



**Computer System Engineering Department**

**ENCS530: Graduation Project**

**Secure IPv6 in Mobile Environments:**

**SEND Implementation for Android Platform**

Students:

Ayham Jaradat (1090708) and Sarah Hattab (1090527)

Supervisor:

Dr. Ahmad Alsadeh.

June 12th, 2014

## **Abstract**

The goal of this project is to implement one of the main security mechanisms on Android mobile devices to secure IPv6 addressing process. We mainly focus on “SEcure Neighbor Discovery” (SEND) mechanism. We built an Android application that is capable of capturing IPv6 packets from the device network stack, and adding the SEcure Neighbor Discovery options to these packets. The application uses C library to capture IPv6 packets and Java classes to modify and verify the SEND options in these packets. However, Android devices are being improved by Google to be able to connect to IPv6 networks dynamically in the next future versions, and because this task is not ready yet we tend to use static configuration method as a temporary solution to clarify our Android mobile application functionalities.

# Table of Contents

Chapter 1: Introduction .....	1
1.1 Motivation .....	1
1.2 Problem Statement .....	2
1.3 Contributions .....	2
1.4 Report Outline .....	2
Chapter 2: Background and Related Work .....	3
2.1 Internet Protocol version 6 (IPv6).....	3
2.1.1 IPv6 Header Format .....	4
2.1.2 IPv6 Addresses Types.....	5
2.1.3 Unicast Addresses Format .....	6
2.1.4 IPv6 Stateless Address Autoconfiguration .....	7
2.2 Neighbor Discovery Protocol (NDP).....	8
2.2.1 Neighbor Discovery Functionalities and Messages .....	8
2.2.2 Neighbor Discovery Implications .....	10
2.3 SEcure Neighbor Discovery (SEND) .....	12
2.3.1 Router Authorization.....	12
2.3.2 Address Ownership Proof Mechanism and Protected messages.....	13
2.4 Android .....	16
2.4.1 Open Source Advantages.....	16
2.4.2 Architecture overview.....	16
2.4.3 Integrated Development Environment (IDE) and Tools.....	18
2.5 Related Work.....	19
2.5.1 TrustRouter .....	19
2.5.2 EASY-SEND.....	20
2.5.3 NDprotector .....	20
2.6 Chapter summary .....	21
Chapter 3: SEND Implementation on Android.....	22
3.1 Capturing IPv6 Packets on Android .....	22
3.1.1 The C-language Code Functionalities.....	23

3.2 Handling Captured IPv6 Packets .....	24
3.3 SEND Software Structure.....	25
3.3.1 Use Case Diagram.....	25
3.3.2 Class Model Diagram (UML) .....	30
3.3.3 Component Relationship Model.....	33
3.4 Problems Faced and Constraint .....	34
3.5 How to Run the Application.....	35
<b>Chapter 4: Results and Conclusions.....</b>	<b>37</b>
4.1 Results.....	38
4.1.1 CGA Computational Cost Results .....	38
4.1.2 Application Testing Results.....	39
4.2 Conclusions .....	42
<b>References .....</b>	<b>43</b>

## List of Figures

Figure 2.1 IPv6 Packet Fix Header Format[2].....	4
Figure 2.2 IPv6 Types and their notations.....	6
Figure 2.3 IPv6 Unicast Address Format.....	6
Figure 2.4 Android software Architecture[27].....	17
Figure 3.1 Ip6table Rules .....	23
Figure 3.2 The C-language code functionality.....	24
Figure 3.3 Use Case Diagram of our SEND software.....	26
Figure 3.4 Application main Page .....	27
Figure 3.5 Application SEND Parameters Page .....	28
Figure 3.6 Application Test CGA Page.....	29
Figure 3.7 UML digram of SEND Option classes.....	31
Figure 3.8 UML Diagram of Packet Handling Classes.....	32
Figure 3.9 Component Relationship model.....	34
Figure 3.10 commands to compile and copy the C executable file to Android device.....	36
Figure 4.1 eclipse LogCat output for running C executable file.....	37
Figure 4.2 eclipse LogCat output for Android Java application.....	38
Figure 4.3 snapshot from CGA computational cost testing.....	39
Figure 4.4 some logging information from Android Application.....	40
Figure 4.5 some logging information from Android Application.....	41

## Table of Abbreviations

aapt	Android Asset Packaging Tool
adb	Android Debug Bridge
ADD	Authorization Delegation Discovery
ADT	Android Developer Tools
ARP	Address Resolution Protocol
ASL	Apache Software License
CGA	Cryptographically Generated Addresses
CPA	Certification Path Advertisement
CPS	Certification Path Solicitation
DAD	Duplicate Address Detection
ddms	Dalvik Debug Monitor Service
DHCPv6	Dynamic Host Configuration Protocol for IPv6
DNS	Domain Name Server
DoS	Denial of Service
dx	Dalvik Cross-Assembler
EUI-64	Extended Unique Identifier
GPLv2	General Public License, version 2.0
ICMP	Internet Control Message Protocol
ICMPv6	Internet Control Message Protocol for IPv6
ID	Interface Identifier
IETF	Internet Engineering Task Force
IPv4	Internet Protocol Version 4
IPv6	Internet Protocol Version 6
ISPs	Internet Service Providers
JPL	Java programming language
MAC	Media Access Control
MITM	Man- In- The- Middle
NA	Neighbor Advertisement
ND	Neighbor Discovery
NDK	Native Development Kit
NDP	Neighbor Discovery Protocol
NS	Neighbor Solicitation
OS	Operating System
PKCS	Public-Key Cryptography Standards
RA	Router Advertisement
RM	Redirect message
RS	Router Solicitation
SDK	Software Development Kit
SEND	SEcure Neighbor Discovery
SLAAC	Stateless Address Auto Configuration

# Chapter 1: Introduction

Internet Protocol version 4 (IPv4) addresses are running out, and thus there is a necessary change from IPv4 to IPv6. IPv6 was developed by the Internet Engineering Task Force (IETF) to deal with the problem of IPv4 address exhaustion [1], IETF put security as a design goal for this new Internet Protocol. Internet Protocol version 6 has integrated the security capability based on IPsec [2]. But, IPsec cannot be used before the nodes have a valid IPv6 address [3].

A host needs to automatically configure a valid IPv6 on its interface, in order to benefit from the network. This operation is achieved using the Neighbor Discovery Protocol (NDP) [12], which includes Neighbor Discovery for IPv6 and IPv6 Stateless Address Auto Configuration (SLAAC) [4]. The NDP is used for critical functionalities, but it is also prone to critical attacks. To defend these attacks, a set of enhancements were added to NDP to become SEcure Neighbor Discovery (SEND) [5], RFC 3971.

The SEcure Neighbor Discovery (SEND) is a security extension of the Neighbor Discovery Protocol (NDP) in IPv6. It uses Cryptographically Generated Addresses (CGAs) [6], a digital signature and a X.509 certification to protect NDP. SEND enhances and solves NDP security issues and makes NDP a safe protocol, but its deployment is not easy and it is a compute intensive and bandwidth consuming, for example, CGA computations can take a long time varies from seconds to hours for different security-level values, especially for devices with limited resources such as mobile phones.

## 1.1 Motivation

Smart phones world is growing rapidly, and most of companies are providing their products and services to smart phones users. With major Internet Service Providers(ISPs) and companies around the world permanently enabled IPv6 for their products and services, examples of these companies are Google, Cisco and Facebook, Therefore there is a need for migrating to IPv6 in a secure way in mobile environments, where devices have limited resources and capabilities.

## 1.2 Problem Statement

The Internet has become a very essential part of our life. Manufacturing devices for enabling Internet and developing their software is the fastest technology nowadays. Smart phones are the most popular devices achieving that purpose in the late few years, and due to large number of smart phones which are needed to connect to Internet, secure IPv6 become a necessary requirement. Accordingly, our main goal in this project is to implement the IPv6 security mechanism SEND for Android mobile devices to secure IPv6 addressing process. We chose Android platform for many reasons, most importantly because it is an open source operating system, and because Android is the world's most popular mobile platform [17].

## 1.3 Contributions

Our contribution mainly appears in the SEND implementation on Android platforms, we implemented the *libnetfilter\_queue* library on Android platform to capture IPv6 packets; we modified the library's queues names to run on Android, then we compiled the library and build it as external library to be used with Android devices.

Moreover, we wrote all of the needed classes to handle IPv6 packets; the classes modify the packets, add SEND options and verify them on IPv6 packets.

## 1.4 Report Outline

The rest of the report includes Chapter 2 that describes background and related work, Chapter 3 describes the proposed work, and Chapter 4 concludes the work and gives an outlook for the future work.



