A
Dissertation
On

Hibernation Support for Wi-Fi Direct

Submitted in Partial Fulfillment of the Requirement

For the Award of the Degree of

Master of Technology

in

Software Technology

by

Baba Abinash
University Roll No. 2K11/SWT/05

Under the Esteemed Guidance of

Mr. R.K. Yadav
Assistant Professor, Computer Engineering Department, DTU

2011-2014

COMPUTER ENGINEERING DEPARTMENT

DELHI TECHNOLOGICAL UNIVERSITY

DELHI – 110042, INDIA

ABSTRACT

Wi-Fi Direct is defined by the Wi-Fi Alliance in order to enable efficient device-to-device Communication. Portable devices are a main target of this technology and hence, power saving is a key objective. Research presented in this paper enhances the power management schemes of Wi-Fi Direct to provide better services for a wider range of Wi-Fi Peer-to-peer (P2P) applications.

Lower power consumption means lower heat dissipation, which increases system stability, and less energy use, which saves money and reduces the impact on the environment. For mobile device and embedded system device, it's much more important because the battery power is very limited. Nowadays, android phone and iPhone are more and more pervasive. There are more and more sensors and I/O in mobile device that can be used to improve the effectiveness of PM.

Although the Wi-Fi Direct defines two power management schemes to provide energy efficiency in P2P devices, neither is enough to meet power efficiency and reliability requirements for various services.

To alleviate this problem, the proposed scheme recognizes the properties of applications and Implement Hibernate Mode in Wi-Fi Direct to analyse its effectiveness.

ACKNOWLEDGEMENT

I take this opportunity to express my deepest gratitude and appreciation to all those who have helped me directly or indirectly towards the successful completion of this thesis.

Foremost, I would like to express my sincere gratitude to my guide **Mr. R.K. Yadav, Assistant Professor, Department of Computer Engineering, Delhi Technological University**, Delhi whose benevolent guidance, constant support, encouragement and valuable suggestions throughout the course of my work helped me successfully complete this thesis. Without his continuous support and interest, this thesis would not have been the same as presented here.

Besides my guide, I would like to thank the entire teaching and non-teaching staff in the Department of Computer Engineering, DTU for all their help during my course of work.

Baba Abinash
University Roll no: 2K11/SWT/05
M.Tech (Software Technology)
Department of Computer Engineering
Delhi Technological University
Delhi – 11004

Computer Engineering Department
Delhi Technological University
Delhi-110042
www.dce.edu

CERTIFICATE

This is to certify that the thesis entitled "**Hibernation Support for Wifi Direct** " submitted by **Baba Abinash (Roll Number: 2K11/SWT/05),** in partial fulfillment of the requirements for the award of degree of Master of Technology in Software Technology, is an authentic work carried out by her under my guidance. The content embodied in this thesis has not been submitted by her earlier to any institution or organization for any degree or diploma to the best of my knowledge and belief.

Date: _ _ ___

(Mr. R.K. Yadav)
Assistant Professor & Project Guide
Department of Computer Engineering
Delhi Technological University

Table of Contents

CHAPTER 1

INTRODUCTION

Recent advances in mobile communication have opened opportunities for exciting new mobile services. Applications utilizing Peer-to-Peer (P2P) communications are among them, which enables more sophisticated methods of data exchange, visual conference, gaming, and social media sharing through these mobile devices. One very promising technology for supporting future P2P services is Wi-Fi Direct, which enables the popular IEEE 802.11 WLAN technology to provide reliable and high-rate wireless data transmission.

Various networking chip vendors have already announced their Wi-Fi Direct chipsets, and recently applications have been developed in the Android operating system, being introduced in the markets for usage in Smartphone's .Following the recent advancement of Wi-Fi Direct and mobile computing devices, the issue of energy preservation have also dramatically risen. Quite differently to traditional networks, power management schemes in future mobile devices cannot solely account for energy efficiency as the utmost goal to achieve. This is because the future of Wi-Fi Direct considers not only the reliability of current applications (Transmission of files and documents, data sharing), but also support of quality of service (multimedia data) and better user experience (games, Human Interface Device: HID peripherals) for potential future applications. To achieve these goals, research on power saving exclusively for mobile P2P communications has become more important than ever. Current power management modes in Wi-Fi Direct are Opportunistic Power save mode and Notice of Absence (NoA) mode. Although either of these protocols can be utilized for better energy efficiency depending on the application or environment, they do not provide enough energy efficiency. Therefore, the power saving modes in Wi-Fi Direct need to be enhanced to cope with various services and applications that has different properties.

The proposed scheme scheme recognizes the properties of applications and Implement Hibernate Mode in Wi-Fi Direct to analyse its effectiveness is evaluated through simulator to Prove its superiority over existing Wi-Fi Direct power saving schemes.

CHAPTER 2

Wi-Fi Direct

2.1. Introduction

Direct device to device connectivity was already possible in the original IEEE 802.11 standard by means of the ad-hoc mode of operation. However this never became widely deployed in the market and hence presents several drawbacks when facing nowadays requirements, e.g. lack of efficient power saving support

Wi-Fi direct takes a different approach to enhance device to device connectivity. Instade of Ad-hoc mode of operation Wi-Fi direct build upon the successful IEEE 802.11 infrastructure mode. Wi-Fi Direct is an open source implementation which we have used to enhance performance of this technology in realistic scenarios.

2.2. A Technical Overview

In a typical Wi-Fi network, client scans and associate to wireless networks available, which are created and announced by Access Points (AP). Each of these devices has roles involving a different set of functionality. A major novelty of Wi-Fi Direct is that these roles are specified as dynamic, and hence a Wi- Fi Direct device has to implement both the role of a client and the role of an AP (sometimes referred to as Soft AP). These roles are therefore logical roles that could even be executed simultaneously by the same device, this type of operation is called Concurrent mode.

In order to establish a communication, P2P devices have to agree on the role that each device will assume at the time of negotiation. In the following we describe how this communication is configured using specified procedures, namely device discovery, role negotiation, service discovery, security provisioning and power saving.

2.3. Architecture

Wi-Fi Direct devices, formally known as P2P Devices, communicate by establishing P2P Groups, which are functionally equivalent to traditional Wi-Fi infrastructure networks. The device implementing AP like functionality in the P2P Group is referred to as the P2P Group Owner (P2P GO), and devices acting as clients are known as P2P Clients.

This GO and client functionality is dynamic and is negotiated at the time of initial network setup. Two P2P devices discover each other; they negotiate their roles (P2P Client and P2P GO) to establish a P2P Group. Once the P2P Group is established, other P2P Clients can join the group as in a traditional Wi-Fi network. Legacy clients can also communicate with the P2P GO, as long as they support the required security mechanisms. By default Wi-Fi Direct uses WPA2PSK as security standard. In this way, legacy devices do not formally belong to the P2P Group and do not support the enhanced functionalities defined in Wi-Fi Direct, but they simply "see" the P2P GO as a traditional AP.

The logical nature of the P2P roles supports different architectural deployments; one of this is illustrated in below Figure 2.1 represents a scenario with two P2P groups. The first scenario is a mobile phone sharing its 3G connection with two laptops; in this first scenario, the three devices form a group, the phone is acting as P2P GO while the two laptops behave as P2P Clients. In order to extend the network, one of the laptops establishes a second P2P Group with a printer; for this second group, the laptop acts as P2P GO. In order to act both as P2P Client and as P2P GO the laptop will typically alternate between the two roles by time-sharing the Wi-Fi interface.

Like a traditional AP, a P2P GO announces itself through beacons containing additional P2P Information Element. P2P IE is included in all management frames. Legacy devices ignore these information elements and action frames. The Wi-Fi Direct Specification requires that the P2P device which becomes the group owner should also provide the DHCP server application in their system [3] to provide P2P Clients with IP addresses. In addition, only the P2P GO is allowed to cross-connect the devices in its P2P Group to an external network. Finally, Wi-Fi Direct does not allow transferring the role of P2P GO within a P2P Group. In this way, if the P2P GO leaves

the P2P Group then the group is torn down, and has to be re-established using some of the specified procedures.

2.4. Group Formation

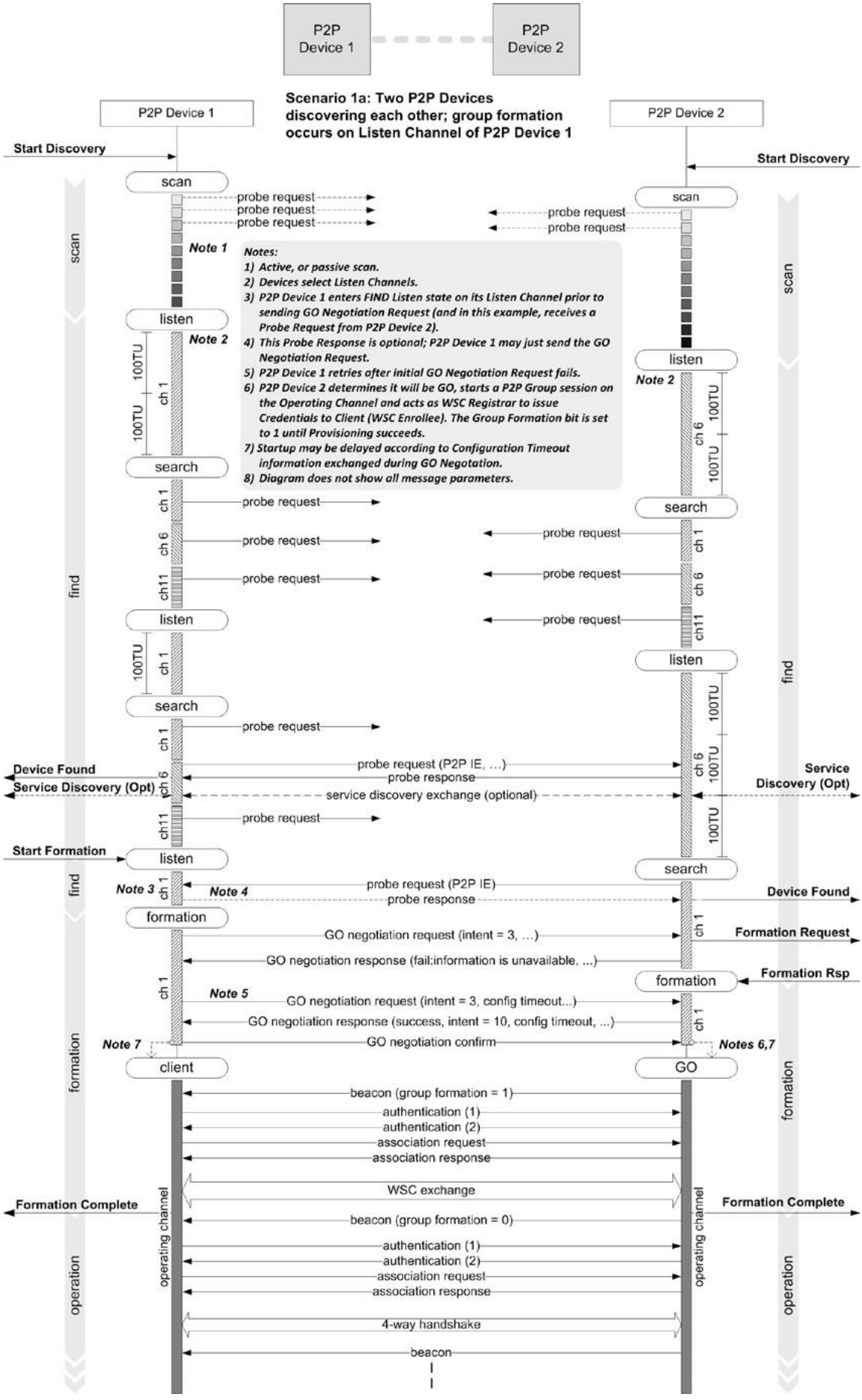
There are several ways in which two devices can establish a P2P Group. Three types of group formation techniques are Standard, Autonomous and Persistent cases. Group Formation procedure involves two phases-

1) Determination of P2P Group owner

Negotiated - Two P2P devices negotiate for P2P group owner based on desire/capabilities to be a P2P GO. Selected - P2P group Owner role established at formation or at an application level.

2) Provisioning of P2P Group

Establishment of P2P group session using appropriate credentials Using Wi-Fi simple configuration to exchange credentials.


