p_invite_send()* function. The *P2P Invitation Request* (process into the *p2p_process_invitation_req()* function) and the P2P Invitation Response (process into the *p2p_process_invitation_resp()* function) frames are exchanged to establish a persistent group. The following figure presents the *Group Formation* phase process on Android

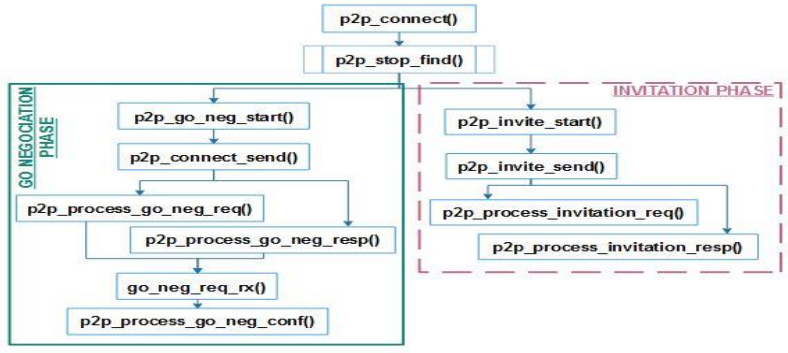

Fig. 10: Group Owner Negotiation and Group formation process

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 02, February-2020

After a successful GO negotiation, the Group Owner runs the DHCP to assign IP addresses to itself as well as to the Peer to Peer clients or legacy clients in its group. The assigned addressing range is defined into the *WifiP2pService* file. These addresses are declared as constants: SERVER_ADDRESS for the server (GO of the group) and DHCP_RANGE is the address range for group clients.

V. WI-FI DIRECT TECHNICAL SPECIFICATION ON ANDROID OS

After the technical study of Wi-Fi Direct on Android, we propose in this section a formal specification. This formal specification will permit to verify if it can really take into account certain MANET needs such as channel access, routing, auto organisation, addressing and scaling. We analyzed the different java classes and C codes mentioned in the previous section to highlight the formal specification using the Z and Object-Z (object-oriented version of Z) languages.

- **Initialization and channel access**

As we explained above, the application needs to do an initialization in the *WifiP2pManager()* class with initialize before doing any p2p operation. In the initialization, a channel that connects the application to the Wi-Fi p2p framework is instanced. It registers the application with the framework. This function initializes the *Asyncchannel* binder and must be the first to be called. The *WifiP2pManager()* class presents a set of operations which are in fact methods of this class. These methods take the context of the *Asyncchannel* binder and the listener as input parameters. The context of the *Asyncchannel* binder (some time called Channel) is obtained after the initialization and the listener listen events on the Channel. Each operation sends a message (with the same name as the operation) to the *WifiP2pService()* which will activate one or more states corresponding to the action. The message is sent to the destination handler by using *sendMessage* method.

Message is the set of message sent to the *WifiP2pService(),MsgFailed* and *MsgSucceeded* are sets of failure and success messages respectively corresponding to each message in the set *Message* and *MsgResponse* is the set of response messages whose corresponding to the response of the request type messages.

$Message == ENABLE\_P2P \mid DISABLE\_P2P \mid DISCOVER\_PEERS \mid CONNECT \mid CANCEL\_CONNECT \mid CREATE\_GROUP \mid REMOVE\_GROUP \mid REQUEST\_PEERS \mid REQUEST\_CONNECTION\_INFO \mid REQUEST\_GROUP\_INFO$

$MsgFailed == ENABLE\_P2P\_FAILED \mid DISABLE\_P2P\_FAILED \mid DISCOVER\_PEERS\_FAILED \mid CONNECT\_FAILED \mid CANCEL\_CONNECT\_FAILED \mid CREATE\_GROUP\_FAILED \mid REMOVE\_GROUP\_FAILED \mid REQUEST\_PEERS\_FAILED \mid REQUEST\_CONNECTION\_INFO\_FAILED \mid REQUEST\_GROUP\_INFO\_FAILED$

$MsgSucceeded == ENABLE\_P2P\_SUCCEEDED \mid DISABLE\_P2P\_SUCCEEDED \mid DISCOVER\_PEERS\_SUCCEEDED \mid CONNECT\_SUCCEEDED \mid CANCEL\_CONNECT\_SUCCEEDED \mid CREATE\_GROUP\_SUCCEEDED \mid REMOVE\_GROUP\_SUCCEEDED \mid REQUEST\_PEERS\_SUCCEEDED \mid REQUEST\_CONNECTION\_INFO\_SUCCEEDED \mid REQUEST\_GROUP\_INFO\_SUCCEEDED$