## Chapter 2: Background and Related Work

Smart phones world set to become the fastest spreading technology in the human history. As engineers we have to search about gaps in that world to fill them, and that was the main purpose of our project implementation as mentioned before. This chapter introduces the background theory, technology used, and related work with different implementations of SEcure Neighbor Discovery mechanism (SEND).

### 2.1 Internet Protocol version 6 (IPv6)

The Internet Protocol version 6 (IPv6) was proposed in 1995 as a solution to the limitations on globally unique addressing that IPv4's 32-bit addressing space represented [1]. The changes from IPv4 to IPv6 can be summarized in five categories as following.

- Expanded Addressing Capabilities

The IP address size increased from 32 bits in IPv4 to 128 bits in IPv6.

- Header Format Simplification

IPv6 have less header format than IPv4, to reduce the processing cost of packet handling and to limit the bandwidth cost of the IPv6 header.

- Improved Support for Extensions and Options

IPv6 header options allow for more efficient forwarding and greater flexibility for introducing new options in the future.

- Flow Labeling Capability

IPv6 enables the labeling of packets that belongs to particular traffic; this allows special handling of packets such as in real-time service.

- Authentication and Privacy Capabilities

IPv6 specifies some extensions to support authentication, data integrity, and data confidentiality.

### 2.1.1 IPv6 Header Format

Any IPv6 packet consists of two parts, control information and payload. The control information part contains a mandatory fixed header and an optional extension header, while the payload is the part that contains the actual data being sent.

The IPv6 packet Fixed Header is a 40 octets (320 bits) representing the following fields as depicted in Figure 2.1 [2].

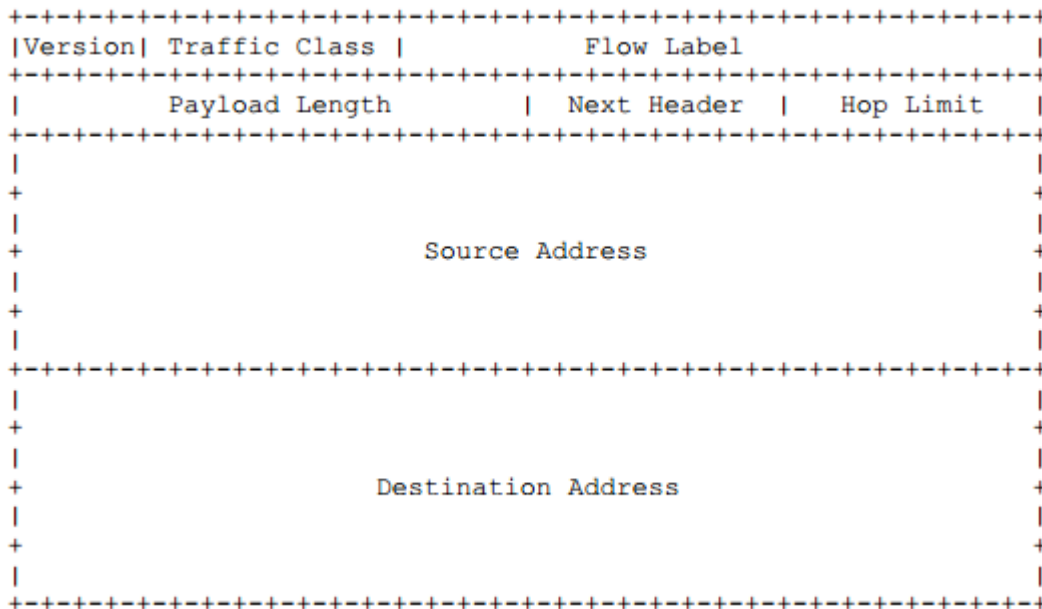


Figure 2.1: IPv6 Packet Fix Header Format [2]

- Version  
4-bit Internet Protocol version number equals to 6.
- Traffic Class

8-bit traffic class field, the 6 most significant bits are used for differentiated services, and the least 2 significant bits are used for priority values.

- Flow Label

20-bit flow label, it is used to help in real time applications, when the sender requests special handling.

- Payload Length

16-bit unsigned integer represents the length of IPv6 payload in octets.

- Next Header

8-bit selector, it identifies the type of header immediately following the IPv6 header.

- Hop Limit

8-bit unsigned integer, it is similar to IPv4 time-to-live field, it is decremented by 1 by each node that forwards the packet, and the packet is discarded when Hop Limit reaches to zero.

- Source Address

128-bit address representing the IPv6 address of the sender.

- Destination Address

128-bit address representing the IPv6 address of the intended recipient.

### 2.1.2 IPv6 Addresses Types

The 128-bit IPv6 addresses can be classified into three types differ from each other in their formats and use. They are Unicast, Anycast and Multicast. All types of IPv6 addresses are assigned to interfaces, not nodes. And the type of an IPv6 address is identified by the high-order bits of the addresses.

- Unicast address is used to identify a single interface. There are several types of unicast addresses in IPv6, such as Global unicast and Link-local unicast addresses. All interfaces are required to have at least one Link-local unicast address.
- Anycast address is used to identify a set of interfaces belonging to different nodes. When a packet is sent to an anycast address, it is delivered to the nearest interface from the interfaces identified by that address. It depends on the routing protocols' measure of distance to find the nearest one.
- Multicast Address is used to identify a set of interfaces belonging to different nodes. When a packet is sent to a multicast address, it is delivered to all interfaces identified by that address.

There are also some special reserved addresses such as the Unspecified Address (0:0:0:0:0:0:0:0) which is used to indicate the absence on an address, and the Loopback Address (0:0:0:0:0:0:0:1) which is a unicast address used by a node to send an IPv6 packet to itself.

Figure 2.2 shows the binary prefix and the IPv6 notation for IPv6 address types.

Address type	Binary prefix	IPv6 notation
Unspecified	00...0 (128 bits)	::/128
Loopback	00...1 (128 bits)	::1/128
Multicast	11111111	FF00::/8
Link-Local unicast	1111111010	FE80::/10
Global Unicast	(everything else)	

Figure 2.2: IPv6 Types and their notations

### 2.1.3 Unicast Addresses Format

There are several types of Unicast IPv6 addresses, such as site-local unicast and IPv6 addresses with embedded IPv4 addresses. But our main focus in on Link-Local unicast and Global Unicast.

Mainly any Unicast address consists of two parts, subnet prefix and interface identifier as shown in figure 2.3.

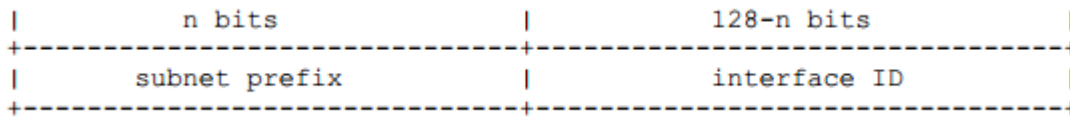


Figure 2.3: IPv6 Unicast Address Format

The subnet prefix is a typically hierarchically structure value assigned to a cluster or subnet. The Interface Identifier (ID) in IPv6 unicast address is used to identify an interface on a link, so it is required to be unique within a subnet prefix. And for all unicast addresses, except the addresses that start with the binary value 000, the interface identifier is required to be 64 bits long [10].

#### 2.1.4 IPv6 Stateless Address Autoconfiguration

One of the main advantages of IPv6 is IP addresses Autoconfiguration where a node can easily configure its IP address when connecting to a network. The Autoconfiguration process includes generation a link-local address, generating global addresses via Stateless Address Autoconfiguration (SLAAC), and the Duplicate Address Detection (DAD) procedure to verify the uniqueness of the addresses on a link.

A node uses Stateless Address Autoconfiguration (SLAAC), and a Neighbor Discovery Protocol to automatically generate a link-local address on each enabled IPv6 interface. The address is generated using a combination of locally available information and information advertised by routers. Routers advertise prefix that identify the subnet, while hosts generate an Interface Identifier that is unique on a subnet. An address is formed by combining the prefix from router and the Interface Identifier from host itself [4].

The 64-bit Interface Identifier part of the IPv6 address is usually constructed in Modified Extended Unique Identifier (EUI-64) format. The 48-bit MAC address of the interface is used in this mechanism to construct the 64-bit interface identifier [10].

## 2.2 Neighbor Discovery Protocol (NDP)

Neighbor Discovery Protocol is a one of the IPv6 protocols that operates in the Link Layer and allows nodes on the same link to advertise their existence to their neighbors and to learn about the existence of other neighbors. The Neighbor Discovery Protocol is build on top of Internet Control Message Protocol version 6 (ICMPv6). The protocol defines five different ICMPv6 packet types to replace the Address Resolution Protocol (ARP), Internet Control Message Protocol (ICMP), Router Discovery (RDISC) and Router Redirect protocols of IPv4.