P2P Device 1
P2P Device 2
Scenario 1a: Two P2P Devices discovering each other; group formation occurs on Listen Channel of P2P Device 1
P2P Device 1
P2P Device 2
Start Discovery
Start Discovery
scan
scan
probe request
probe request
probe request
probe request
probe request
Note 1
Notes:
1) Active, or passive scan.
2) Devices select Listen Channels.
3) P2P Device 1 enters FIND Listen state on its Listen Channel prior to sending GO Negotiation Request (and in this example, receives a Probe Request from P2P Device 2).
4) This Probe Response is optional; P2P Device 1 may just send the GO Negotiation Request.
5) P2P Device 1 retries after initial GO Negotiation Request fails.
6) P2P Device 2 determines it will be GO, starts a P2P Group session on the Operating Channel and acts as WSC Registrar to issue Credentials to Client (WSC Enrollee). The Group Formation bit is set to 1 until Provisioning succeeds.
7) Startup may be delayed according to Configuration Timeout information exchanged during GO Negotation.
8) Diagram does not show all message parameters.
listen
Note 2
100TU
ch 1
100TU
search
ch 1
ch 6
ch11
probe request
probe request
probe request
probe request
probe request
probe request
listen
100TU
ch 1
search
probe request
Device Found
Service Discovery (Opt)
probe request (P2P IE, ...)
probe response
service discovery exchange (optional)
Service Discovery (Opt)
probe request
Start Formation
listen
search
Note 3
Note 4
probe request (P2P IE)
probe response
Device Found
formation
GO negotiation request (intent = 3, ...)
Formation Request
GO negotiation response (fail:information is unavailable, ...)
Formation Rsp
Note 5
GO negotiation request (intent = 3, config timeout...)
GO negotiation response (success, intent = 10, config timeout, ...)
Note 7
GO negotiation confirm
Notes 6,7
client
GO
beacon (group formation = 1)
authentication (1)
authentication (2)
association request
association response
WSC exchange
Formation Complete
Formation Complete
beacon (group formation = 0)
authentication (1)
authentication (2)
association request
association response
4-way handshake
beacon
scan
find
formation
operation
operating channel

Standard: In this case the P2P devices have first to discover each other, and then negotiate which device will act as P2P GO. Wi-Fi Direct devices usually start by performing traditional Wi-Fi scan (active or passive), by means of which they can discover existent P2P Groups and Wi-Fi networks. After this scan, a new Discovery algorithm is executed. First, a P2P Device selects one of the Social channels, namely channels 1, 6 or 11 in the 2.4 GHz band, as its Listen channel. Then, it alternates between two states: a search state, in which the device performs active scanning by sending Probe Requests in each of the social channels; and a listen state, in which the device listens for Probe Requests in its listen channel to respond with Probe Responses. Once the two P2P Devices have found each other, they start the GO Negotiation phase. This is implemented using a three-way handshake, namely GO Negotiation Request/ Response/ Confirmation, where by the two devices agree on which device will act as P2P GO and on the channel where the group will operate, which can be in the 2.4 GHz or 5GHz bands. In order to agree on the device that will act as P2P GO, P2P devices send a numerical parameter, the GO Intent value, within the three-way hand-shake, and the device declaring the highest value becomes the P2P GO.

To prevent conflicts when two devices declare the same GO Intent, a tie-breaker bit is included in the GO Negotiation Request, which is randomly set every time a GO Negotiation Request is sent.

Persistent: During the formation process, P2P devices can declare a group as persistent, by using a flag in the P2P Capabilities attribute present in Beacon frames, Probe Responses and GO negotiation frames. In this way, the devices forming the group store network credentials and the assigned P2P GO and Client roles for subsequent re-instantiations of the P2P group. Specifically, after the Discovery phase, if a P2P Device recognizes to have formed a persistent group with the corresponding peer in the past, any of the two P2P devices can use the Invitation Procedure (a two-way handshake) to quickly re-instantiate the group. This is shown in Figure 2.2, where the Standard case is assumed as baseline, and the GO Negotiation phase is replaced by the invitation exchange, and the WPS Provisioning phase is significantly reduced because the stored network credentials can be reused.

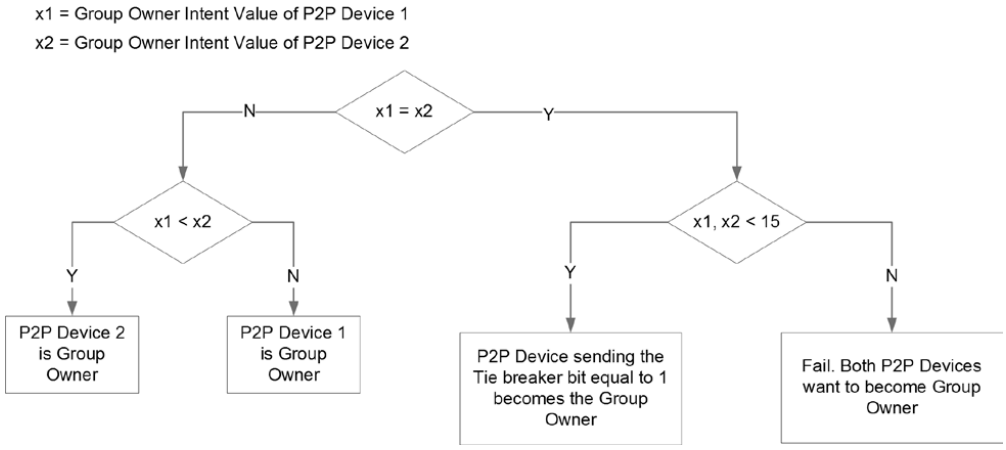


Figure 2.2 Go Negotiation Flow Diagram

P2P Invitation procedure: The P2P Invitation Procedure is an optional procedure used for the following:

A P2P Group Owner invites a P2P Device to become a P2P Client in its P2P Group.

A P2P Client inviting another P2P Device to join the P2P Group of which the P2P Client is a member.

Requesting to invoke a Persistent P2P Group for which both P2P Devices have previously been provisioned and one of the Devices is P2P Group Owner for the Persistent P2P Group.

A P2P Device that is invited to join an operational P2P Group through successful completion of the P2P Invitation Procedure, Use Wi-Fi Simple Configuration to obtain Credentials. Provision Discovery and Wi-Fi Simple Configuration will take place on the Operating Channel of the P2P Group Owner.

P2P Invitation Request: A P2P Invitation Request frame may be transmitted by:

1. A P2P Device that is a member of a P2P Group (i.e. P2P Group Owner or P2P Client) to another P2P Device that supports P2P Invitation Procedure and is currently not a member of the P2P Group to invite that P2P Device to join the P2P Group. When used for this purpose, the invitation Type in the Invitation Flags attribute in the P2P Invitation Request frame set to 0.

2. A P2P Device that is a member of a Persistent P2P Group to another member of that P2P Group and one of the Devices is the P2P Group Owner, to request that the P2P Group be invoked. When used for this purpose, the Invitation Type in the Invitation Flags attribute included in the P2P Invitation Request frame shall be set to 1.

P2P Invitation Response: A P2P Invitation Response frame (with the Status attribute set to Success) transmitted by the P2P Group Owner of a Persistent P2P Group in response to a request to invoke that P2P Group, include the P2P Group BSSID, Channel List, Operating Channel and Configuration Timeout attributes to indicate the Group BSSID, potential Operating Channels, intended Operating Channel and any GO Configuration Time.

2.5. Service Discovery

A salient feature of Wi-Fi Direct is the ability to support service discovery at the link layer. In this way, prior to the establishment of a P2P Group, P2P Devices can exchange queries to discover the set of available services and, based on this, decide whether to continue the group formation or not. Generic Advertisement Protocol (GAS) specified by 802.11u [5]. GAS is a layer two query /response protocols implemented through the use of public action frames, that allows two non-associated 802.11 devices to exchange queries belonging to a higher layer protocol (e.g. a service discovery protocol). GAS is implemented by means of a generic container that provides fragmentation and reassembly, and allows the recipient device to identify the higher layer protocol being transported. GAS is used as a container for ANQP (Access Network Query Protocol) elements sent between clients and APs

WFD State Machine

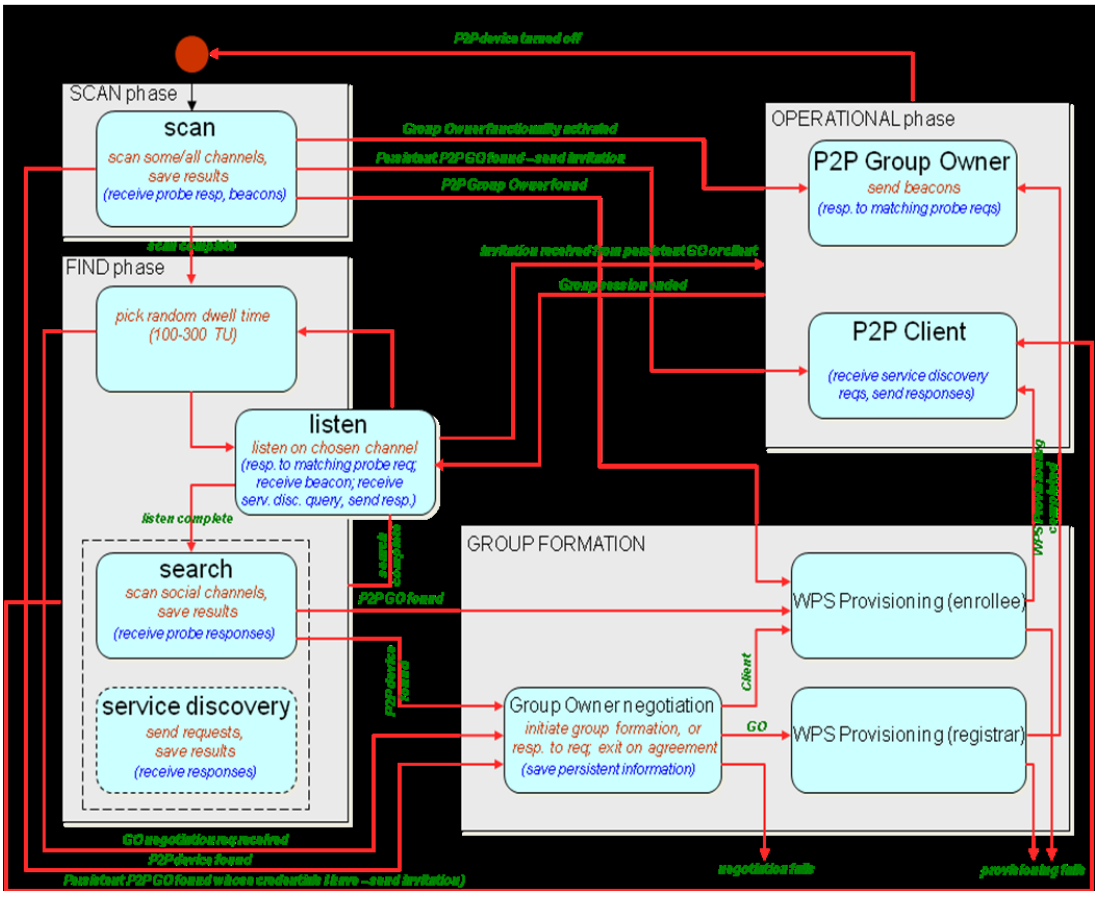


Figure 2.3 Wfd State machine

2.6. Security

Security provisioning starts after discovery has taken place and, if required, the respective roles have been negotiated. Wi-Fi Direct devices are required to implement Wi-Fi Protected Setup (WPS) to support a secure connection with minimal user intervention. In particular, WPS allows establishing a secure connection by introducing a PIN in the P2P Client, or pushing a button in the two P2P Devices. Following WPS terminology, the P2P GO is required to implement an internal Registrar, and the P2P Client is required to implement an Enrollee. The operation of WPS is composed of two parts. In the first part, the internal Registrar is in charge of generating and issuing the network credentials, i.e., security keys, to the Enrollee. WPS is based on WPA-2 security and uses Advanced Encryption Standard (AES)-CCMP as cipher, and a randomly generated Pre-Shared Key (PSK) for mutual authentication. In the second part, the Enrollee (P2P Client) disassociates and

reconnects using its new authentication credentials. In this way, if two devices already have the required network credentials (this is the case in the Persistent group formation) , there is no need to trigger the first phase , and they can directly perform the authentication .

2.7. Power Saving

Wi-Fi Direct application, in battery-constrained devices may typically act as P2P GO (soft-AP), and therefore energy efficiency is importance. However, power saving mechanisms in current Wi-Fi networks is not defined for APs but only for clients. To support energy savings for the AP, Wi-Fi Direct defines two new power saving mechanisms: the Opportunistic Power Save protocol and the Notice of Absence (NoA) protocol.