$MsgResponse == RESPONSE\_PEERS \mid RESPONSE\_CONNECTION\_INFO \mid RESPONSE\_GROUP\_INFO$

$GlobalMessage = Message \cup MsgFailed \cup MsgSucceeded \cup MsgResponse$

A set of failure reason: $FailureReason == ERROR \mid P2P\_UNSUPORTED \mid BUSY$

[*Context*] and [*Looper*] are sets of context and looper respectively while [*ChannelListener*], [*ActionListener*], [*PeerListListener*], [*ConnectionInfoListener*] and [*GroupInfoListener*] are sets of listeners obtain after performing respectively *channelListener*, *actionListener*, *peerListListener*, *connectionInfoListener* and *groupInfoListener* methods. The following schema present the initialize method, it return the channel which is the context of the *AsyncChannel* binder whose looper correspond to not null *ChannelListener* provide by the channelListener method.

Initialize

$\Delta[Context]$

$Cl? : ChannelListener$

$Looper? : Looper$

$ChlLooper : [Looper] \rightarrow [ChannelListener]$

$AsyncChannelState = STATUS\_SUCCESFUL$

$C! = cont : Context \bullet (Cl? \neq NULL) \wedge (Looper? \mapsto Cl?) \in ChlLooper$

$C' = C!$

The success of this initialization makes it possible to launch *p2p_init()* function which allows to configure the device by initializing the variables of the *p2p_config* structure in the wpa_supplicant. The schema following present some parameters of the *p2p_config* structure:

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 02, February-2020

P2p_config

Channel?, *channel_forced*,*op_channel*: *u*8

Channels?: [*p2p_channels*]

Pref_chan?: [*p2p_channel**]

Max_peers?: ℕ*

Max_listen?: ℕ*

The *channel* and the *op_channel* parameters are respectively the own listen channel and the own operational channel of the device, while the *channel_forced* is the listen channel was forced by configuration or by control interface and cannot be overridden.*Channels* is the own supported regulatory classes and channels of the device; it is the list of supported channels per regulatory class.*Pref_chan* is the preferred channels for GO Negotiation. *Max_peers* is the number of discovered peers to remember by the device. If there are peers to discover, older entries will be removed to make room for the new ones and *max_listen* is the maximum listen duration in milliseconds. Once initialized, the parameters of the channels will be converted into the corresponding frequency (in MHz) by using *p2p_channel_to_freq()*function; this frequency will be used as input in other functions such as: *p2p_find()*, *start_listen()*, *p2p_search()*, etc.

The channel method checks the status of the previously initialized channel and defines the messages to be returned to the application after the response of the *wifiP2pService()*.

ChannelInit

Cl? : *ChannelListener*, *Al*? : *ActionListener*, *Pll*? : *PeerListListener*

Cil? : *ConnectionInfoListener*, *Gil*? : *GroupInfoListener*

RcvF : [*Message*] ↣ [*MsgFailed*], *RcvS* : [*Message*] ↣ [*MsgSucceeded*]

RcvRep : [*Message*] ↣ [*MsgResponse*]

AsyncChannelState ≠*CMD_CHANNEL_DISCONNECTED*

Cl? ≠*NULL*

Failed

Al? ≠*NULL*

∃ *x* : *MsgFailed* •((*m*? ↦ *x*) ∈*RcvF*) ∧ (*m* ∈dom*RcvF*)

Al′ = *actionListener.onFailure*

Success

Al? ≠*NULL*

∃ *y* : *MsgSucceeded* •((*m*? ↦*y*) ∈*RcvS*) ∧ (*m* ∈dom*RcvS*)

Al′ = *actionListener.success*

Peers

Rep?: *MsgResponse*

Pll? ≠*NULL*

Rep? = *RESPONSE_PEERS*

((*m*? ↦*Rep*?) ∈*RcvRep*) ∧ (*m*? ∈dom*RcvRep*)

Pll′ = *peerListListener.onPeersAvailable*

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 02, February-2020

ConnectionInfo

Rep?: *MsgResponse*

Cil? ≠*NULL*

Rep? = *RESPONSE_CONNECTION_INFO*

((*m*? ↦*Rep*?) ∈*RcvRep*) ∧ (*m*? ∈ dom*RcvRep*)

Cil′ = *connectionInfoListener.onConnectionInfoAvailable*

GroupInfo

Rep?: *MsgResponse*

Gil? ≠*NULL*

Rep? = *RESPONSE_GROUP_INFO*

((*m*? ↦*Rep*?) ∈*RcvRep*) ∧ (*m*? ∈*domRcvRep*)

Gil′ = *groupInfoListener.onGroupInfoAvailable*

Channel≙ChannelInit∧ (Failed∨Success∨Peers∨ConnectionInfo∨GroupInfo)

The operations performed are represented by the following schemas.