### 2.2.1 Neighbor Discovery Functionalities and Messages

Neighbor Discovery Protocol solves some of the problems related to the interaction between nodes on the same link. It is responsible for address autoconfiguration, discovery of other nodes on the link, determining the Link Layer addresses of other nodes, duplicate address detection, parameter discovery, finding available routers, redirect, next-hop determination, address prefix discovery, and neighbor unreachability detection.

Neighbor Discovery Protocol defines five different ICMPv6 packet types, each one of Neighbor Discovery messages consist of an ICMPv6 header, neighbor discovery message specific data, and ND message options, which provides additional information such as link-layer addresses, on-link network prefixes, on-link maximum transmission unit information, redirection data, mobility information and router specification [12].

The five ICMPv6 packet types of the Neighbor Discovery Protocol are described as follow:

- Router Solicitation (RS): an ICMPv6 message with type 133, hosts may send out RS to discover routers on the same link, and learn network information such as prefixes and Domain Name Server (DNS) addresses. Hosts send out RS to request routers to generate Router Advertisements (RA) immediately rather than at their next scheduled time.
- Router Advertisement (RA): an ICMPv6 message with type 134, Routers advertise their existence by sending RA message periodically, or in response to a Router

Solicitation message. Router Advertisements contain prefixes, address configuration, a suggested hop limit value, etc.

- Neighbor Solicitation (NS): an ICMPv6 message with type 135, node sends NS message to determine the link-layer address of neighbor nodes, or to verify that the node is still reachable. NS messages are also used for Duplicate Address Detection (DAD).
- Neighbor Advertisement (NA): an ICMPv6 message with type 136, nodes send NA message as a response to a NS message with information about its link-layer address. A node may also send unsolicited Neighbor Advertisements to announce a link-layer address change.
- Redirect message (RM): an ICMPv6 message with type 137. Routers use RM to inform hosts of a better first hop for a destination.

Routers send unsolicited Router Advertisements (RA) every 200 seconds (by default) to all nodes link-local multicast group (FF02::1). In this way; hosts discover routers on the same link they are connected to. Router Advertisement (RA) message contains the following information:

- Prefixes on the link
- Prefix lifetime
- A flag to indicate whether it can be used for stateful or stateless autoconfiguration.
- Default router information
- Parameter information such as MTU to use, and maximum hop-limit.

Hosts do not need to wait the assigned time to receive a periodic RA message [13], they can issue Router Solicitations (RS) messages in order to request Router Advertisement (RA) from routers. RS messages are sent to all routers link local multicast address (FF02::2). Hosts can use the unspecified source address (::/128) or link-local address as the source address of RS message.

Hosts can use prefixes sent by routers in RA messages to configure themselves with a unique IPv6 address using Stateless Address Autoconfiguration (SLAAC) and the EUI-64 conversion.

The flag sent with RA message indicates whether to use stateful or stateless autoconfiguration. DHCPv6 is used if the link is set to use stateful autoconfiguration. DHCPv6 is much like DHCP for IPv4, it allocates and stores host IP addressing information [13].

Address Resolution functionality means determining the Link Layer addresses of other node. Router resolution is used when a node needs to send a packet to IPv6 address but does not know the link-layer address to forward it to. This is similar to Address Resolution Protocol (ARP) in IPv4. In IPv6 the Neighbor Solicitation and Neighbor Advertisement messages are used to find which link layer address to use in order to send data to a destination IPv6 address.

When an interface is initialized or reinitialized, it uses SLAAC to associate a link-local address with that interface. To ensure that the configured IP address is unique on the link, the node run Duplicate Address Detection (DAD) algorithm. The node joins the multicast group FF02::1 and the solicited-node multicast address. It sends NS messages to the solicited-node multicast with that address and waits for NA. If no node responded with NA then the address is unique.

### 2.2.2 Neighbor Discovery Implications

Neighbor Discovery Protocol and Stateless Address Autoconfiguration are used by nodes in an IPv6 network to learn the local topology, including the Media Access Control (MAC) address to IP mappings for the local nodes. Since Neighbor Discovery is a link-local protocol, some protection mechanisms are used to protect NDP based on its scope, for example, the source address must be unspecified or a link-local address, the hop limit must be set to 255, and the routers do not forward link-local messages. However, this basic protection is not enough to protect IPv6 local networks. The initial NDP specifications suggest that IPsec may be used to secure Neighbor Discovery Protocol, but does not specify how. It appears that using current IPsec mechanisms is problematic due to key management problems [14].

Neighbor Discovery Protocol suffers from security and privacy implications. It is vulnerable to several security threats that attacks against pure Neighbor Discovery functions without involving routers or routing information, or other threats relevant to router discovery or router



related mechanisms [15]. At the end, any threat could be considered as one of the following attack types or combinations of them.

- Denial of Service (DoS): these attacks prevent communication between two legitimate nodes, using significant system resources. For example, attackers can generate DoS on DAD to prevent a network node from obtaining a network address. There are also Flooding Denial of Service attacks, in which a malicious node redirects other hosts' traffic to a victim node, and thereby creates a flood of bogus traffic at the victim host.
- Spoofing: spoofing attack happens when a malicious node impersonates another device or user on a network by using the victim node's address or identifier. Spoofing attacks could be used to launch attacks such as man-in-the-middle (MITM), steal data, create DoS attacks, spread malware, and abuse the trust relation between legitimate nodes. The NDP does not provide mechanisms for authenticating the source or destination of a message so it is possible for attackers to generate crafted IPv6 packets with spoofed source addresses.
- Replay: All Neighbor Discovery and Router Discovery messages are prone to replay attacks, in this attack, an attacker would capture valid messages and replay them later. The attacker will use these captured messages and replay them even if they were cryptographically protected. If the messages were not cryptographically protected then the attacker can change the content of the captured messages and take over the traffic flow between two hosts.
- Redirect: redirect attack takes place when a malicious node redirects packets away from router or other legitimate receiver to another node on the link. This attack can be used for DoS purposes by having the node to which the packets were redirected drop the packets.
- Rogue router: this attack happens when a malicious node acts as a router, it advertise itself as a last-hop router to make hosts select the attacker as its default router, then the

attacker acts as an MITM and effectively receive, drop, or replay the packets. An attacker can configure a rogue router on an unsecured link easily, but a node cannot easily distinguish between a fake and authorized router.

For the privacy implications, Neighbor Discovery Protocol suffers from serious privacy issues. NDP and SLAAC provides autoconfiguration facilities to generate IPv6 interface identifier (ID) based on MAC address using Modified EUI-64 format, which result in a static constant ID. This ID will not change with time or in different networks, so it is easily for an attacker to track the node with specific ID or capture the traffic related to that node. However, there is a solution for this privacy issue, it was introduced in RFC 4941, “Privacy Extensions for Stateless Address Autoconfiguration in IPv6” which generate global scope addresses from IDs that change over time. But it does not protect against the security threats. The Internet Engineering Task Force developed SEND as a countermeasure to NDP security vulnerabilities. But it also solves the privacy problem.

## 2.3 SEcure Neighbor Discovery (SEND)

SEcure Neighbor Discovery (SEND) is a mechanism that enhances the Neighbor Discovery Protocol. It is considered as a security extension of the NDP. Since Neighbor Discovery Protocol is insecure and vulnerable to various attacks, the IETF developed SEND to provide a mechanism to secure NDP with a cryptographic method that is independent of IPsec.

SEND provides some features to Neighbor Discovery Protocol to secure its various functions; these features are router authorization mechanism, address ownership proof mechanism and message protection. SEND uses the five NDP messages with new options *CGA*, *RSA signature*, *nonce* and *timestamp*. It also adds two new ICMPv6 messages used for router authorization; these two messages are certificate path solicitation (CPS) type 148 and certificate path advertisement (CPA) type 149.

### 2.3.1 Router Authorization

SEND introduce an authorization delegation discovery (ADD) process to authenticate routers, its solution is based on Certificate paths, anchored on trusted parties. Before a host adopts the

router as its default router, the host must be configured with a trust anchor to which the router has a certification path. When a node receives a protected Router Advertisement (RA) message, it must check if there is a certification path available. And if there is no one, then it must trigger the authorization delegation discovery process. SEND uses Certification Path Solicitation (CPS) and Certification Path Advertisement (CPA) messages to discover a certification path to the trust anchor without requiring the actual Router Discovery messages to carry lengthy certification paths.