1) Opportunistic Power Save: The basic idea of Opportunistic Power Save is to leverage the sleeping periods of P2P Clients. The mechanism assumes the existence of a legacy power saving protocol, and works as follows. The P2P GO advertises a time window, denoted as CTWindow, within each Beacon and Probe Response frames. This window specifies the minimum amount of time after the reception of a Beacon during which the P2P GO will stay awake and therefore P2P Clients in power saving can send their frames. If after the CTWindow the P2P GO determines that all connected clients are in doze state, either because they announced a switch to that state by sending a frame with the Power Management (PM) bit set to 1, or because they were already in the doze state during the previous beacon interval, the P2P GO can enter sleep mode until the next Beacon is scheduled; otherwise, if a P2P Client leaves the power saving mode (which is announced by sending a frame with the PM bit set to 0) the P2P GO is forced to stay awake until all P2P Clients return to power saving mode.

Notice that, using this mechanism, a P2P GO does not have the final decision on whether to switch to sleep mode or not, as this depends on the activity of the associated P2P Clients. To give a P2P GO higher control on its own energy consumption Wi-Fi Direct specifies the Notice of Absence protocol, which is described next.

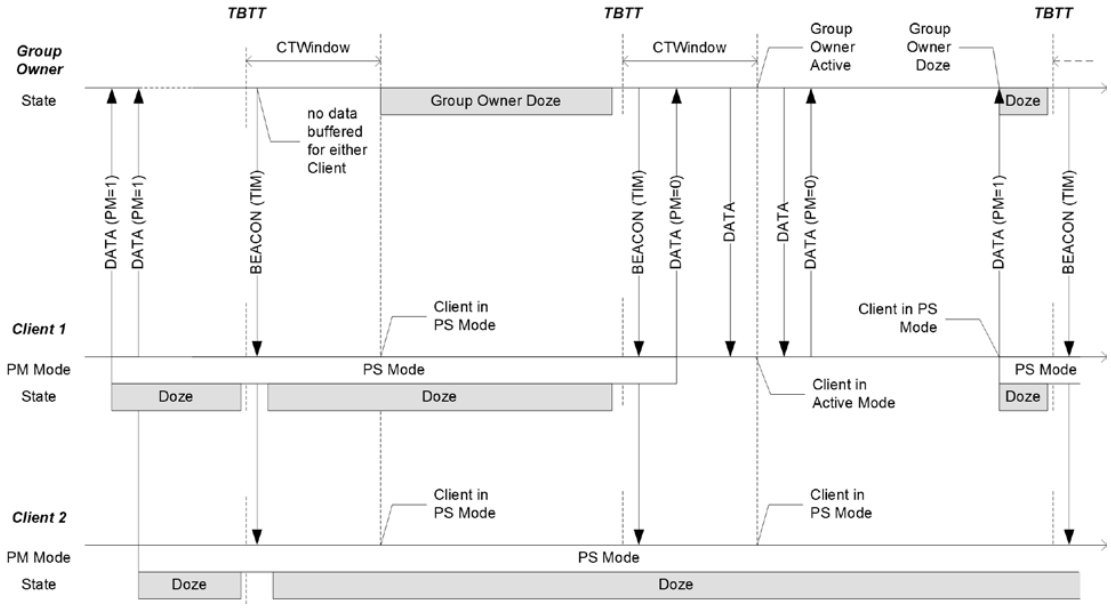


Figure 2.4 Opportunistic powersave Opeartion

2) Notice of Absence: The Notice of Absence (NoA) protocol allows a P2P GO to announce time intervals, referred to as absence periods, where P2P Clients are not allowed to access the channel, regardless of whether they are in power save or in active mode. In this way, a P2P GO can autonomously decide to power down its radio to save energy.

Like in the Opportunistic Power Save protocol, in the case of NoA the P2P GO defines absence periods with a signaling element included in Beacon frames and Probe Responses. In particular, a P2P GO defines a NoA schedule using four parameters: (I) duration that specifies the length of each absence period, (II) interval that specifies the time between consecutive absence periods, (III) start time that specifies the start time of the first absence period after the current Beacon frame, and (IV) count that specifies

how many absence periods will be scheduled during the current NoA schedule. A P2P GO can either cancel or update the current NoA schedule at any time by respectively omitting or modifying the signaling element. P2P Clients always adhere to the most recently received NoA schedule.

In order to foster vendor differentiation, the Wi-Fi Direct specification does not define any mechanism to compute the CTWindow in the Opportunistic Power Save protocol or the schedule of absence periods in the Notice of Absence protocol. In Section III we provide some experimental results comparing the impact of different power saving policies that can be used to configure the NoA protocol.

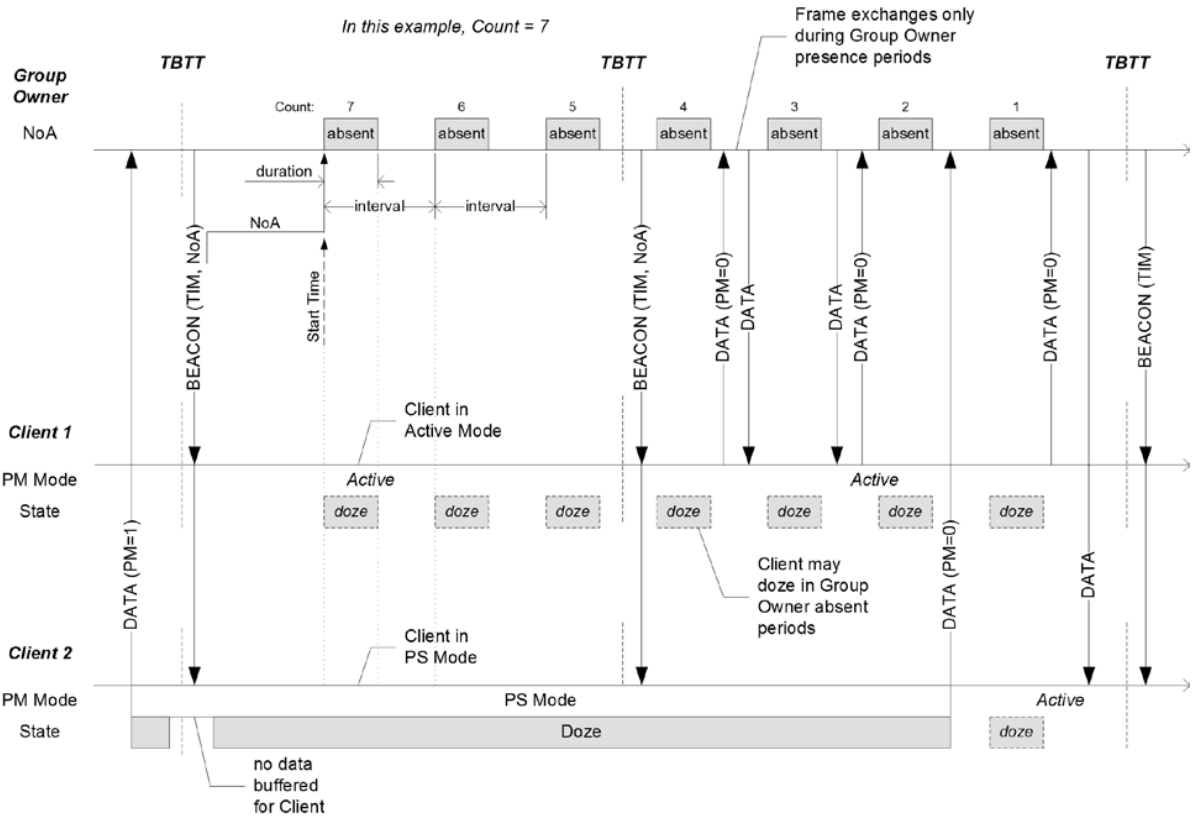


Figure 2.5 Notice of Absence Powersave Opeartion

2.8. Packet Analysis

This section describes packet details of Wi-Fi Direct protocol.

Wi-Fi Direct IE: The format of the P2P IE is shown figure 2.6. The P2P attributes are defined to have a common general format consisting of a 1 octet P2P Attribute ID field, a 2 octet Length field and variable-length attribute-specific information fields, shown in figure 2.7.

A P2P Device that encounters an unknown or reserved Attribute ID value in a P2P IE received without error shall ignore that P2P attribute and parse any remaining fields for additional P2P attributes with recognizable Attribute ID values. A P2P Device that encounters a recognizable but unexpected Attribute ID value in the received P2P IE may ignore that P2P attribute. More than one P2P IE may be included in a single frame. If multiple P2P IEs are present, the complete P2P attribute data consists of the concatenation of the P2P Attribute fields of the P2P IEs. The P2P Attributes field of each P2P IE may be any length up to the maximum (251 octets).

Field	Size (Octet)	Value (Hexadecimal)	Description
Element ID	1	0xDD	IEEE 802.11 vendor specific usage.
Length	1	Variable	Length of the following fields in the IE in octets. The length field is a variable and set to 4 plus the total length of P2P attributes.
OUI	3	50 6F 9A	WFA specific OUI
OUI Type	1	0x09 (to be assigned)	Identifying the type or version of P2P IE. Setting to 0x09 indicates WFA P2P v1.0.
P2P Attributes	Variable		One of more P2P attributes appear in the P2P IE.

Figure 2.6 P2p IE Format

Field	Size (Octet)	Value (Hexadecimal)	Description
Attribute ID	1	Variable	Identifying the type or version of P2P attribute.
Length	2	Variable	Length of the following fields in the attribute.
Attribute body field	Variable		Attribute specific information fields.

Figure 2.7 General Format of P2P attributes

The P2P Capability attribute contains a set of parameters that can be used to establish a P2P connection. The format of the P2P Capability attribute is shown in figure 2.8

Field	Size (Octet)	Value	Description
Attribute ID	1	2	Identifying the type of P2P attribute.
Length	2	2	Length of the following fields in the attribute.
Device Compatibility Bitmap	1	Variable	A set of parameters indicating P2P Device's capabilites.
Group Compatibility Bitmap	1	Variable	A set of parameters indicating the current state of a P2P Group.

Figure 2.8 P2P Capability attribute format

The Public Action frame format (as defined in IEEE 802.11k) is used to define the P2P public action frames. The general format of the P2P public action frames is shown in figure 2.9

Field	Size (Octet)	Value (Hexadecimal)	Description
Category	1	0x04	IEEE 802.11 public action usage
Action field	1	0x09	IEEE 802.11 vendor specific usage
OUI	3	50 6F 9A	WFA specific OUI
OUI Type	1	0x09 (to be assigned)	Identifying the type of version of action frame. Setting to 09 indicates WFA P2P v1.0
OUI Subtype	1		Identifying the type of P2P public action frame.
Dialogue Token	1		Set to non zero value to identify the request/ response transaction.
Elements	variable		Including P2P IE or any other information elements defined in IEEE std. 802.11-2007