OperationInit

C?: *Context*; *M*?: *Message*

Al?: *ActionListener*; *Pll*?: *PeerListListener*

Cil?: *ConnectionInfoListener*; *Gil*?: *GroupInfoListener*

C? ≠*NULL*

EnableP2p

(*M*? = *ENABLE_P2P*)⇒ *sendMessage*(*M*?)

DisableP2p

(*M*? = *DISABLE_P2P*)⇒ *sendMessage*(*M*?)

DiscoverPeers

(*M*? = *DISCOVER_PEERS*)⇒ *sendMessage*(*M*?, *Al*?)

Connect

Conf?: *Config*

Conf? ≠*NULL*

(*M*? = *CONNECT*) ⇒*sendMessage*(*M*?, *Al*?, *Conf*?)

CancelConnect

(*M*? = *CANCEL_CONNECT*)⇒ *sendMessage*(*M*?, *Al*?)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 02, February-2020

CreateGroup

(*M*? = *CREATE_GROUP*)⇒ *sendMessage*(*M*?, *Al*?)

RemoveGroup

(*M*? = *REMOVE_GROUP*)⇒ *sendMessage*(*M*?, *Al*?)

RequestPeers

(*M*? = *REQUEST_PEERS*)⇒ *sendMessage*(*M*?, *Pll*?)

RequestConnectionInfo

(*M*? = *REQUEST_CONNECTION_INFO*)⇒ *sendMessage*(*M*?, *Cil*?)

RequestGroupInfo

(*M*? = *REQUEST_GROUP_INFO*)⇒ *sendMessage*(*M*?, *Gil*?)

Then ***Operation≙OperationInit∧ (EnableP2p ∨ DisableP2p ∨ DiscoverPeers ∨ Connect ∨ CancelConnect ∨ CreateGroup ∨ RemoveGroup ∨ RequestPeers ∨ RequestConnectionInfo ∨ RequestGroupInfo)***
The schema of *INIT* initializes the parameters used in the *WifiP2pManager()* class to provide theirs operations.

INIT

M?: *Message*, *Rang* = = {1, 4, 7, 10, 13, 16, 19, 22, 24, 26}
BASE?, *BASE_WIFI_P2P_MANAGER*? : ℕ
Mrang: [*Message*] ↦*Rang*