The Router authorization process can be described as follow:

- 1- The router should be authorized to act as a router; it obtains a router certificate from a trust anchor.
- 2- Router sends protected Router Advertisement message, host sends CPS message to ask the router to provide a valid certificate.
- 3- The router responds with CPA message that contains the certificate.
- 4- The host verifies router legitimacy, and accepts the router as the default router if it is valid.

### 2.3.2 Address Ownership Proof Mechanism and Protected messages

SEND uses RSA cryptosystem, which is one of the widely used public key cryptosystems. A public-private key pair is generated by all nodes before they can claim an address. SEND uses the *CGA* option to carry the public key and associated parameters in order to make sure that the sender of a Neighbor Discovery message is the owner of claimed address. SEND also uses *Timestamp* and *Nonce* options to prevent replay attacks.

#### 2.3.2.1 Cryptographically Generated Address (CGA)

CGA is an IPv6 address that is bound to a public signature key. The Interface Identifier (ID) of this address is generated by computing a cryptographic one-way hash function from public key and auxiliary parameters. Binding the IPv6 address of a node to its public key prevents address stealing and authenticates the IPv6 address without requiring third-party or additional security infrastructure. A receiving node can verify the sender address by re-computing the

hash value using the CGA parameters carried with the CGA option in the SEND messages and compare this hash value with the Interface Identifier of the sender address.

SEND protects messages sent from IPv6 address by attaching the public key and auxiliary parameters and by signing the message with the corresponding private key. Valid CGAs can be generated by any sender, including a potential attacker because CGAs themselves are not certified. However, they cannot use any existing CGAs; an attacker cannot take a CGA created by someone else and send signed messages that appear to come from the owner of that address.

The CGA is associated with a set of parameters that consist of a public key, auxiliary parameters and a security parameter (Sec) that determines its strength against brute-force attacks. The Sec parameter is a three-bit unsigned integer encoded in the three leftmost bits of the Interface Identifier. The CGA parameters are used to compute two hash values, Hash1 (64 bits) and Hash2 (112 bits).

Cryptographically Generated Address (CGA) parameters are Modifier (128-bit), Subnet Prefix (64-bit), Collision Count (8-bit), Public Key and Extension Fields. The process of generating a new CGA takes the subnet prefix, the public key, and the security parameter (Sec) as input to the algorithm. The cost of generating a new CGA depends exponentially on the security parameter Sec.

The process starts by setting the modifier to a random or pseudo-random 128-bit value. Then compute hash2 value which is an SHA-1 hash value over the entire CGA parameters. The hash2 computation loop continues until finding the final modifier, the one that makes  $16 \times \text{Sec}$  leftmost bits of hash2 equals zero. The final modifier is saved and used as an input for hash1 computation. The Interface Identifier is formed from the hash1 value which is the 64 leftmost bits of SHA-1 hash value. The Sec value is encoded in the three leftmost bits, and seventh and eighth bits from left are set to zero [7].

The CGA's computational cost depends mainly on the Sec value, For Sec=0, the algorithm is relatively fast, for Sec values greater than zero, the CGA computations can take a long time, for example the algorithm took 1.65 Hours with Sec=2 on a Machine with 2.67-GHz CPU

[16], the most computationally expensive part of CGA algorithm is finding the final modifier to satisfy the hash2 condition. There is a possibility to avoid repeating the expensive search for an acceptable modifier value by reusing the old modifier value when the subnet prefix of the address change but the address owner's public key does not change. However, this optimization makes it easier for an observer to link two addresses to each other [6].

### 2.3.2.2 RSA Signature

The RSA Signature option is used to protect Neighbor and Router discovery messages, this option allows public key based signature to be attached to NDP messages. RSA signature protects the integrity of the messages and authenticates the identity of their sender.

The RSA option data structure contains 12-bit Type field, Length field, 16-bit reserved field, 128-bit Key Hash field, Digital Signature Field and Padding field. The Key Hash field containing the left most significant 128 bits of a SHA-1 hash of the public key used for constructing the signature. The digital signature field is variable-length field containing the first of a family standards called Public-Key Cryptography Standards (PKCS#1) signature, constructed by using the sender's private key [5].

### 2.3.2.3 Timestamp Option

The Timestamp option offers replay protection without any previously established state or sequence numbers. Its purpose is to make sure that unsolicited advertisements and redirects messages have not been replayed.

### 2.3.2.4 Nonce Option

Nonce option is used to protect the solicitation-advertisement pairs of NDP messages against replay attacks. Its purpose is to make sure that an advertisement is a fresh response to a solicitation sent earlier by the node. The option uses a random number selected by the sender of the solicitation messages to achieve the protection. The advertisement messages that respond to any solicitation message should include a matching nonce option [5].

## 2.4 Android

Android is an open source platform that was built on the Linux Kernel to offer a full set of software for mobile and tablet devices with touch screen feature. Google and the Open Handset Alliance developed the Android software platform and operating system based on the Linux operating system. Android was uncovered in 2007 and many versions have been developed since that, its first released started from Cupcake then Donut, Eclair, Froyo, Gingerbread, Honeycomb, Ice Cream Sandwich, Jelly Bean and ending with KitKat [17], it will continue growing as well as developers keep moving towards improving it.

### 2.4.1 Open Source Advantages

Google decided to release the Android open source project under version 2 of the Apache Software License (ASL). The ASL applies only to the user-space components of the Android Platform, while the underlying Linux kernel is licensed under the General Public License, version 2.0 (GPLv2).

The fact that Android is an open source platform has many advantages:

- Consumers can buy more innovative mobile devices at lower prices.
- Mobile operators can easily customize their product lines and they will be able to offer newly developed services at a faster rate.
- Handset manufacturers will benefit from the lower software prices.
- Developers will be able to produce new software more easily, because they have full access to the system source code and extensive API documentation.

### 2.4.2 Architecture overview

Android is a software stack for mobile devices that includes an operating system, middleware and key applications. The architecture components of Android are Linux Kernel, Libraries, Android Run Time, Application framework and Applications [26] as shown in figure 2.4.

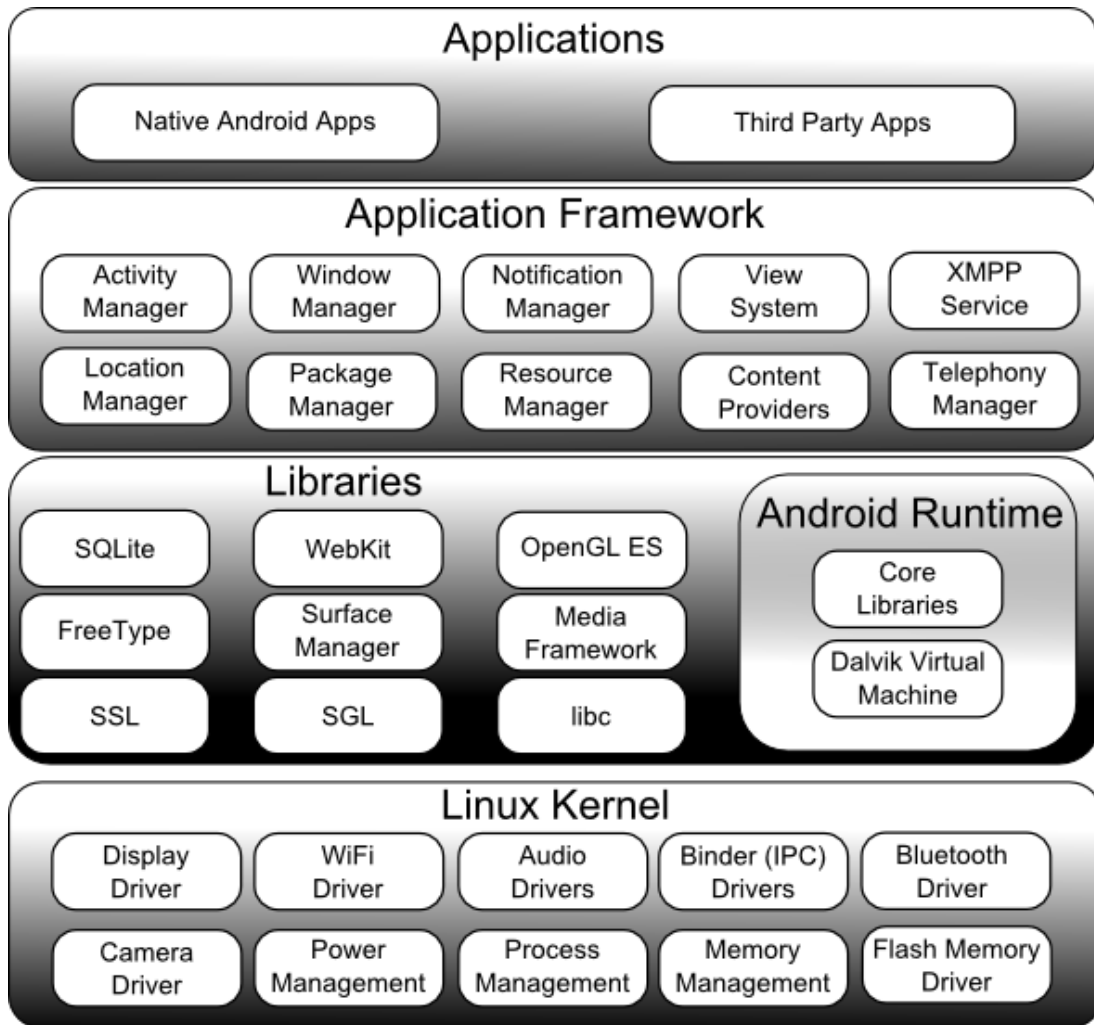


Figure 2.4: Android software stack Architecture [27]

- Linux Kernel

Android uses the Linux 2.6 Kernel. Google decided to use Linux as Hardware Abstraction Layer because of its device drivers, security, memory management, process management, and network stack.