Figure 2.9 General Format of P2P Public Action Frame

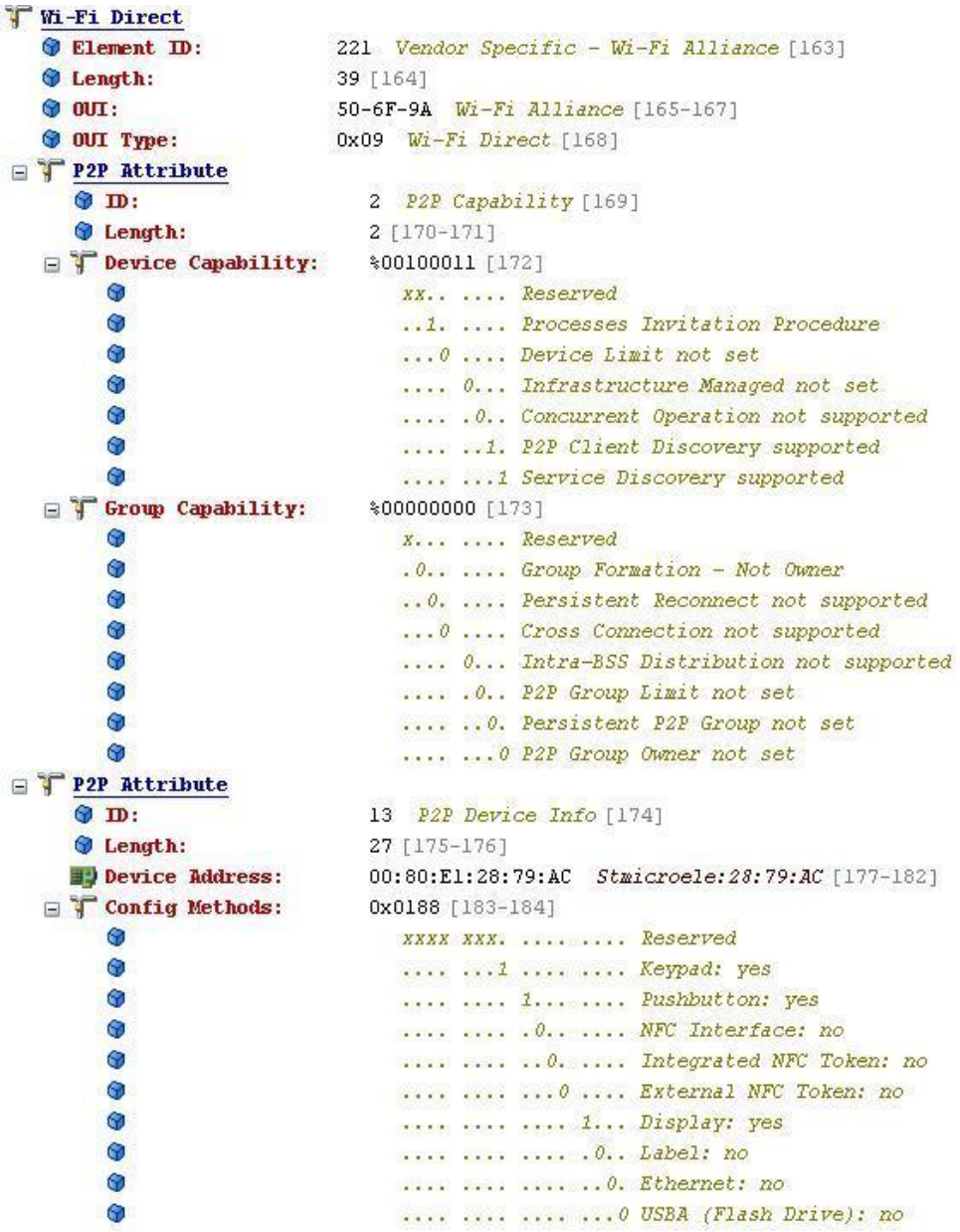


Figure 2.10 P2P Information Element shown from a captured Packet

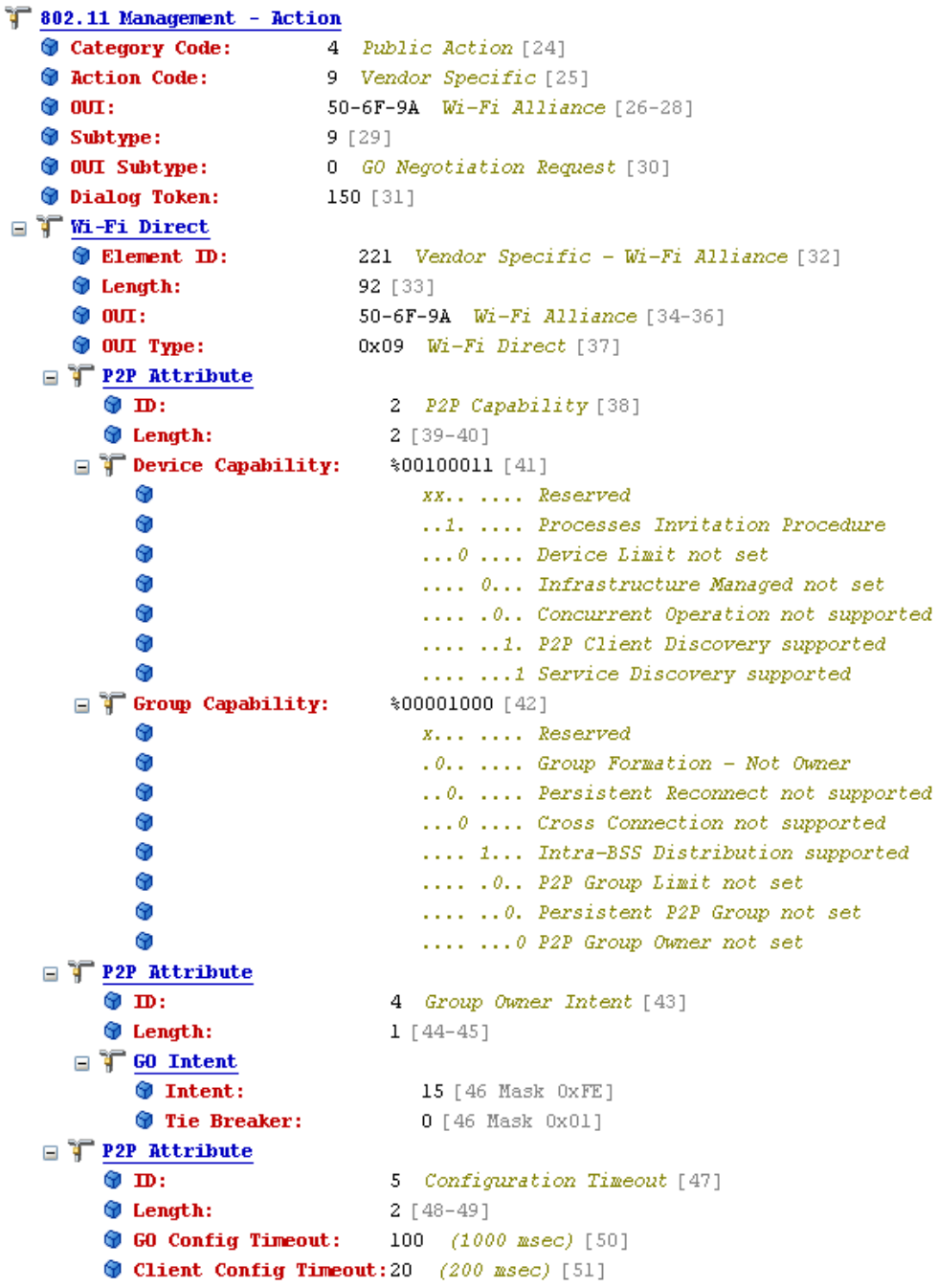


Figure 2.11 P2P Packet Captured with detail view

CHAPTER 3

Android power management

3.1. How does power management system work?

One power management standard for computers is ACPI, which supersedes APM. All recent (consumer) computers have ACPI support. Why ACPI has more advantage than APM? We'll write a brief introduction both of them and compare the difference.

APM (Advanced Power Management)

APM consists of one or more layers of software that support power management in computers with power manageable hardware. APM defines the hardware independent software interface between hardware-specific power management software and an operating system power management policy driver. It masks the details of the hardware, allowing higher-level software to use APM without any knowledge of the hardware interface.

The APM software interface specification defines a layered cooperative environment in which applications, operating systems, device drivers and the APM BIOS work together to reduce power consumption. In brief, APM can extend the life of system batteries and thereby increases productivity and system availability.

ACPI (Advanced Configuration & Power Interface)

The ACPI specification was developed to establish industry common interfaces enabling robust operating system (OS)-directed motherboard device configuration and power management of both devices and entire systems. Different from APM, ACPI allows control of power management from within the operating system. The previous industry standard for power management, APM, is controlled at the BIOS level. APM

is activated when the system becomes idle. The longer the system idles, the less power it consumes (e.g. screen saver vs. sleep vs. suspend). In APM, the operating system has no knowledge of when the system will change power states.

There are several software components that ACPI has:
A subsystem which controls hardware states and functions that may have previously been in the BIOS configuration
These states include:
Thermal control
Motherboard configuration
Power states (sleep, suspend)
a policy manager, which is software that sits on top of the operating system and allows user input on the system policies

The ACPI also has device drivers those control/monitor devices such as a laptop battery, SMBus (communication/transmission path) and EC (embedded controller).

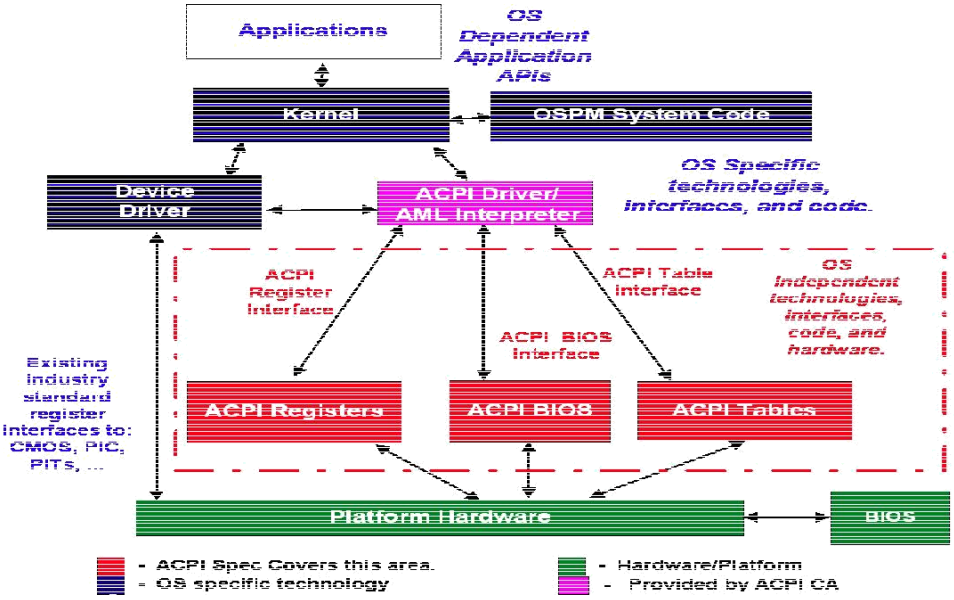


Figure 3.1 CPI architecture

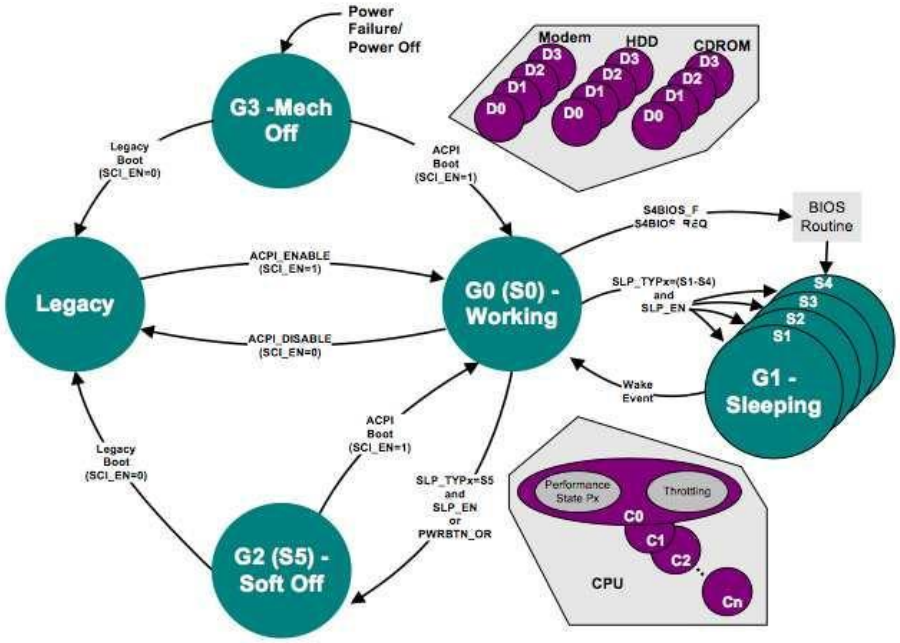


Figure 3 .2 CPI power state transition diagram

3.2. Android architecture

First of all, Android OS design is based on Linux kernel. Linux has its own power management that we have described in previous section. The following diagram (Figure 3.3) shows the main components of the Android OS.