BASE? = *BASE_WIFI_P2P_MANAGER*
(*M*? ↦1) ∈ *Mrang* ⇒(((*M*? = = *ENABLE_P2P*) ∧ (*M*? = *BASE* + 1)) ∧((*M*? ↦*x*) ∈*RcvF*
⇒ ((*x* = =*ENABLE_P2P_FAILED*) ∧ (*x* = *BASE* + 2))) ∧ ((*M*? ↦*y*) ∈*RcvS* ⇒((*y* = = *ENABLE*
_P2P_SUCCEEDED) ∧ (*y* = *BASE* + 3))))
(*M*? ↦4) ∈ *Mrang* ⇒(((*M*? = = *DISABLE_P2P*) ∧ (*M*? = *BASE* + 4)) ∧((*M*? ↦*x*) ∈*RcvF*
⇒ ((*x* = =*DISABLE_P2P_FAILED*) ∧ (*x* = *BASE* + 5))) ∧ ((*M*? ↦*y*) ∈*RcvS* ⇒((*y* = = *DISABLE*
_P2P_SUCCEEDED) ∧ (*y* = *BASE* + 6))))
(*M*? ↦7) ∈ *Mrang* ⇒(((*M*? = = *DISCOVER_PEERS*) ∧ (*M*? = *BASE* + 7)) ∧((*M*? ↦*x*) ∈*RcvF*
⇒ ((*x* = =*DISCOVER_PEERS_FAILED*) ∧ (*x* = *BASE* + 8))) ∧ ((*M*? ↦*y*) ∈*RcvS* ⇒((*y* = = *DISCOVER*
_PEERS_SUCCEEDED) ∧ (*y* = *BASE* + 9))))
(*M*? ↦10) ∈ *Mrang* ⇒(((*M*? = = *CONNECT*) ∧ (*M*? = *BASE* + 10)) ∧((*M*? ↦*x*) ∈*RcvF*
⇒ ((*x* = =*CONNECT_FAILED*) ∧ (*x* = *BASE* + 11))) ∧ ((*M*? ↦*y*) ∈*RcvS* ⇒((*y* = = *CONNECT*
_SUCCEEDED) ∧ (*y* = *BASE* + 12))))
(*M*? ↦13) ∈ *Mrang* ⇒(((*M*? = = *CANCEL_CONNECT*) ∧ (*M*? = *BASE* + 13)) ∧((*M*? ↦*x*) ∈*RcvF*
⇒ ((*x* = =*CANCEL_CONNECT_FAILED*) ∧ (*x* = *BASE* + 14))) ∧ ((*M*? ↦*y*) ∈*RcvS* ⇒((*y* = = *CANCEL*
_CONNECT_SUCCEEDED) ∧ (*y* = *BASE* + 15))))
(*M*? ↦16) ∈ *Mrang* ⇒(((*M*? = = *CREATE_GROUP*) ∧ (*M*? = *BASE* + 16)) ∧((*M*? ↦*x*) ∈*RcvF*
⇒ ((*x* = =*CREATE_GROUP_FAILED*) ∧ (*x* = *BASE* + 17))) ∧ ((*M*? ↦*y*) ∈*RcvS* ⇒((*y* = = *CREATE*
_GROUP_SUCCEEDED) ∧ (*y* = *BASE* + 18))))
(*M*? ↦19) ∈ *Mrang* ⇒(((*M*? = = *REMOVE_GROUP*) ∧ (*M*? = *BASE* + 19)) ∧((*M*? ↦*x*) ∈*RcvF*
⇒ ((*x* = =*REMOVE_GROUP_FAILED*) ∧ (*x* = *BASE* + 20))) ∧ ((*M*? ↦*y*) ∈*RcvS* ⇒((*y* = = *REMOVE*
_GROUP_SUCCEEDED) ∧ (*y* = *BASE* + 21))))
(*M*? ↦22) ∈*Mrang* ⇒(((*M*? = = *REQUEST_PEERS*) ∧ (*M*? = *BASE* + 22)) ∧ (*M* ? ↦*x*) ∈*RcvRep*
⇒((*x* = = *RESPONSE_REQUEST_PEERS*) ∧ (*x* = *BASE* + 23)))
(*M*? ↦24) ∈*Mrang* ⇒(((*M*? = = *REQUEST_CONNECTION_INFO*) ∧ (*M*? = *BASE* + 24)) ∧
(*M* ? ↦*x*) ∈*RcvRep* ⇒((*x* = = *RESPONSE_CONNECTION_INFO*) ∧ (*x* = *BASE* + 25)))
(*M*? ↦26) ∈*Mrang* ⇒(((*M*? = = *REQUEST_GROUP_INFO*) ∧ (*M*? = *BASE* + 26)) ∧ (*M* ? ↦*x*) ∈*RcvRep*
⇒((*x* = = *RESPONSE_GROUP_INFO*) ∧ (*x* = *BASE* + 27)))

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 02, February-2020

Finally the specification of *WifiP2pManager* is given by:

$$WifiP2pManager \triangleq INIT \wedge Initialize \wedge Channel \wedge Operation$$

- **Auto organization and group created state**

Applications communicate with *WifiP2pService* to issues device and connectivity requests through the *WifiP2pManager* interface. The state machine communicates with the Wi-Fi driver through *wpa_supplicant* and handles the event responses through *wifiMonitor*. The methods of this java class are the different states of the state machine. Each state implements three main methods: *enter()* to activate the state, *processMessage()* that processes actions related to the reception of either message from the *WifiP2pManager* through the binder or event from *wpa_supplicant* through the *WifiMonitor* and *exit()* to close and exit to the state. The states implemented in the p2p state machine are presented in figure 4. We present here the formal specification of the *groupCreatedState* method which corresponds to the GroupCreatedState state.
We define *WMEvent*, a set of events sent by the *WifiMonitor* and *MsgWFManager*, a set of message sent par *WifiP2pManager*.
WMEvent = = AP_STA_CONNECTED_EVENT | AP_STA_DISCONNECTED_EVENT | P2P_GROUP_REMOVED_EVENT | P2P_DEVICE_LOST_EVENT | P2P_INVITATION_RESULT_EVENT | P2P_PROV_DISC_PBC_REQ_EVENT | P2P_PROV_DISC_ENTER_PIN_EVENT | P2P_GROUP_STARTED_EVENT
MsgWFManager = = CONNECT | REMOVE_GROUP; $MsgWFManager \subset Message$
The following schema present the *enter()* method for the GroupCreatedState state.