- Libraries

These are Android native libraries, which are written in C/C++ and provide most of the core power of the Android platform. This level is composed of several different components such

as the Surface Manager (for compositing windows), 2D and 3D graphics, Media framework (MPEG-4, H.264, MP3, etc.), the SQL database (SQLite), and a native web browser engine (WebKit).

- Android Run Time

Android run time includes the Dalvik Virtual Machine. Dalvik runs dex files, which are byte codes resulting from converting jar-files at build time. Dex files are more compact and efficient than class files, an important consideration for the limited memory and battery powered devices that Android targets. The Android run time also includes Core Libraries, which contains all collection, classes, and utilities.

- The Application Framework

This framework is written in Java and it is the toolkit that all applications use. The most important component of the framework is the Activity Manager, which manages the life cycle of applications. The Package Manager keeps track of the applications that are installed on the device. The Window Manager is a JPL abstraction on top of the lower level services provided by the Surface Manager. The Telephony Manager contains the API's for the phone applications. Content Providers form a framework that allows applications to share data with other applications. The Resource Manager is used to store localized strings, bitmaps and other parts of an application that are not coded. The View System contains buttons and lists. It also handles event dispatching, layout, and drawing. The Location Manager, Notification Manager and XMPP Manager are also interesting API's that are beyond our scope.

- Applications

The Application Layer contains all applications: Home, Contacts, Browser, user-written applications and so on. They all use the Application Framework.

### 2.4.3 Integrated Development Environment (IDE) and Tools

Android provides the Android Developer Tools (ADT) plug-in for Eclipse, which provides a professional-grade development environment for building Android apps. It is a full Java IDE with advanced features to help for build, test, debug, and package Android apps. The ADT



includes the Software Development Kit (SDK) which provides the API libraries and developer tools. Moreover, Android provides Native Development Kit (NDK) which is a toolset that allows implementing parts of an Android app using native-code languages such as C and C++.

The SDK comes with a bunch of tools that relieve the creation of an Android app. The most important tools are Android Asset Packaging Tool (aapt), Android Debug Bridge (adb), Dalvik Cross-Assembler (dx) and Dalvik Debug Monitor Service (ddms).

## 2.5 Related Work

SEcure Neighbor Discovery (SEND) has some real challenges and limitations in several areas including implementation and deployment, which is why most operating systems support NDP but still not support SEND.

There are some current SEND implementations for specific OS distributions, but they are basically proof of concept rather than production-ready software. None of these implementations is for Android operating system.

These implementations can be divided into two groups; the first one is the implementations that are done in the user space such as send-0.2, NDprotector, and Easy-SEND for Linux and WinSEND for Windows, and TrustRouter for both Windows and Linux. The others are done at the kernel level (send-0.3) such as “Huawei and BUPT (ipv6-send-cga)” implemented in the Linux kernel.

### 2.5.1 TrustRouter

TrustRouter is an implementation of Secure Neighbor Discovery (SEND), which could be installed and run on client’s Operating System (OS). TrustRouter is implemented on three different platforms; Mac OS X, Linux, and Windows. It concentrates on Router Advertisements (RA) securing and does not implement CGAs yet [18]. Router Advertisement messages must be secured since they are the router responses to client’s Router Solicitation (RS) that hold very important information (router address and prefix). So if any node could be in between and claim to be the trusted router and send Router Advertisement (RA) as a reply

for client's Router Solicitation (RS), client will start sending traffic to it allowing loss of information and security violations [19].

#### 2.5.1.1 Working Mechanism

TrustRouter implementation takes Router Advertisement (RA) from OS network stack to be checked before reaching the Operating System (OS) to start autoconfiguration process, this is done using OS-specific module that it has since special APIs for capturing packages in the platforms that TrustRouter supports are not propped. OS-specific module is partly executed in the kernel mode to pass packets to their coordinate in the user mode, which invokes core module that in turn asks for Certification Path (chain of trusted certificates starting with a known trust anchor and ending with the certification of the router) and digital signature by sending Certification Path Solicitation (CPS) message to the router and waiting for router response by Certification Path Advertisement (CPA) messages each contains a certificate from Certification Path. The core module is supported by security module that execute verification process by checking the validity of the certification path first, then if valid it uses router certificate public key to validate the signature and send the result of verification back to the OS-module which in turn put the valid Router Advertisement (RA) in the network stack again to be processed by the Operating System (OS) and drop the invalid ones[20].

#### 2.5.2 EASY-SEND

Easy-SEND is an open source JAVA application that had been developed in Linux user-space as an implementation for SEcure Neighbor Discovery (SEND)[21], it does not implement router authorization[7]. It is mainly used for learning purposes. Easy-SEND verifies Cryptographically Generated Addresses (CGAs)

#### 2.5.3 NDprotector

NDprotector is an implementation of SEcure Neighbor Discovery (SEND) running on Linux platform. It uses Python and scapy6 to analysis and modifies packets. Scapy6 is a Python interpreter that enables you to create, forge, or decode packets on the network [9]. NDprotector implementation takes Neighbor discovery messages and changes its destination to the user space before they reach the kernel. After catching packets, scapy6 adds CGA

producers, certificate processing and SEND messages format to the original messages. Scapy6 stops the packets and analyze them to decide if we need to modify message by adding an RSA signature using the private key of the address that the message comes from or let the message with correct signature pass [8].

## 2.6 Chapter summary

In this chapter we discussed Internet Protocol Version 6 (IPv6) in section 2.1, Neighbor Discovery Protocol (NDP) in section 2.2, Secure Neighbor Discovery Protocol (SEND) in section 2.3, Android in section 2.4, and related works in section 2.5.

## Chapter 3: SEND Implementation on Android

In order to implement IPv6 SEND mechanism on a mobile device, we need to fully understand the IPv6 in general such as IPv6 Addressing and routing, Address Types, header formats, The Neighbor Discovery and the Secure Neighbor Discovery.

The first task for implementing SEND on Android platform is to capture, analysis and filter IPv6 Neighbor Discover messages arrived or transmitted between the Android device and other nodes in a network. The second task is to handle these captured packets by adding or verifying the SEND options.

### 3.1 Capturing IPv6 Packets on Android

Since our implementation works on user-space level and does not modify the kernel level, we need to capture IPv6 packets from the device network stack and send them to our software application. In order to capture the packets we use the *Netfilter* framework, which is a packet filtering framework inside the Linux Kernel; it provides a set of hooks that allow kernel modules to register callback functions with the network stack [22]. This framework with the *ip6tables* kernel module gives the possibility to get IPv6 network packets from kernel into user mode. The *ip6tables* is a table-based system for defining rule sets; each rule defines what packets to capture and what action to do with them [23]. The captured or matched packets are put into the *NFQUEUE* where they can be accessed from user mode through the *libnetfilter\_queue* C-language API. The *libnetfilter\_queue* main features are receiving queued packets from the kernel and issuing verdicts or re-injecting altered packets to the kernel [24].

So the first part of our software is a C-language code that is responsible of capturing packets. We used the Android NDK toolkit to run the *libnetfilter\_queue* API on Android. The Android NDK (Native Development Kit) is a toolset that allows you to implement parts of your app using native-code languages such as C and C++ [25]. NDK allows putting C code in the Android app.

We needed to write an executable C file to work as a firewall between the network interface card and IPv6 stack because we could not find a Java library that can capture the packets and works on Android device. The *libnetfilter\_queue C library* is the only one it worked. We should mention that we tried the *Virtual Services IPQ* [28] or *VServ IPQ* for short, which allows Java users to achieve the same functionalities in *libnetfilter\_queue library*, we compiled *VServ IPQ* using Android NDK but it did not work on the Android devices, because it required a queue target that is not supported by the Android devices.