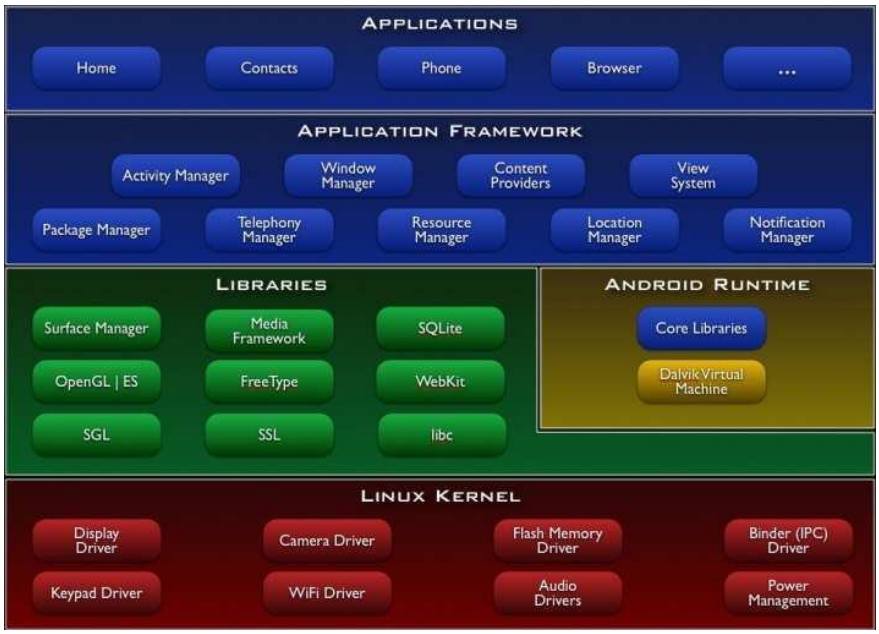


Figure 3.3 Android architecture

Android inherits many kernel components from Linux including power management component. Original power management of Linux is designed for personal computers, so there are some power saving status such as suspend and hibernation. However, these mechanisms of Linux PM do not satisfied and suitable for mobile devices or embedded systems. Mobile devices such as cell phones are not as same as PCs that have unlimited power supply. Because mobile devices have a hard constraint of limited battery power capacity, they need a special power management mechanism. Therefore, Android has an additional methodology for power saving.

3.3. Power Management

The implementation of Android power management was sitting on top of Linux Power Management. Nevertheless, Android has a more aggressive Power Management policy than Linux, in which app and services must request CPU resource with "wake locks" through the Android application framework and native Linux libraries in order to keep power on, otherwise, Android will shut down the CPU.

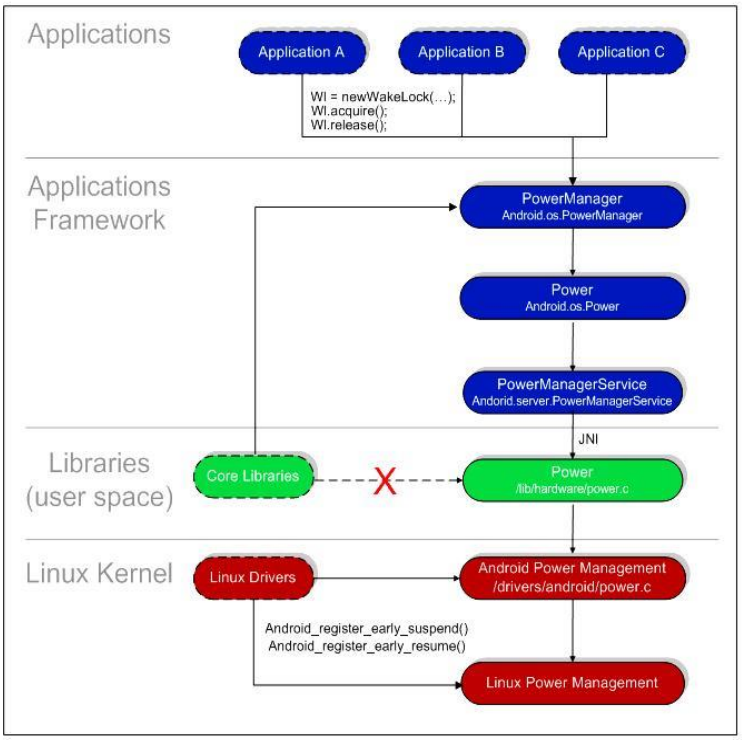


Figure 3.4 Android Power Management.

Refer to Figure 3.4, Android try not to modify the Linux kernel and it implements an applications framework on top of the kernel called Android Power Management Application s Framework. The Android PM Framework is like a driver. It is written by Java which connects to Android power driver through JNI. However, what is JNI? JNI (Java Native Interface) is a framework that allows Java code running in a Java Virtual Machine (JVM) to

call native C applications and libraries. Through JNI, the PM framework written by Java can call function from libraries written by C.

Android PM has a simple and aggressive mechanism called “Wake locks”. The PM supports several types of “Wake locks” . Applications and components need to get “Wake locks” to keep CPU on. If there is no active wake locks, CP U will turn off. Android supports different types of “Wake locks” (Table 3.1).

Table 3.1 Different wake locks of Android PM.

Wake Lock Type
ACQUIRE_CAUSES_WAKE
FULL_WAKE_LOCK
ON_AFTER_RELEASE
PARTIAL_WAKE_LOCK
SCREEN_BRIGHT_WAKE_LOCK
SCREEN_DIM_WAKE_LOCK

Currently Android only supports screen, keyboard, buttons backlight, and the brightness of screen. Because of full usage of CPU capability, it does not support suspend and standby mode. The following diagram shows how Android PM works. Through the framework, user space applications can use “Power Manger” class to control the power state of the device. We will introduce more details about how to implement them in applications later.

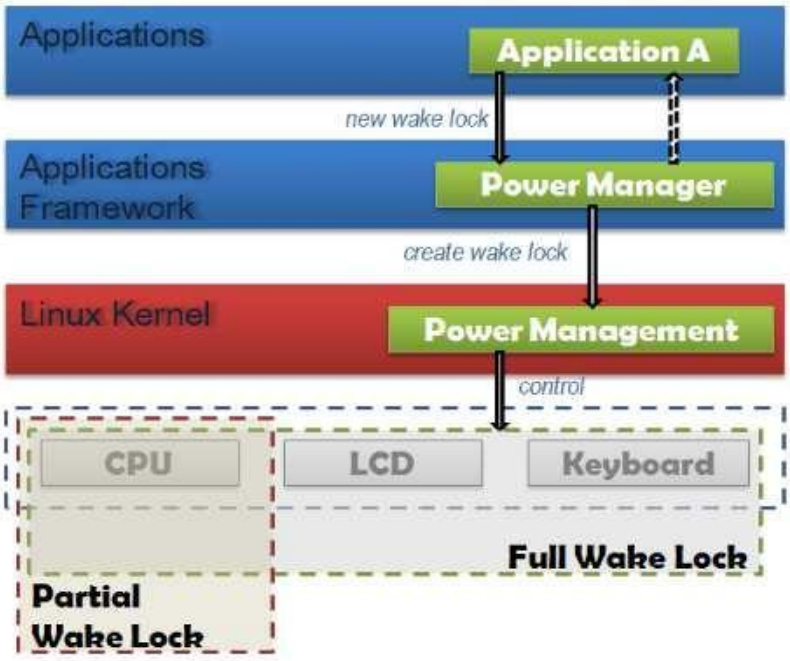


Figure 3.5 Android Power Management Architecture with wake locks.

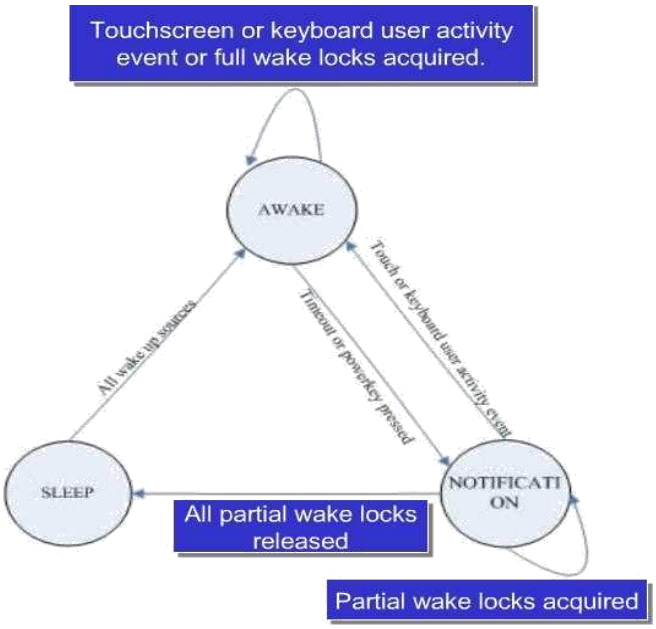


Figure 3.6 A finite state machine of Android PM.

Figure 3.6 shows that the full state machine. There are three states: “SLEEP”, “NOTIFICATION”, and “AWAKE”. The scenario is: While a user application acquire full wake lock or touch screen/keyboard activity event, the machine will enter or keep in the “AWAKE”. If timeout happen or power key pressing, the machine will enter “NOTIFICATION”. While partial wake locks acquiring, it will keep in “NOTIFICATION”. While all partial locks released, t he machine will go into “SLEEP”. In “SLEEP” mode, it will transit if all re source awake. This state machine make power saving of Android more feasible for mobile devices.

Finally, the main concept of Android PM is through wake locks and time out mechanism to switch state of system power, so that system power consumption will decrease. The Android PM Framework provides a software solution to accomplish power saving for mobile devices. The following diagram (Figure 3.7) shows the overall architecture of Android PM.

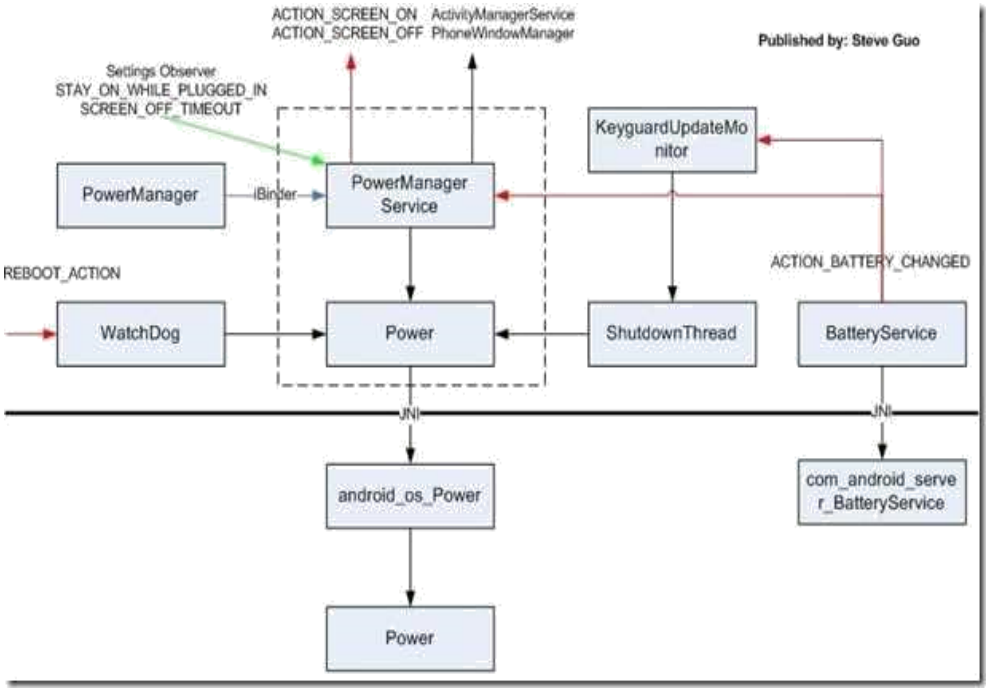


Figure 3.7 overall architecture of Android PM

3.4. Android PM Implementation

Android PM Framework provides a service for user space applications through the class PowerManger to achieve power saving. Hence, the app must get an instance of PowerManger in order to enforce its power requirements. The flow of exploring Wake locks are here:

Acquire handle to the PowerManager service by calling Context.getSystemService().
Create a wake lock and specify the power management flags for screen, timeout, etc.
Acquire wake lock.
Perform operation such as play MP3.
Release wake lock.

Here we provide an example code of PM. We will put the wake locks code on the function of onCreate() which will initialize first while the program start. And then release locks on the function of onDestroy() method. Then, we can control different type wake locks to accept different timeout or power saving mechanisms after finishing the implement.

```
PowerManager.WakeLock wl;
@Override
public void onCreate(Bundle savedInstanceState) {
    PowerManager pm = (PowerManager) getSystemService(Context.POWER_SERVICE);

    wl = pm.newWakeLock(PowerManager.FULL_WAKE_LOCK, "FULL_WAKE_LOCK");
    wl.acquire();

    super.onCreate(savedInstanceState);
    ...
}
```

CHAPTER 4

PROPOSED WORK

4.1. System Power Management States

Android operating system uses the Linux kernel and its power reduction model to achieve a more aggressive power management solution.

4.1.1. Standby

Standby is a low-latency power state that is sometimes referred to as “power-on suspend”. In this state, the system conserves power by placing the CPU in a halt state and the devices in the state.