Enter

$STATUS : \mathbb{N}$
$Role == GO \mid Client$
$DevRole : [Device] \rightarrowtail Role$
$D? : Device;$

$D?.STATUS = CONNECTED$
$(D? \mapsto r) \in DevRole \wedge (r = GO)$
$D?.Addr = SERVER\_ADDRESS$
$Rep! = sendP2pConnectionChangedBroadcast$

In the GroupCreatedState state, any element of *WMEvent* or *MsgWFManager* received, messages are processed according to the events.
If a device connects in the group, which corresponds to the receipt of the AP_STA_CONNECTED_EVENT event sent by the *WifiMonitor*.It should be noted that a new device can connect to the group only if the number of peers currently present in the group is still less than the*max_peers*of the GO.

ConnectedEvent

$Wfe? : WMEvent$

$Wfe? = AP\_STA\_CONNECTED\_EVENT$
$D?.Addr \neq NULL$
$D?.STATTUS = CONNECTED$
$D? \notin Group \wedge DevRole(D?) \neq GO$
$Group' = Group \cup \{D?\}$
$\#Group' = \#Group + 1$
$Rep? = sendP2pPeersChangedBroadcast$

Or if a device leaves the group, which corresponds to the receipt of the AP_STA_DISCONNECTED_EVENT event, the schema is the following.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 02, February-2020

DisconnectEvent

Wfe?: *WMEvent*
[*iface*]; *p2pGroupRemove* : [*iface*] ↣{*T*, *F*}
If? : *iface*

Wfe? = *AP_STA_DISCONNECTED_EVENT*
D?.*Addr* ≠ *NULL*
D?.*STATUS* = *AVAILABLE*
D? ∈ *Group* ∧ *DevRole*(*D*?) ≠*GO*
Group′ = *Group* \{*D*?}
#*Group*′ = #*Group* – 1
(*Group*′ = ∅) ⇒ (*D*?.*If*?↦*T*) ∈*p2pGroupRemove*

DeviceLostEvent

Wfe? = *p2p_DEVICE_LOST_EVENT*
D?.*Addr* ≠ *NULL*
DevRole(*D*?) = *GO* ∧ *D*? ∈*Group*
Group \{*D*?} ⇒ (*D*?.*If* ↦ *T*)

If the CONNECT message is received from the *WifiP2pManager*, we have the two following schemas

ConnectT

Wpm?: *MsgWFManager*
P2pInvite : [*Device*] ↣ {*T*, *F*}

Wpm? = *CONNECT*
D?.*Addr* ≠ *NULL*
(*P2pInvite*(*D*?) ≠*F*) ⇒*D*?.*STATUS* = *INVITED*
Rep! = *sendP2pPeersChangedBroadcast*
wifiP2pManager = *CONNECT_SUCCEEDED*

ConnectF

(*P2pInvite*(*D*?) = *F*) ⇒*wifiP2pManager* = *CONNECT_FAILFED*

And *Connect*≙*ConnectT*∨*ConnectF*
If the group is destroyed:

RemoveGroup

If?: *Iface*

Wpm? = *REMOVE_GROUP*
P2pGroupRemove (*If*?) = *F*
wifiP2pManager = *REMOVE_GROUP_SUCCEEDED*

So we have the following schema

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 02, February-2020

ProcessMessage≜ConnectEvent∨DisconnectEvent∨Connect∨RemoveGroup

The *exit()* method of the state is represent by:

Exit

STATUS:ℕ
Role = = *GO* |*Client*
DevRole: [*Device*] ↣*Role*
D?: *Device*

D?.*STATUS* = *AVAILABLE*
D ? ∈*Group*
(*DevRole*(*D* ?) = *GO*) ⇒ (*D* ?.*Addr* =*Null*)
Rep! = *sendP2pConnectionChangedBroadcast*

Then ***groupCreatedState ≜ Enter ∨ ProcessMessage ∨ Exit***

- **Addressing**

To allow communications in the group, each device must have an address. Wi-Fi Direct uses class C IPv4 addresses in the 192.168.49.x, x ∈ [1, 254]; with the mask on 3 bytes. The addressing of devices in the group is as follows:

GOAddressing

Ξ*Device*
X!:ℕ
D?: *Device*
DevRole: [*Device*]↣*Role*

(*D*?∈*Group*)∧ (*DevRole* (*D*?) = *GO*)
X! = 1

ClientAddressing

Ξ*Device*
CA: ℙℕ
X!:*CA*
D?: *Device*
DevRole: [*Device*] ↣*Role*

(*D*? ∈*Group*) ∧ (*DevRole* (*D*?) = *Client*)
X′∈ *CA*
X′≥2
X′≤254)
X! = *X′*

The in the group ***Addressing ≜ GOAddressing ∧ ClientAddressing***

VI. DISCUSSION

The detailed study of the Wi-Fi Implementation on Android and the formal specification reveal that

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 02, February-2020

Wi-Fi Direct is an interesting and suitable candidate technology for ad hoc communication in several applications, but his implementation on the Android OS has some limits on several levels relatively to the formation of a large scale Ad hoc network:

- **In terms of connection and association**: the scan, listen, and search processes may consume considerable amount of time in Wi-Fi Direct. Two devices attempting a peer discovery can share probe messages only if a search phase and a listen phase is met on the same channel. However, these phases are randomized; as each listen phase of a device can vary from a random multiplier of 100 time units. Therefore, two devices could attempt multiple discoveries before actually synchronizing their channels, which could consume time in scales of seconds. Moreover, this delay can be relatively high if several P2P Devices are simultaneously performing Device Discovery in the same wireless range. Another limit comes from access to the communication channel: it is impossible for a device to have several communication channels at the same time since the *p2p_init()* function just permit to reserve a specific random channel (a frequency) on which the p2p group formation process will be initiated. So each p2p group can only work on one communication channel. However only the devices supporting concurrent operations can also be connected to a BSS of Wi-Fi while connected to the P2P Group. A device supports concurrent operations when the *concurrent_operations* parameter was initialized during the configuration.
- **At the topology and mobility**: In Android, upper-layer applications can specify a Group Owner Intent; otherwise the Wi-Fi Direct framework simply sets it with a random value. The Group Owner election mode based on the Intent value is not optimal enough, since based on a single criterion. It can happen that after GO Negotiation, equipment having reduced performances becomes Group Owner at the expense of another which has higher performances. Once the group is created, all communication goes through the Group Owner. Clients cannot communicate directly without pass to the GO. So the GO acts as a router within the group, but he cannot communicate simultaneously with Clients that are belonging in the group. Moreover, the number of devices that can be associated in a P2P Group depends on the maximum number of discovered peers the GO can remember, which limits the scale of the group and if the GO is leaving the group, the group is destroyed making the GO the central equipment of the group.
- **At the addressing and routing**: In Android devices, once a Wi-Fi Direct connection is established, the Group Owner will always have 192.168.49.1 IP address and Clients in the group will have different addresses (192.168.49.x /24, where x is a random number ∈ [2, 254]) according to GO assignments. This method of assigning addresses can cause conflicts in case the Group Owner is part of several groups.
  In fact the Peer to Peer interfaces of all Group Owners have the same IP address, namely 192.168.49.1. The Wi-Fi interfaces of the Group Owners that act as legacy clients in another group are assigned an IP address in the format 192.168.49.x/24. Similarly, Peer to Peer interfaces of clients are assigned different IP addresses in the format 192.168.49.x/24. This provides the address conflicts for Peer to Peer interface of the Group Owners. Thus even if there is a way to allow multi-group membership, the devices in different groups may not be able to reach each other especially since no routing mechanism is defined.

VII. CONCLUSION

Wi-Fi Direct is one of the promising technologies to establish MANET among mobile devices because of its high popularity in Android OS, high data rate and long communication range. The protocol has also the potential to be used in several applications such as files transferring, sharing resources, online gaming, alert dissemination, social networking, etc. Most of research work is focused on the performance evaluation Wi-Fi Direct. Providing efficient group formation techniques and supporting multi-hop communication are part of the key research issues highlighted in the literature. In this paper, we presented a technical overview of Wi-Fi Direct technology on Android in order to well understand its implementation. We took out the layer structure of the Wi-Fi Direct framework on Android by presenting the different classes and their interactions in each layer. The main components of the Wi-Fi Direct framework on Android are the *WifiP2pManager()* class found in the SDK and the wpa_supplicant package in the HAL. We showed how the *WifiP2pManager()* class works for discovery and connection to peers. We have explained and presented the group formation and group owner election mechanisms in the wpa_supplicant. We provide also the formal specification of some classes by using Z and Object-Z specification languages. Which leads us raise the limits of Wi-Fi Direct implementation on Android in terms of the development of large-scale networks. These limits are of several orders: in the connection and association, in topology and mobility and in the addressing and routing. To implement new solutions on Android with Wi-Fi Direct, we can make changes in the *WifiP2pManager()*and *WifiP2pService()* classes and in some functions of the wpa_supplicant; e.g. *p2p_connect(),p2p_invite()* for MANETs functionalities or *p2p_find(), p2p_listen()* for implementing new device discovery mechanisms.