### 3.1.1 The C-language Code Functionalities

The C executable file first adds the *ip6table rules* to the kernel, we added rules to match the five Neighbor Discovery messages, Figure 3.2 shows the ip6table rules that we added to the kernel. Then the C executable file opens the *netfilter queue handler* and binds it to the protocol family *AF\_INET6*, and then it creates a queue and registers a callback function to be called when a packet matches the ip6table rules. The callback function which works on separated thread receives the packet as a parameter and writes its content on a buffer to send it to the second part of our application to handle the packet and decide whether to accept or drop it. The C-language code functionality is shown in figure 3.2.

```
368
369  int retRS =
370      system(
371          "su -c \"ip6tables -A OUTPUT -p icmpv6 --icmpv6-type 133 -j NFQUEUE --queue-num 19\""); // RS
372  int retRA =
373      system(
374          "su -c \"ip6tables -A INPUT -p icmpv6 --icmpv6-type 134 -j NFQUEUE --queue-num 19\""); // RA
375
376  int retNS =
377      system(
378          "su -c \"ip6tables -A OUTPUT -p icmpv6 --icmpv6-type 135 -j NFQUEUE --queue-num 19\""); // NS
379  int retNA =
380      system(
381          "su -c \"ip6tables -A INPUT -p icmpv6 --icmpv6-type 136 -j NFQUEUE --queue-num 19\""); // NA
382
```

Figure 3.1: ip6table Rules

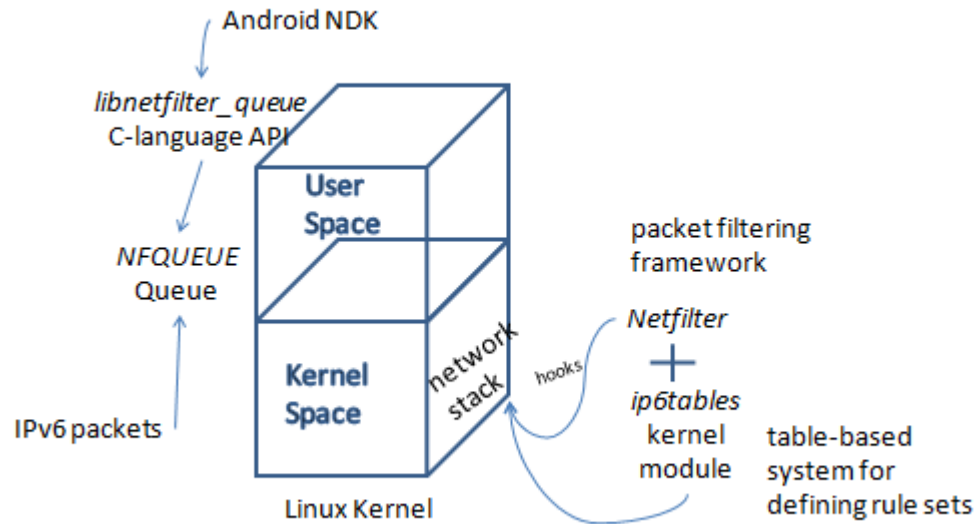


Figure 3.2: The C-language code functionality

### 3.2 Handling Captured IPv6 Packets

The second part of our software is an Android Java Application; the main objective of the Android Application is to handle the captured IPv6 packets. The application adds and verifies the four SEND options in the packets.

The Android application has classes for creating and validating the four SEND options (*CGA*, *RSA*, *Timestamps* and *Nonce*). Also it has classes for handling IPv6 Packets, ICMPv6 Packets and ND Messages. There are also Android user interface classes which shows logging information to give the user indication of what is happening in the background.

The *IP6Packet* class wraps the raw bytes comprising an IPv6 packet; the class declares the offset of IPv6 header fields such as version and header length. It also declares a function to compute the checksum. The *ICMP6Packet* class inherits the *IP6Packet* class to handle ICMPv6 packet. Also, there is the *NDMessage* Class which inherits the *ICMP6Packet* class to implement Neighbor Discovery Protocol and add the Secure Neighbor Discovery options.

The classes *CGA*, *RSASignature*, *TimeStamp* and *Nonce* all inherits the *SENDOPT* class that wraps the bytes comprising a SEND option. These classes contain the methods to generate and verify the SEND options.

We declared all the parameters in the class *SendParameters* such as the key length, the Sec value and the timestamp delta. These parameters will be changeable by the application user through the user interface classes. The secret parameters such as private key are stored in a private file in the internal memory of the mobile device, which could not be accessed through the user or other applications on the device. We used the *Files class* to handle reading and writing these parameters to the internal memory.

We also declared a class to simulate the CGA algorithm and to generate and verify CGA addresses. The class contains a method that takes the Sec value, the subnet prefix and the CGA parameters to generate a valid CGA address.

We used *Util and Inet6Util utility class* that contains a number of methods for handling bytes, and strings. They contain also general methods for the calculation of the checksum and the generation of Hashes.

### 3.3 SEND Software Structure

To describe in more details the software we built, we will show use case diagram, class model diagrams (UML) and component relationship model.

#### 3.3.1 Use Case Diagram

The use case model in figure 3.3 shows the different use cases implemented in the SEND software we built. There are mainly five use cases described below.

1- Choose SEND Parameters.

The software gives the Android Application user the ability to choose among different SEND parameters. These parameters are *Sec Value* which determines the security level required, *Key Length* of the RSA key pair, *TimeStamp Delta Value*, *TimeStamp Fuzz Factor* and *TimeStamp Clock Drift Value*.

The Main Activity Class shows the main page of the application which contains an option to set these parameters.

Figure 3.4 shows the main application page, and Figure 3.5 shows the SEND Parameters Page.

2- Test CGA Computational Cost.

This use case gives the user the ability to test CGA computational cost for different Parameters such as *Sec Value* and *Key Length*. The Test CGA option is shown in the

main application page. User can click this option to open Test CGA Page which is shown in Figure 3.6. The Test page allows a user to set the parameters value and click run, then the application will start CGA algorithm and shows Logging information such as the required time in millisecond and the CGA Parameters.

3- Capture Packets.

This is the main use case that runs in the background, it is responsible of capturing Packets from the network stack and sending them to the local socket of our application. This use case is initiated when the user clicks *Start SEND Service* option from the main Application page.

4- Add SEND Options For Outgoing Packets.

This use case handles the outgoing packets by adding the four SEND options (*CGA, RSA, Timestamp, and Nonce*) to the packet.

5- Verify SEND Options For Incoming Packets.

This use case checks and verifies the SEND options in the incoming packets, and decides to accept the packet or drop it.

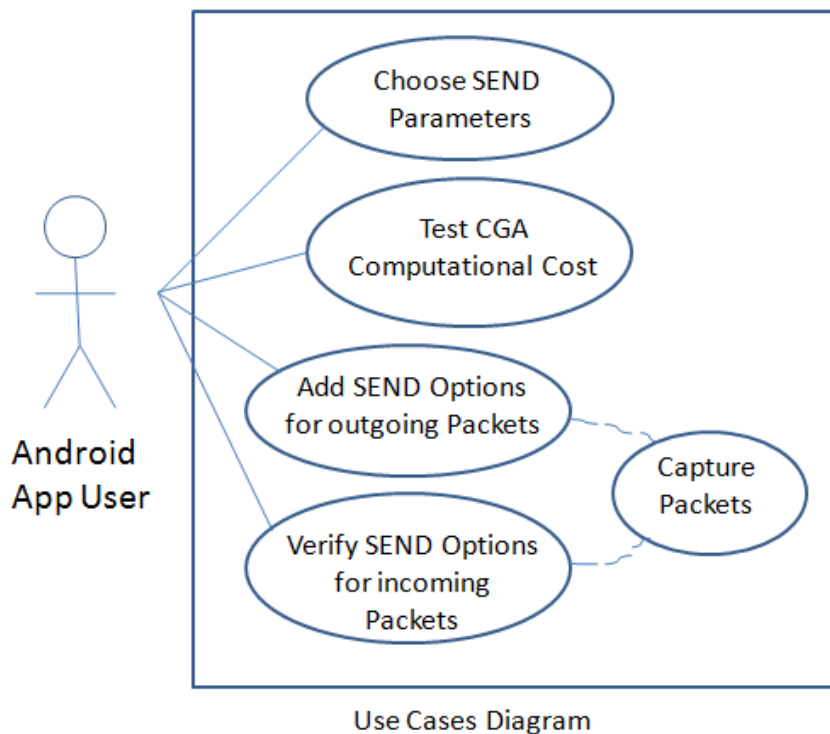


Figure 3.3: Use Case Diagram of our SEND software.