4.1.2. Suspend

Suspend is also commonly known as “suspend to-RAM”. In this state, all devices are placed in sleep state except main memory, is expected to maintain power. Memory is placed in self-refresh mode, so its contents are not lost

4.1.3. Hibernate

Hibernate conserves the most power by turning off the entire system, after saving state to a persistent medium, usually a disk. All devices are powered off unconditionally. Hibernate is the only low-power state that can be used in the absence of any platform support for power management. Instead of entering a low-power state, the configured Power-Manager driver may simply turn the system off. This mechanism provides perfect power savings (by not consuming any), and can be used to work around broken power management firmware or hardware.

4.2. Proposed Technique- Enhancement of Power Management in Wi-Fi Direct

4.2.1. Problem Statement

For mobile device and embedded system device, it's much more important because the battery power is very limited. Nowadays, android phone and iPhone are more and more pervasive. There are more and more sensors and I/O in mobile device that can be used to improve the effectiveness of PM.

To improve power management on wifi direct my purposed idea is to implement hibernate concept for a process (wifi direct) for power Management. In hibernate mode the current state of the Process is saved to the non volatile memory and the process will power down. Once in Hibernation mode, system can come out of hibernation mode due to activity on selected peripherals (like LCD On, User interaction). Hibernate mode is similar to sleep mode, however in sleep mode the power cannot be shut off.

4.2.2. Proposed Solution-Hibernate Mode in Wi-Fi Direct

4.2.2.1. High Level Design

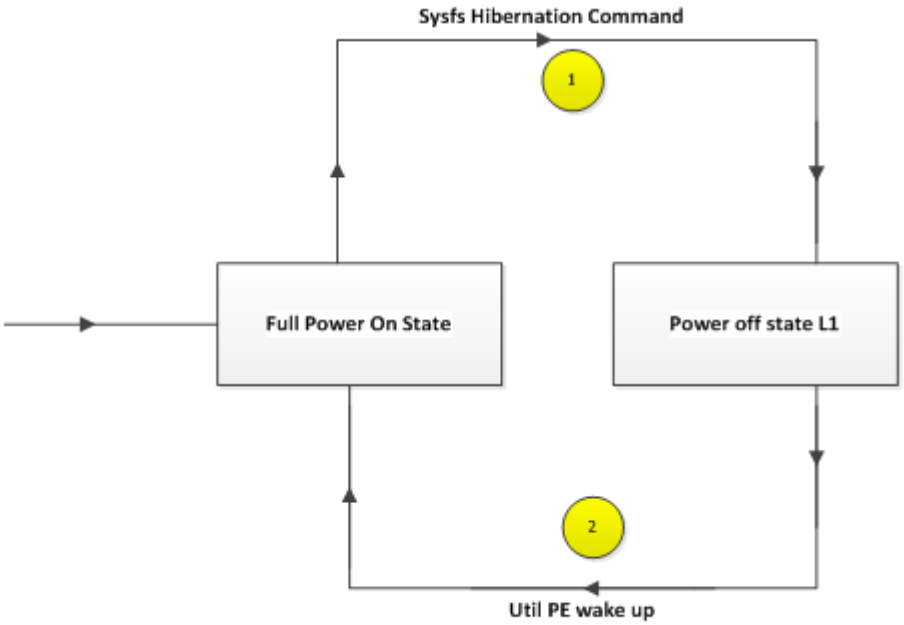


Figure 4.1 Hibernation

1) Devices entered to “Power off L1 state” on the request of user (Sysfs hibernation command)
2) Resumes its operation upon getting external event from Power Management Unit.

Transition to Hibernate mode can be invoked using sysfs command. A user application monitoring the system can invoke this sysfs command to enter into hibernation in case of no activity for certain period of time (Implementation Dependent).
As soon as the command is invoked, PM frameworks call all the device suspends functions registered by individual device drivers to take respective application in Power Off state.
Once in Hibernation mode, system can come out of hibernation mode due to activity on selected peripherals (like LCD On, User interaction). This is achieved by Power Management Unit, which generates interrupt to wake up the application.

4.2.2.2. Detailed Design (Software Implementation Detail)

4.2.2.2.1. Sysfs Interface

Sysfs Interface The interface exists in /sys/power/ directory
/sys/power/state: This controls the system power state

Reading from this file returns what states are supported:

Standby: Power-on Suspend
Mem: Suspend-to-RAM
Disk: Suspend-to-Disk

Writing to this file one of these strings causes the system to transition into that state

Sysfs Interface /sys/power/disk:

This controls the operating mode of the Suspend-to-Disk mechanism Operating modes:

Platform: put the system to sleep using platform driver (ACPI)
Shutdown: power off the process
Testproc: will cause the kernel to disable nonboot CPUs and freeze tasks, wait for 5 sec unfreeze tasks and enable nonboot CPUs
Test: will cause the kernel to disable nonboot CPUs and freeze tasks, shrink memory, suspend devices, wait for 5 seconds, resume devices, unfreeze tasks and enable nonboot CPUs

Sysfs Interface /sys/power/image_size

This controls the size of the image created by the Suspend-to-Disk mechanism

It can be written a string representing a non-negative integer that will be used as an upper limit of the image size, in bytes. The suspend-to-disk mechanism will do its best to ensure the image size will not exceed that number. However, if this turns out to be impossible, it will try to suspend anyway using the smallest image possible. In particular, if "0" is written to this file, the suspend image will be as small as possible

Reading from this file will display the current image size limit, which is set to 500 MB by default

/sys/power/resume

Reading from this file will display the major: minor numbers of the swap partition to be used for Suspend-to-Disk Writing major: minor numbers to this file will cause the system to resume from the mentioned partition

4.2.2.2.2 Power Management

Tasks: Freeze, Thaw

The freezing of tasks is a mechanism by which user space processes and some kernel threads are controlled during hibernation or system-wide suspend. The tasks are frozen before the hibernation image is created and are thawed after the system memory state has been restored from a hibernation image and devices have been reinitialized .

Relevant flags

include/linux/shed.h:

PF_NOFREEZE: this thread should not be frozen

PF_FROZEN: frozen for system suspend

PF_FREEZER_SKIP: Freezer should not count it as freezable

arch/arm/include/asm/thread_info.h

TIF_FREEZE

The tasks that have PF_NOFREEZE unset (all user space processes and some kernel threads) are regarded as freezable and treated in a special way before the Process enters a suspend state as well as before a hibernation image is created

Tasks: Freeze, Thaw
kernel/power/process.c, include/linux/freeze.h
Freeze:
freeze_processes():
It executes try_to_freeze_tasks() that sets TIF_FREEZE for all the freezable tasks and either wakes them up, if they are kernel threads, or sends fake signals to them if they are user space processes

A task that has TIF_FREEZE set should react to it by calling the function refrigerator(), which sets the tasks PF_FROZEN flag, changes its state to TASK_UNINTERRUPTIBLE and makes it loop until PF_FROZEN is cleared for it. Then, we say that the task is frozen

For user space processes try_to_freeze() is called automatically from the signal-handling code, but the freezable kernel threads need to call it explicitly in suitable places or use the wait_event_freezable() or wait_event_freezable_timeout()

Tasks: Freeze, Thaw
Thaw:
thaw_processes ():
It clears the PF_FROZEN flag for each frozen task. Then the tasks that have been frozen leave the refrigerator () and continue running

Tasks: Freeze, Thaw

Why is freezing of tasks required:

To prevent file systems from being damaged after hibernation. The hibernation image contains some file system-related information that must be consistent with the state of the on-disk data and metadata after the system memory state has been restored from the image (otherwise the file systems will be damaged in a nasty way). This is accomplished by freezing the tasks that might cause the on-disk file systems data to be modified after the hibernation image has been created and before the process is finally powered off. The majority of these are user space processes, but if any of the kernel threads may cause something like this to happen, they have to be freezable

To create the hibernation image a sufficient amount of memory (approximately 50% of available RAM) needs to be freed before the devices are deactivated. Then, after the memory for the image has been freed, we don't want tasks to allocate additional memory and that is prevented by freezing them earlier

To prevent user space processes and some kernel threads from interfering with the suspending and resuming of devices

Device Power Management

System Sleep Model:

Drivers can enter low power states as part of entering system-wide low-power states like Suspend-to-Ram, Suspend-to-Disk (hibernation) This is something that device, bus, and class drivers collaborate on by implementing various role-specific suspend and resume methods to cleanly power down hardware and software subsystems, then reactivate them without loss of data

include/linux/device.h

Bus Driver Methods: The core methods to suspend and resume devices reside in struct bus_type. Bus drivers implement those methods as appropriate for the hardware and the drivers using it

```
struct bus_type { ...
int (*suspend)(struct device *dev, pm_message_t state);
 int (*suspend_late)(struct device *dev, pm_message_t state);
int (*resume_early)(struct device *dev);
 int (*resume)(struct device *dev);
struct dev_pm_ops *pm;
};
```

Upper layers of driver stacks:

Device drivers generally have at least two interfaces, and the but_type methods are the ones which apply to the lower level (bus hardware). The network and block layers are examples of upper level interfaces, as is a character device talking to userspace.

Power management requests normally need to flow through those upper levels, which often use domain-oriented requests. In some cases those upper levels will have power management intelligence that relates to end-user activity, or other devices that work in cooperation. For the upper level interfaces that are structured using class interfaces, there is a standard way to have the upper layer stop issuing requests to a given class device (and restart later)

```
struct class { ... int (*suspend)(struct device *dev, pm_message_t state);
 int (*resume)(struct device *dev);
 struct dev_pm_ops *pm; };
```

Device:

```
struct device { ... struct device_type *type;
struct bus_type *bus;
struct class *class; ...
};

struct device_type { ... int (*suspend)(struct device *dev, pm_message_t state);
int (*resume)(struct device *dev);
struct dev_pm_ops *pm;
};
```

include/linux/sysdev.h

Sysdev

```
struct sysdev_class {
…
struct list_head drivers;
int (*shutdown)(struct sys_device *);
int (*suspend)(struct sys_device *, pm_message_t state);
int (*resume)(struct sys_device *);
};

struct sysdev_driver {
...
int (*shutdown)(struct sys_device *);
int (*suspend)(struct sys_device *, pm_message_t state);
int (*resume)(struct sys_device *);
};

struct sys_device { ...
struct sysdev_class *cls;
};
```

include/linux/pm.h

```
struct dev_pm_ops {
int (*prepare)(struct device *dev);
void (*complete)(struct device *dev);
 int (*suspend)(struct device *dev);
 int (*resume)(struct device *dev);
 int (*freeze)(struct device *dev);
int (*thaw)(struct device *dev);
 int (*poweroff)(struct device *dev);
int (*restore)(struct device *dev);
 int (*suspend_noirq)(struct device *dev);
int (*resume_noirq)(struct device *dev);
int (*freeze_noirq)(struct device *dev);
int (*thaw_noirq)(struct device *dev);
int (*poweroff_noirq)(struct device *dev);
 int (*restore_noirq)(struct device *dev); };
```

prepare(): Prepare the device for the upcoming transition, but do NOT change its hardware state. Prevent new children of the device from being registered after prepare() returns (the drivers subsystem and generally the rest of the kernel is supposed to prevent new calls to the probe method from being made too once prepare() has succeeded)

freeze(): Hibernation-specific, executed before creating a hibernation image. Quiesce operations so that a consistent image can be created, but do NOT otherwise put the device into a low power device state and do NOT emit system wakeup events. Save in main memory the device settings to be used by restore() during the subsequent resume from hibernation or by the subsequent thaw(), if the creation of the image or the restoration of main memory contents from it fails

thaw(): Hibernation-specific, executed after creating a hibernation image OR if the creation of the image fails. Also executed after a failing attempt to restore the contents of

main memory from such an image. Undo the changes made by the preceding freeze(), so the device can be operated in the same way as immediately before the call to freeze()

poweroff(): Hibernation-specific, executed after saving a hibernation image. Quiesce the device, put it into a low power state appropriate for the upcoming system state, and enable wakeup events as appropriate

restore(): Hibernation-specific, executed after restoring the contents of main memory from a hibernation image. Driver starts working again, responding to hardware events and software requests

complete(): Undo the changes made by prepare(). This method is executed for all kinds of resume transitions, following one of the resume callbacks: resume(), thaw(), restore(). Also called if the state transition fails before the drivers suspend callback (suspend(), freeze(), poweroff()) can be executed. The PM core executes complete() after it has executed the appropriate resume callback for all devices

freeze_noirq(), thaw_noirq(), poweroff_noirq(), restore_noirq(): Complete the operations of the corresponding function (freeze(), thaw(), poweroff(), restore()) by carrying out any actions required for freezing the device that need interrupts to be disabled