REFERENCES

[1] M. Asif Khan, W. Cherif, F. Filali, and R. Hamila, "Wi-Fi Direct Research – Current status and Future perspectives," *Journal of Network and Computer Applications*, Vol. 93, pp 245-258, 2017.
[2] D. Camps-Mur, A. Garcia-Saavedra, and P. Serrano, "Device-to-Device Communications with Wi-Fi Direct: Overview and Experimentation,"*IEEE Wireless Communications*, vol. 20, no. 3, pp.96-104, 2013.
[3] M. Conti, F. Delmastro, G. Minutiello, and R. Paris, "Experimenting opportunistic networks with Wi-Fi Direct," *Wireless Days (WD), 2013 IFIP*, pp. 1–6, 2013.
[4] H. Je, D. Kwon, H. Kim, and H. Ju, "Mobile Network Configuration for Large-scale Multimedia delivery on a single WLAN," 2014.
[5] R. Kanaoka and Y. Tobe, "Design of Data Transfer System on Smartphones using Wi-Fi Direct and Accelometers," *IEEE 3rd Global Conference on Consumer Electronics*, GCCE 2014, pp.71-75, 2014.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 02, February-2020

[6] "Wi-fi peer-to-peer (p2p) technical specification V1.7", *WiFi Alliance, Tech. Rep*., 2016.
[7] IEEE 802.11-2012 IEEE Standard for Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications, Mar. 2012.
[8] W. Sun, C. Yang, S. Jin, and S. Choi, "Listen Channel Randomization for Faster Wi-Fi Direct Device Discovery," *The 35th Annual IEEE International Conference on Computer Communications*. IEEE INFOCOM. 2016.
[9] J. Han, K. W. Lim, and Y. B. Ko, "Fast Location-based Association of Wi-Fi Direct for Distributed Wireless Docking Services," *International Conference on Green and Human Information Technology* (ICGHIT), 2014.
[10] J. Feng, Z. Liu, and Y. Ji, "Wireless Channel Loss Analysis – A Case Study Using Wi-Fi Direct," *International Wireless Communications and Mobile Computing Conference* (IWCMC), 2014.
[11] H. Yoo, S.Kim, S. Lee, J. Y. Hwang, and D. Kim, "Traffic-aware Parameter tuning for Wi-Fi Direct Power Saving," *International Conference on Ubiquitous and Future Networks* (ICUFN), 2014.
[12] K. W. Lim, Y.Seo, Y. B. Ko, J. Kim, and J. Lee, "Dynamic Power Management in Wi-Fi Direct for Future Wireless Serial Bus," *Wireless Networks*, vol.20, no.7, pp.1777-1793, 2014.
[13] K. W. Lim, W. S. Jung, and Y. B. Ko, "Energy Efficient Quality-of-Service for WLAN-Based D2D Communications," *Ad Hoc Networks*, vol. 25, pp.102-116, 2015.
[14] D. Camps-Mur, X. Pérez-Costa, and S. Sallent-Ribes, "Designing energy efficient access points with Wi-Fi Direct," *Computer Networks*, vol.55, no. 13, pp 2838-2855, 2011.
[15] H. Zhang, Y. Wang, and C. C. Tan, "Wd2: An improved wi-fi-direct group formation protocol," in *Proceedings of the 9th ACM MobiCom workshop on Challenged networks*. ACM, 2014, pp. 55–60.
[16] U. Botrel Menegato, L. Souza Cimino, S. E. Delabrida Silva, F. A. Medeiros Silva, J. Castro Lima, and R. A. R. Oliveira, "Dynamic clustering in wifi direct technology," in *Proceedings of the 12th ACM international symposium on Mobility management and wireless access*. ACM, 2014, pp. 25–29.
[17] A. A. Shahin, and M. Younis, "Efficient multi-group formation and communication protocol for wi-fi direct," *in Local Computer Networks (LCN), 2015 IEEE 40th Conference on. IEEE*, 2015, pp. 233–236.
[18] V. Arnaboldi, M. G. Campana, and F. Delmastro, "Context-Aware Configuration and Management of WiFi Direct Groups for Real Opportunistic Networks," *IEEE 14th International Conference on Mobile Ad hoc and Sensors Systems* (MASS), 2017.