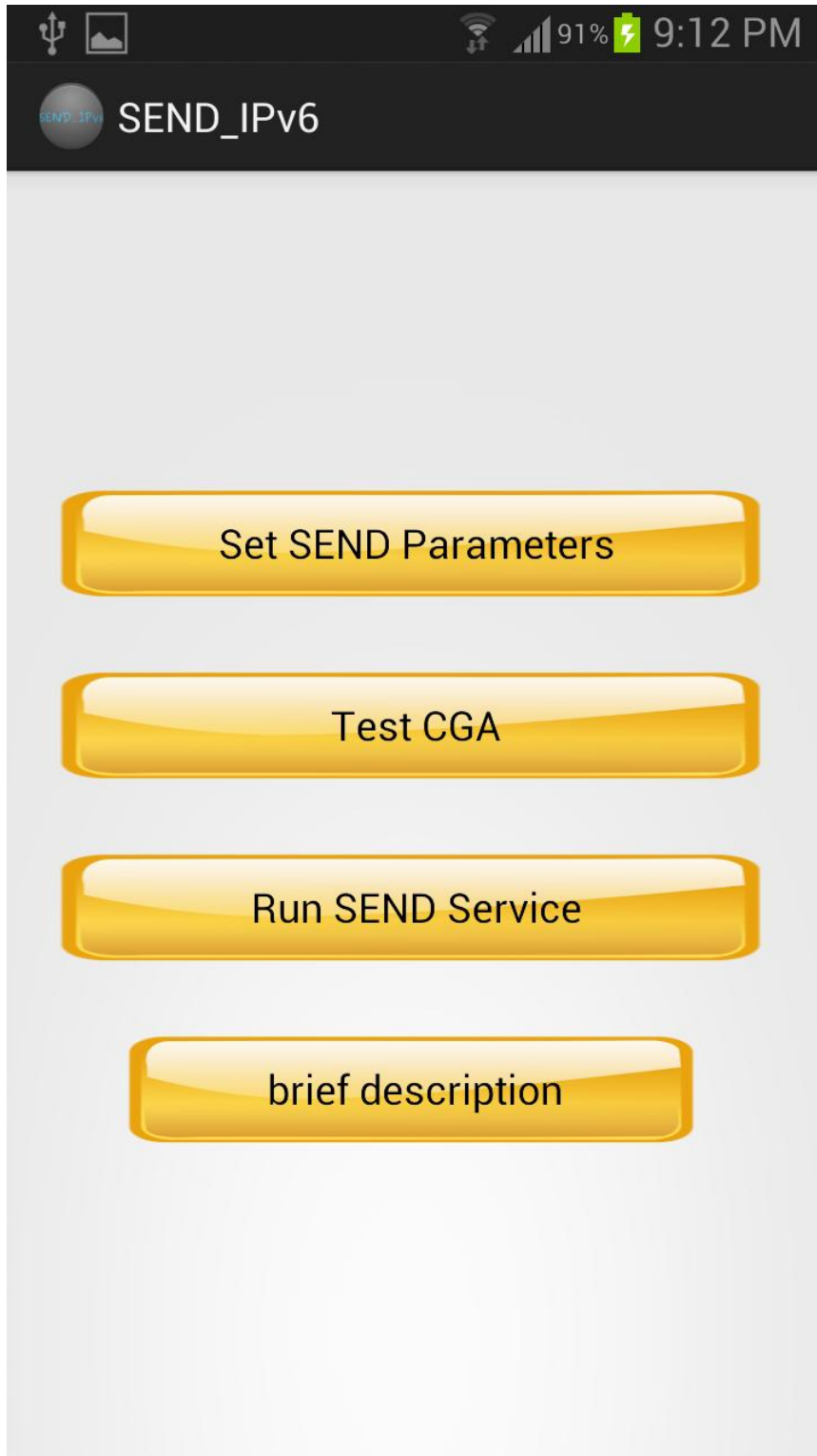


Figure 3.4: Application main Page

SEND\_IPv6 SEND\_IPv6

CGA Parameter

Sec Value

Timestamp Parameters

delta Value

fuzz Value

dift Value

RSA Key Parameter

Key Length

Figure 3.5: Application SEND Parameters Page

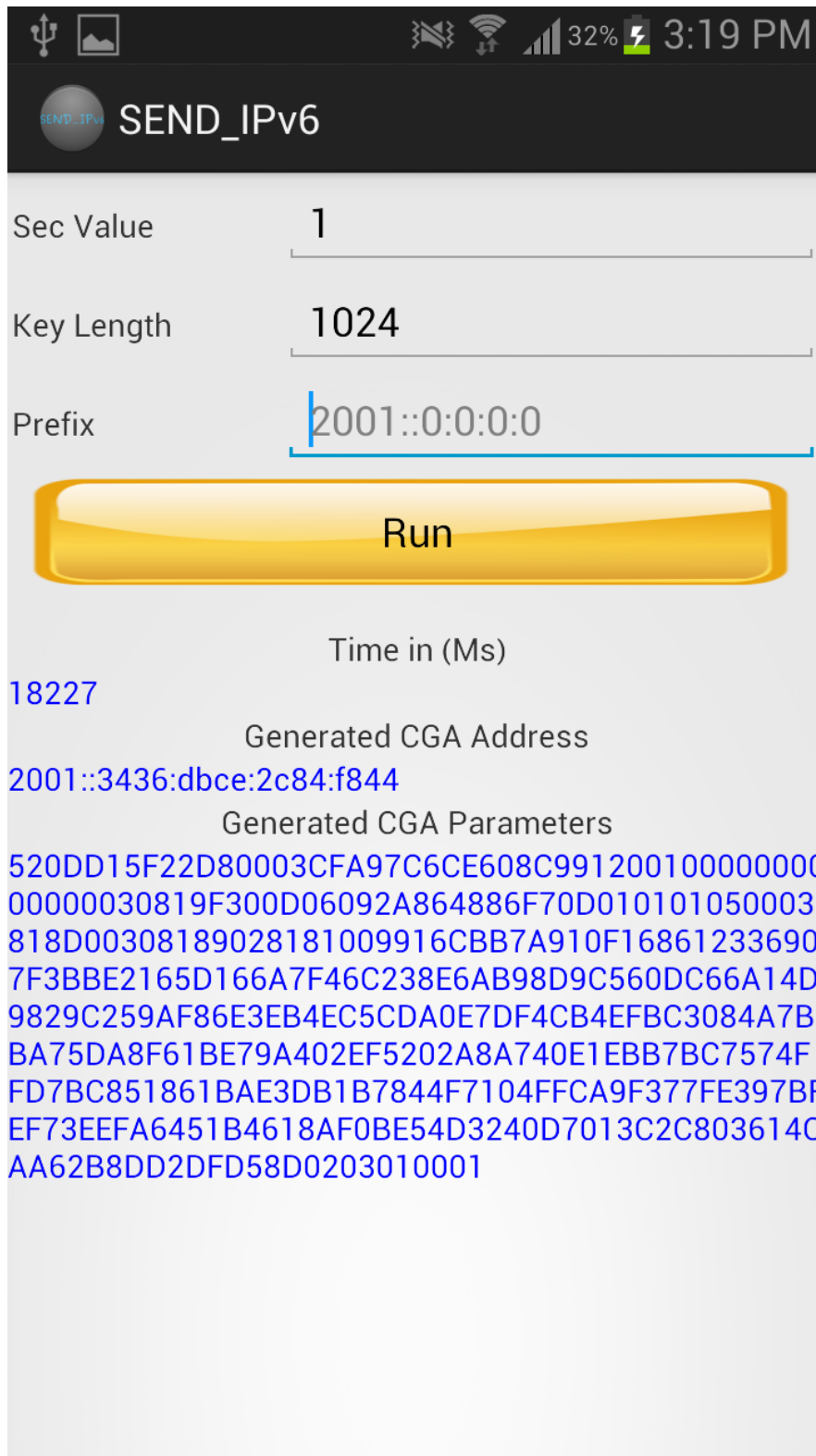


Figure 3.6: Application Test CGA Page

### 3.3.2 Class Model Diagram (UML)

The UML diagrams in Figure 3.7 and figure 3.8 show the main classes used and their relationships. The UML diagram in Figure 3.7 shows the classes which are responsible of the four SEND options. The UML diagram in Figure 3.8 shows the classes which are responsible of handling the captured packets.