Notifiers :

There are some operations that device drivers may want to carry out in their suspend()/prepare() routines, but shouldnt, because they can cause the hibernation or suspend to fail. For example, a driver may want to allocate a substantial amount of memory (like 50 MB) in suspend()/prepare(), but that shouldnt be done after the swsusps memory shrinker has run. Also, there may be some operations, that subsystems want to carry out before a hibernation/suspend or after a restore/resume, requiring the system to be fully functional

A Hibernation notifier may be used for this purpose. The subsystems that have such needs can register suspend notifiers that will be called upon the following events by the suspend core:

PM_HIBERNATION_PREPARE: The system is going to hibernate or suspend, tasks will be frozen immediately

PM_POST_HIBERNATION: The system memory state has been restored from a hibernation image or an error occurred during the hibernation. Device drivers resume()/restore() callbacks have been executed and tasks have been thawed

PM_RESTORE_PREPARE: The system is going to restore a hibernation image. If all goes well the restored kernel will issue a PM_POST_HIBERNATION notification

PM_POST_RESTORE: An error occurred during the hibernation restore. Device drivers resume() callbacks have been executed and tasks have been thawed PM_SUSPEND_PREPARE: The system is preparing for a suspend

PM_POST_SUSPEND: The system has just resumed or an error occurred during the suspend. Device drivers resume () callbacks have been executed and tasks have been thawed

4.2.2.2.3 Memory Power Management

Memory Snapshot:

The hibernation core snapshots system memory by indexing and copying every active page in the system. Once a snapshot is complete, the saved image and index is stored persistently

The snapshot sequence is a three-step process. First, all of the active pages are indexed, enough new pages are allocated to clone these pages, then each page is copied into its clone

The snapshot process has one critical requirement: that at least half of the memory be free. For this the memory needs to be freed before the snapshot is taken. This is done using the memory shrinker

4.2.2.2.4. Hibernate Algorithm

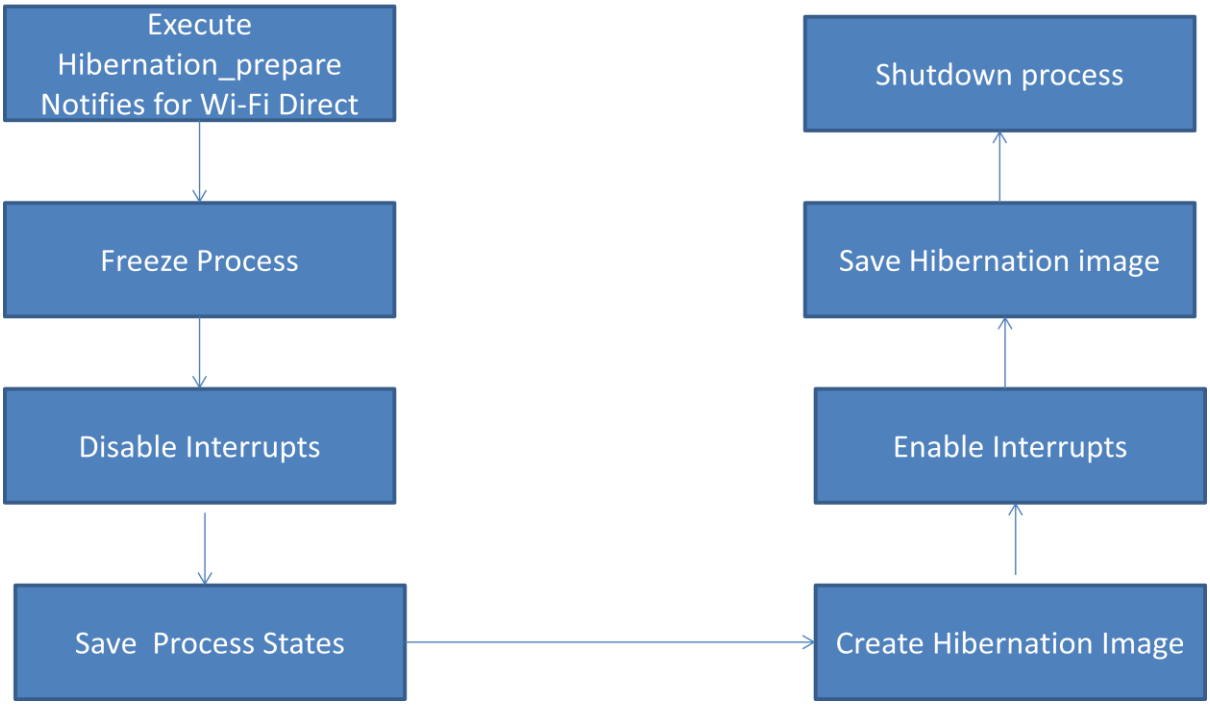


Figure 4.2 Hibernation Algorithm

1)The power management notifiers registered for the event PM_HIBERNATION_PREPARE are executed

2) All the file systems are synced (Flush file system buffers, force changed blocks to disk, update the superblock)

3) Tasks are frozen

4) The platform driver is informed that hibernation is being started

5) The requisite memory to save the snapshot is freed using the memory shrinker

6) The process are suspended for the event P_EVENT_FREEZE. For each dev in dpm_list the following are executed (in sequence): (dev->bus->pm, dev->type->pm, dev->class->pm)->prepare(), dev- >class->pm->freeze() or dev->class->suspend(), dev->type->pm->freeze() or dev->type- >suspend(), dev->bus->pm->freeze() or dev->bus->suspend()

7) Process is prepared for hibernation using the platform driver

8) Interrupts are disabled for the process

9) Late suspend of the process (that require interrupts to be disabled) is carried out for the event PM_EVENT_FREEZE. For each dev in dpm_list the following is executed: dev->bus->pm- >freeze_noirq() or dev->bus->suspend_late()

10) Process are suspended for the event PM_EVENT_FREEZE. For each cls in system_kset->list following is executed (in sequence): (For each drv in cls->drivers) ->suspend(), cls->suspend()

11) Co-processor and the processor state is saved

12) Atomic copy of the system memory (hibernation image) is created

13) The co-processor state is restored

14) Process is switched to normal mode of operation using platform driver

15) The devices are resumed for the event P_EVENT_THAW. For each dev in dpm_list the following is executed (in sequence): dev->bus->pm->thaw() or dev->bus->resume(), dev->type->pm->thaw() or dev->type->resume(), dev->class->pm->thaw() or dev->class->resume(), (dev->class->pm, dev->type->pm, dev->bus->pm)->complete()

16) The platform driver is informed that the system has entered the working state

17) Hibernation image is saved in the allocated swap partition

18) Process is shutdown

4.2.2.2.5. Resume Algorithm

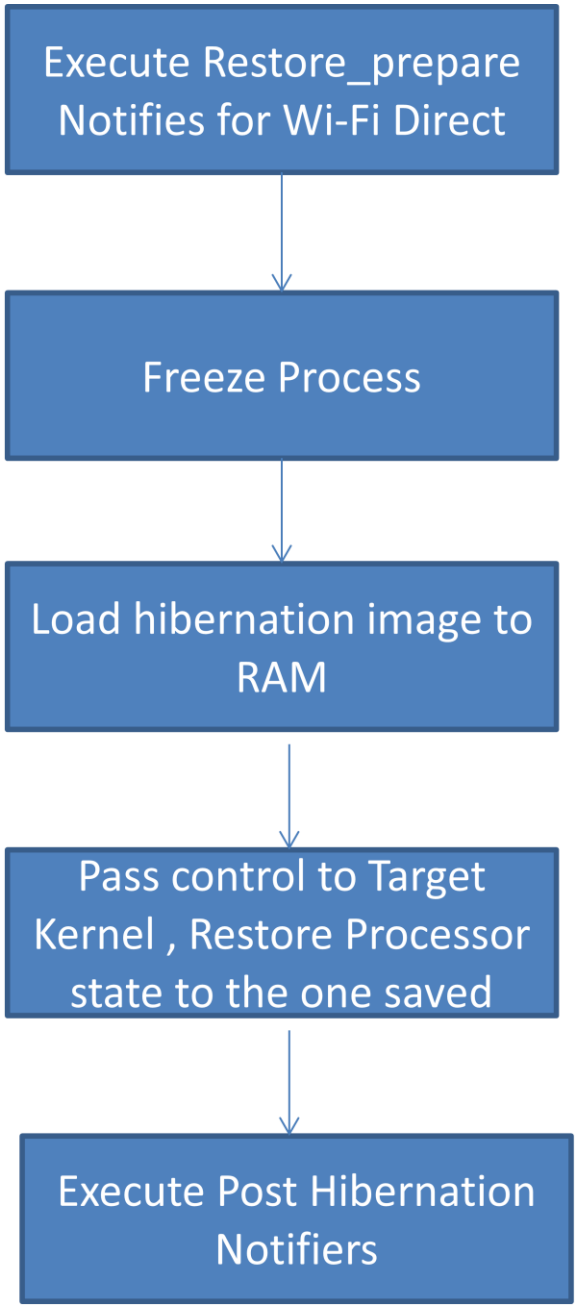


Figure 4.3 Resume Algorithm

1) The power management notifiers registered for the event PM_RESTORE_PREPARE are executed

2) Tasks are frozen

3) Hibernation image is loaded into RAM

4) The devices are suspended for the event P_EVENT_QUIESCE. For each dev in dpm_list the following is executed (in sequence): (dev->bus->pm, dev->type->pm, dev->class->pm)->prepare(), dev- >class->pm->freeze() or dev->class->suspend(), dev->type->pm->freeze() or dev->type- >suspend(), dev->bus->pm->freeze() or dev->bus->suspend()

5) The platform is prepared for restoration from a hibernation image

6) Interrupts are disabled on the main CPU

7) Late suspend of the devices (that require interrupts to be disabled) is carried out for the event PM_EVENT_QUIESCE. For each dev in dpm_list the following is executed: dev->bus->pm- >freeze_noirq() or dev->bus->suspend_late()

8) System devices are suspended for the event PM_EVENT_QUIESCE. For each cls in system_kset- >list the following is executed (in sequence): (For each drv in cls->drivers)->suspend(), cls- >suspend()

9) Machine is prepared for switching to normal mode of operation using the platform driver System devices are resumed. For each cls in system_kset->list the following is executed (in sequence): cls->resume(), (For each drv in cls->drivers)->resume()

10) Early resume of the devices is carried out for the event PM_EVENT_RESTORE. For each dev in dpm_list the following is executed: dev->bus->pm->restore_noirq() or dev->bus->resume_early()

11) Interrupts are enabled on the the main CPU

12) Process is switched to normal mode of operation using the platform driver

13) The devices are resumed for the event P_EVENT_RESTORE. For each dev in dpm_list the following is executed (in sequence): dev->bus->pm->restore() or dev->bus->resume(), dev->type->pm- >restore() or dev->type->resume(), dev->class->pm->restore() or dev->class->resume(), (dev- >class->pm, dev->type->pm, dev->bus->pm)->complete()
The platform driver is informed that the system has entered the working state

14) Tasks are thawed

15) Power management notifiers are executed for PM_POST_HIBERNATION

CHAPTER 5

SIMULATION RESULTS & ANALYSIS

5.1. Simulation Setup: A Power Monitor for Android-Based Mobile Platforms

PowerTutor is an application for Google phones that displays the power consumed by major system components such as CPU, network interface, display, and GPS receiver and different applications. The application allows software developers to see the impact of design changes on power efficiency. Application users can also use it to determine how their actions are impacting battery life. PowerTutor uses a power consumption model built by direct measurements during careful control of device power management states. This model generally provides power consumption estimates within 5% of actual values.We can use PowerTutor to monitor the power consumption of any application.

PowerTutor's power model was built on Samsung, HTC G1, HTC G2 and Nexus one. It will run on other versions of the GPhone, but when used with phones other than the above phone models, power consumption estimates will be rough.

PowerTutor was developed by University of Michigan Ph.D. students Mark Gordon, Lide Zhangand Birjodh Tiwana under the direction of Robert Dick and Zhuoqing Morley Mao at the University of Michigan and Lei Yang at Google. The work is supported primarily by National Science Foundation grant CNS-1059372 under Program Manager Professor Theodore Baker. It received prior support from Google and the National Science Foundation under awards CCF-0964763 and CNS-0720691, and was done in collaboration with the joint University of Michigan and North western University Empathic Systems Project.