[19] A. Laha, X. Cao, W. Shen, X. Tian, and Y. Cheng, "An Energy Efficient Routing Protocol for Device-to-Device Based Multihop Smartphone Networks," *IEEE International Conference on Communications* (ICC), 2015.
[20] M. A. Khan, W. Cherif, and F. Filali, "Group Owner Election in Wi-Fi Direct," *IEEE 7th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference* (UEMCON), 2016.
[21] A. A Shahin, and M. Younis, "A Framework for P2P Networking of Smart Devices Using Wi-Fi Direct," *IEEE 25th International Symposium on Personal, Indoor and Mobile Radio Communications*, 2014.
[22] W. Cherif, M. A. Khan, F. Felali, S. Sharafedine, and Z. Dawy, "P2P Group Formation Enhancement for Opportunistic Networks with Wi-Fi Direct," *IEEE Wireless Communications and Networking Conference* (WCNC), 2017.
[23] I. M. Lombera, L. E. Moser, P. M. Melliar-Smith, and Y. T. Chuang, "Peer Management for iTrust over Wi-Fi Direct," *International Symposium on Wireless Personal Multimedia Communications*, 2013.
[24] J. E. Park, J. Park, and M. J. Lee, "DirectSpace: A Collaborative Framework for Supporting Group Workspaces over Wi-Fi Direct," *4th International Conference on Mobile, Ubiquitous, and Intelligent Computing* (MUSIC), 2013.
[25] Y. Duan, et al. "Wi-Fi Direct Multi-group Data Dissemination for public Safety," *World Telecommunications Congress* (WTC), 2014.
[26] C. Casetti, C. F. Chiasserini, L. C. Pelle, C. Del Valle, Y. Duan, and P. Giaccone, "Content-centric routing in wi-fi direct multi-group networks," in *2015 IEEE 16th International Symposium on a World of Wireless, Mobile and Multimedia Networks* (WoWMoM). IEEE, 2015, pp. 1-9.
[27] Z. Wang, F. Li, X.Wang, T. Li, and T. Hong, "A Wi-Fi Direct Based Local Communication System," IEEE IWQoS, 2018.
[28] K. Liu, W. Shen, B. Yin, X. Cao, L. X. Cai, and Y. Cheng, "Development of Mobile Ad-hoc with Off-the-Shelf Android Phones," *Ad-hoc and Sensor Networking Symposium*, IEEE ICC, 2016.
[29] J. Loo, J. L. Mauri, and J. H. Ortiz "Mobile Ad Hoc Networks: Current Status and Future Trends" *CRC* Press, 2011.
[30] L. Van Hoang and H. Ogawa, "A platform for building ad hoc social networks based on Wi-Fi Direct," *IEEE 3rd Glob. Conf. Consum. Electron.*, pp. 626-629, 2014.
[31] Downloading Android source tree: https://source.android.com/setup/build/downloading. Visited on 17 December 2018.
[32] B. Zores, "Dive Into Android Networking: Adding Ethernet Connectivity," *ABS*, 2013 available on line https://speakerdeck.com/gxben/dive-into-android-networking-adding-ethernet-connectivity-1. Visited on 15 December 2018.
[33] Android Platform Architecture: https://developer.android.com/guide/platform/ visited on 17 December 2018.
[34] Android API reference: https://developer.android.com/reference/ visited on 12 December 2018.
[35] Wi-Fi Direct innovation: http://www.wi-fi.org/discover-wi-fi/wi-fi-direct, visited on 29 November 2018.
[36] S. Ranakarthik and C Zang, "Generating Java Skeletel code with Design Contracts from Specifications in a Subset of Object-Z"*International Conference on Computer and Information Science (ACIS)* 2006.
[37] A. F. Al Azzawi, M. Bettaz, and H. M. Al-Refai, "Generating Python Code from Object-Z Specifications" *International Journal of Software Engineering & Applications (IJSEA)*, vol8. N°4. July 2017.
[38] M. Najafi and H. Haghighi, "An Approach to animate Object-Z Specifications using C++", *Scientia Iranica*, April 2012.