Experiment is done on below combination of setup:

Platform: Android
OS: Linux
Chipset: Qualcomm (8930)
Driver: Wi-Fi Prima
Simulation Tool : Power Tuitor

5.2.2. Performance Evaluation- Hibernate Mode in Wi-Fi Direct

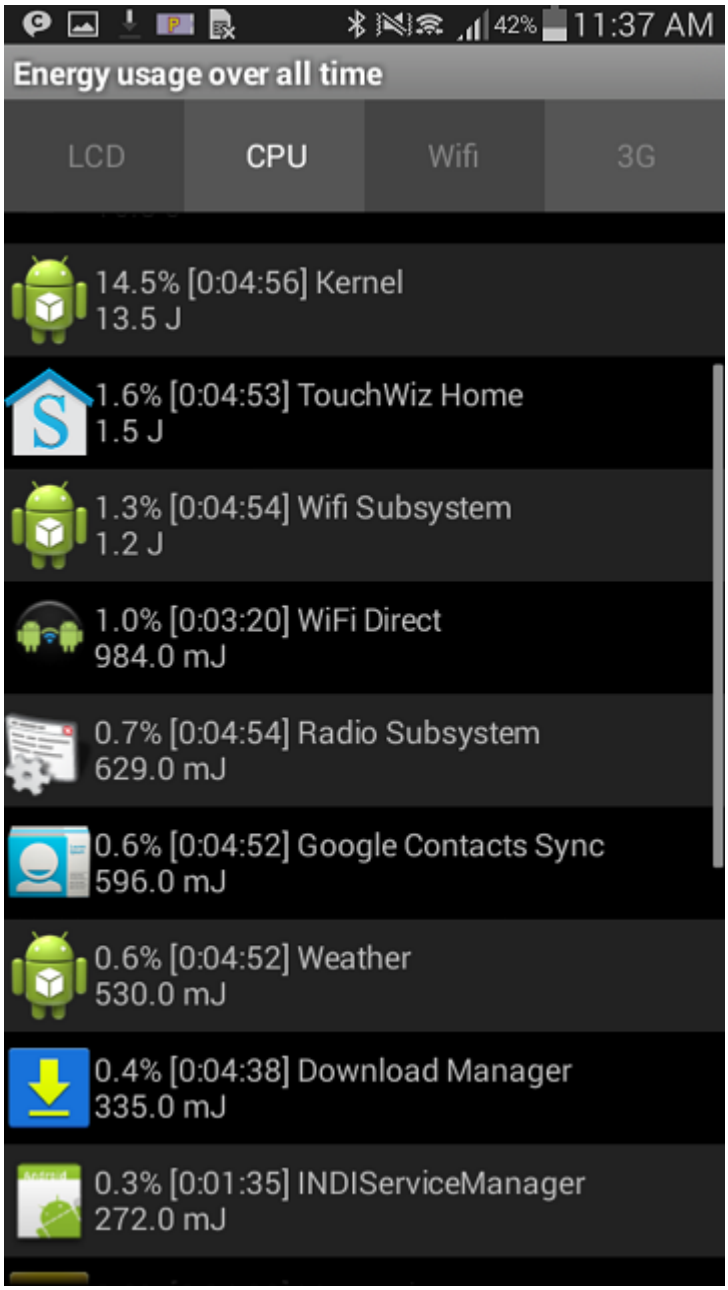


Figure 5.1 List of running application

5.2.1. Simulation Result (Normal Mode)

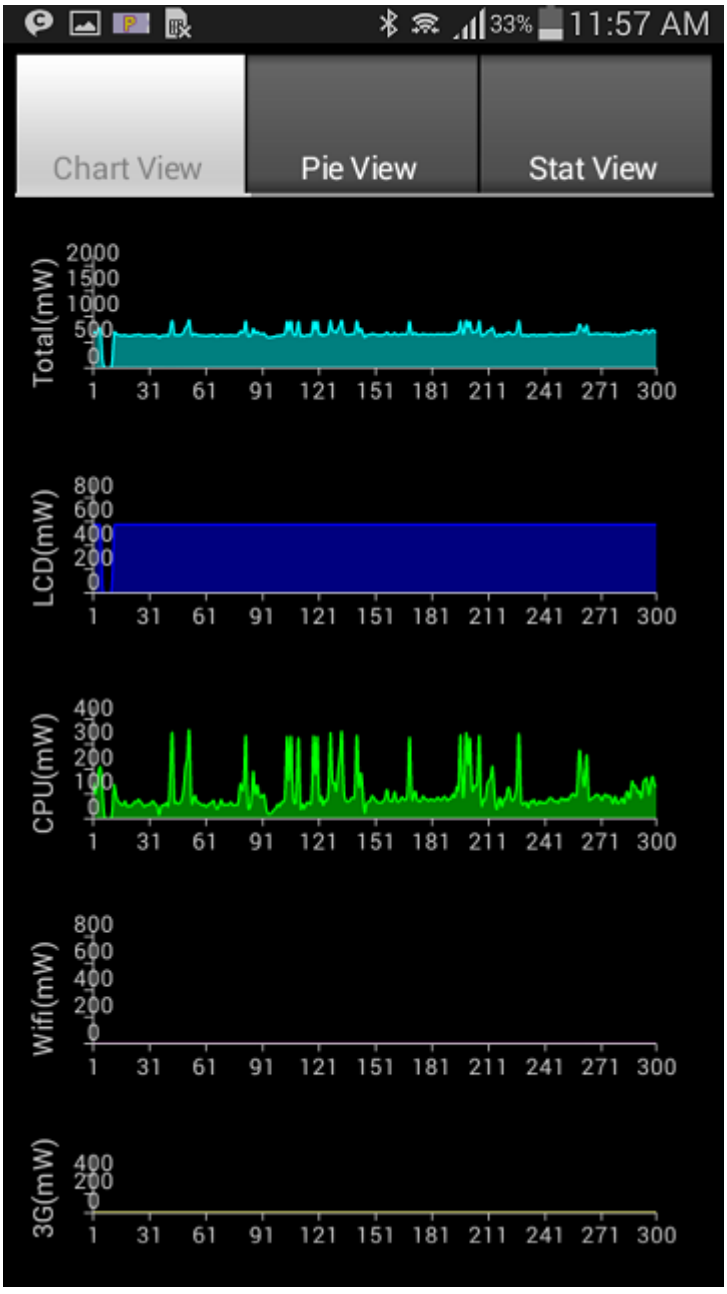


Figure 5.2 Wifi Direct power consumption Graph on Normal Mode

Simulation Result (Hibernate Mode)

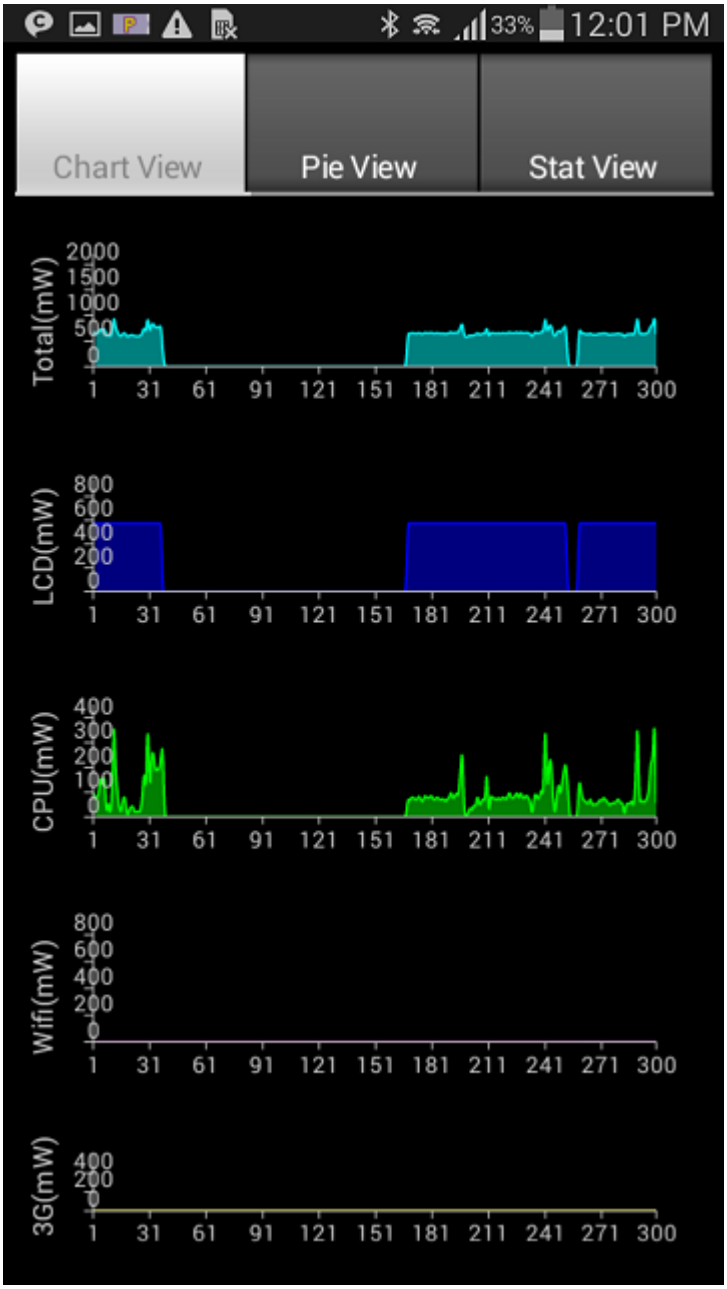


Figure 5.3 Wifi Direct power consumption Graph on Hibernate Mode

5.2.2. Analysis

From the simulation performed the observations can be quantified in normal mode and hibernation mode simulation result.

We observe Power consumption is less in case of hibernation mode in compare with normal model of execution in case when LCD is off and there is no packet transfer between devices through wifi direct.

It is observe there is minimum power consumption in case of hibernation. (Note: Power Tutor tool, power consumption estimates within 5% of actual values)

Hibernation concept is used in system to save more power and in experiment it is observed that hibernation technique can be used for any process depending upon the requirement and situation to save more power.

CHAPTER 6

CONCLUSION AND FUTURE WORK

Power saving is an important issue for mobile devices, and there are many ways to implement. How to design a PM for mobile device need is a good question. Android builds up its user space level solution in order to maintain Linux fundamental support and increase flexibilities.

The future of Wi-Fi Direct is very promising as it is becoming the next prominent P2P communication technology for various mobile applications and services. Anticipating this trend, we propose an enhancement of power saving in Wi-Fi Direct to improve its performance. Performance evaluation through simulation shows that our proposed scheme can tune improves power management, guaranteeing better energy efficiency and service reliability. In the future, we plan to experiment with the Wi-Fi Direct implementation using more devices and to explore the practical costs. In addition; we plan to add an easy-to-use graphical user interface and to release an application that is useable by mobile user.

To sum up, PM is very important for mobile device but it still have room for improvements.

References

[1] Wi-Fi Alliance, P2P Technical Group, Wi-Fi Peer-to-Peer (P2P) Technical Specification v1.1, 2010.

[2] IEEE P802.11™-2012, Part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications, IEEE, 2012.

[3] Google, Android 4.1 Compatibility Definition, Revision 2, Sept. 2012.

[4] Powet Tuitor

[5] Wi-Fi Alliance, http://www.wi-fi.org

[6] D. Camps-Mur, X. Perez-Costa, S. Sallent-Ribes, "Designing energy efficient hop access points with Wi-Fi Direct," Computer Networks, 55, pp.2838-2855, 2011.

[7] J. Choi, Y. Ko, J. Kim, "Enhanced Power Saving Scheme for IEEE 802.11 DCF based Wireless Networks", 8th International Conference on Personal Wireless Communications, pp. 835-840, Sep. 2003.

[8] E.Jung, N. Vaidya, "Improving IEEE 802.11 power saving mechanism,' Wireless Networks, 14, pp. 375-391, 2008

[9] X. Hu, Z. Chen, Z. Yang, "Energy-efficient Scheduling Strategies in IEEE 802.11 Wireless LANs," International Conference on ComputerScience and Automation Engineering, pp. 570-572, May, 2012.

[10] Robin Kravets, P. Krishnan , Application-driven power management for mobile communication .

[11] Andreas Weissel, Frank Bellosa, Process cruise control: event-driven clock scaling for dynamic power management.

[12] APM V1.2 spec.

[13] ACPI, http://www.lesswatts.org/projects/acpi/index.php

[14] Android project- power management
http://www.netmite.com/android/mydroid/development/pdk/docs/power_management.html

[15] Steve Guo, Android Power Management

[16] Matt Hsu, Jim Huang, Power Management from Linux Kernel to Android, 0xlab, 2009.