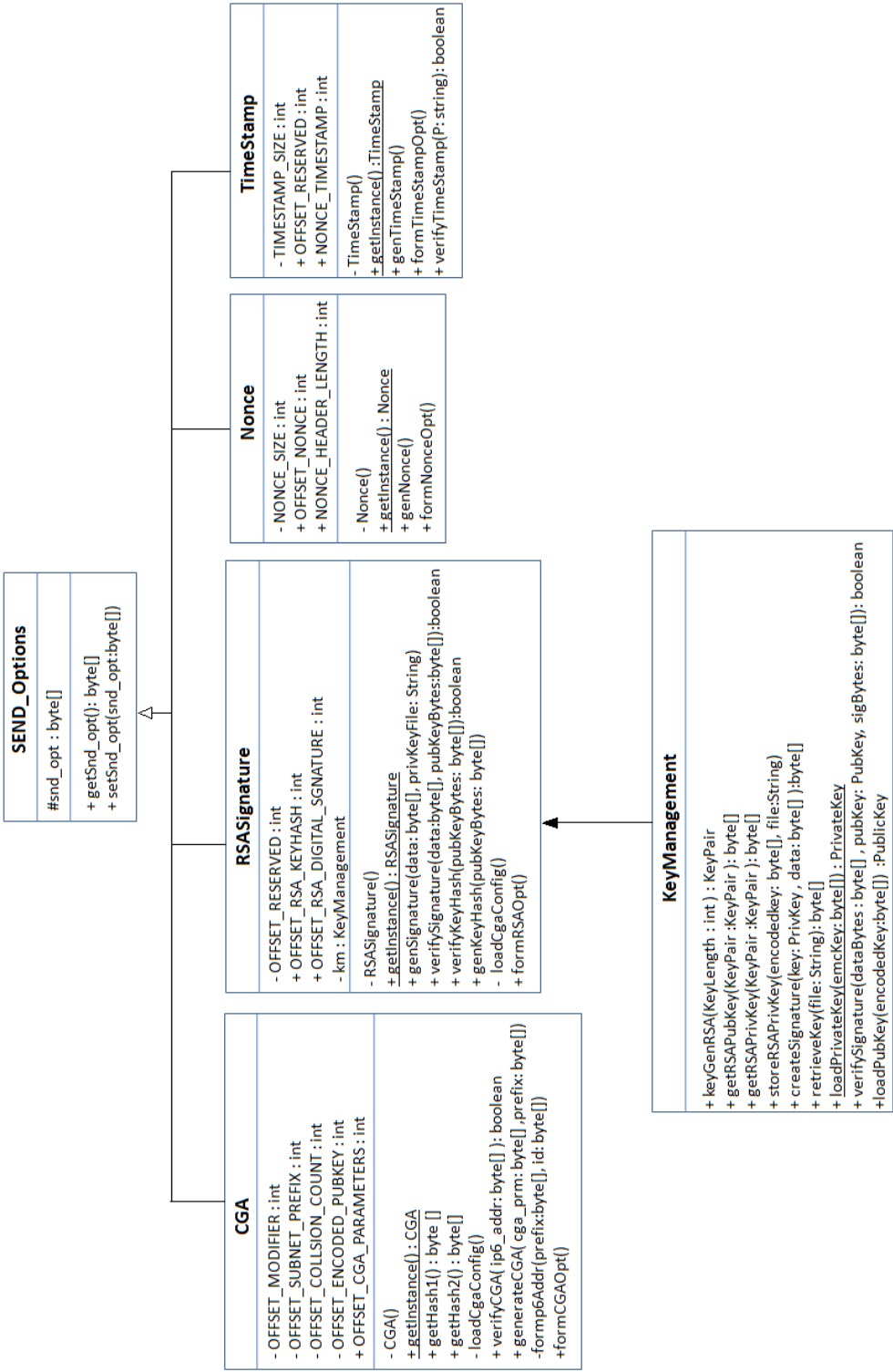
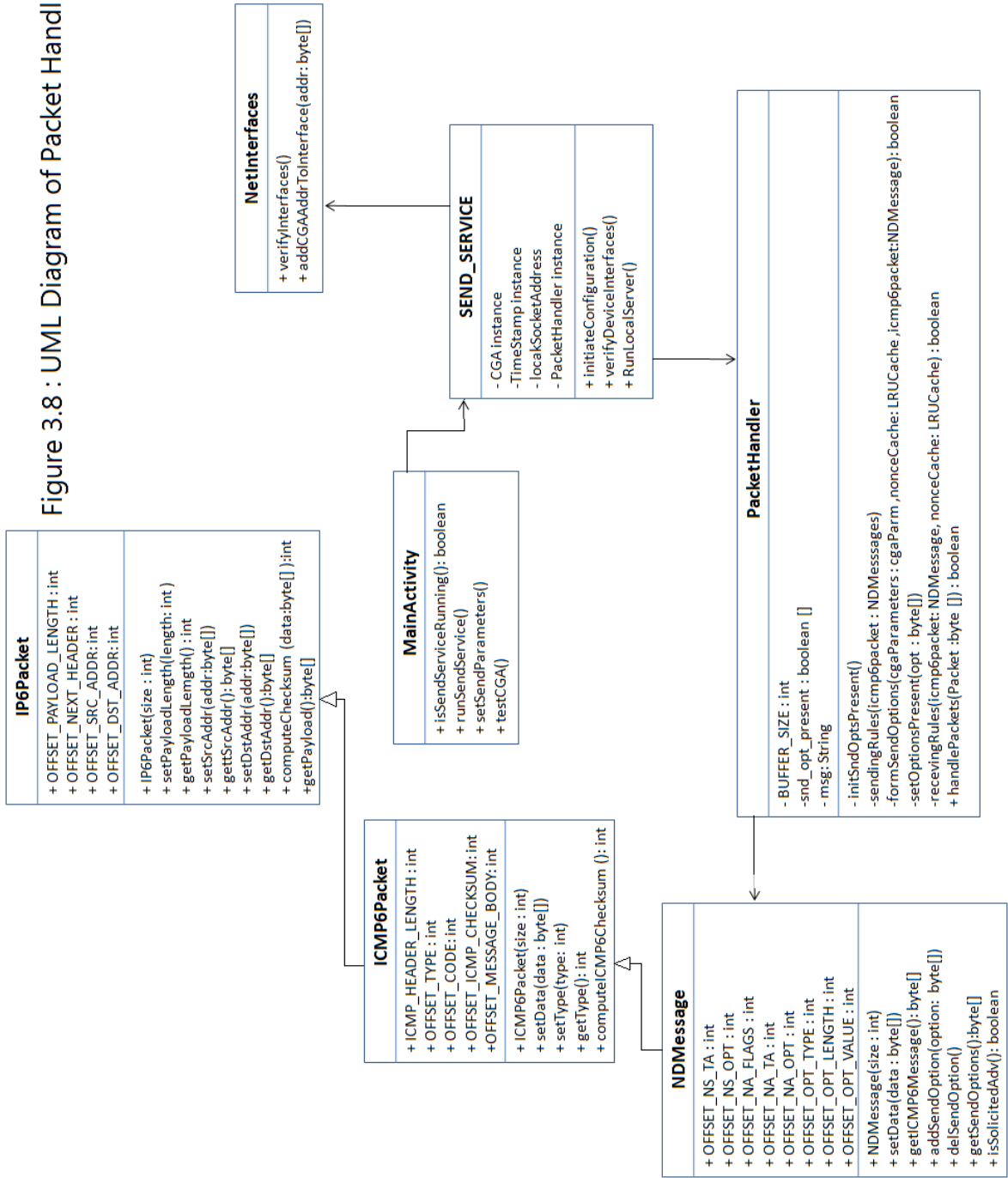


Figure 3.7 : UML Diagram of SEND Option Classes

Figure 3.8 : UML Diagram of Packet Handling Classes



### 3.3.3 Component Relationship Model

Our software contains two main components which are the Java Android application and the C Executable file as shown in Figure 3.9. The Java component will start the process when user decides to run the SEND Service.

At First, the Java Application will initiate the configuration class to get the SEND parameters. Then creates an instance of the CGA class, and checks the interfaces on the device. The application will search for any interface with link-local IPv6 address and generate an equivalent CGA address with the same prefix. The application will replace the original IPv6 Addresses with the new generated CGA addresses. After that, the application will create the local server to communicate with the C executable file.

Secondly, the Java application will run the executable C file. The executable C file will add the *ip6table rules* to the kernel and will create and open the *netfilter queue handler* to start capturing packets that matches the provided rules.

The Last step is to communicate between the executable C file and the Java application, The executable C file will sends the captured packets to the local socket created by the Java application, the java application will handle the packets and send a response to the executable C file to accept the packet or drop it. In order to increase the performance we used threads to control the communication process, the executable C file does not to wait for a response of a captured packet before capturing the second one. Each socket transition is accomplished in a new thread.

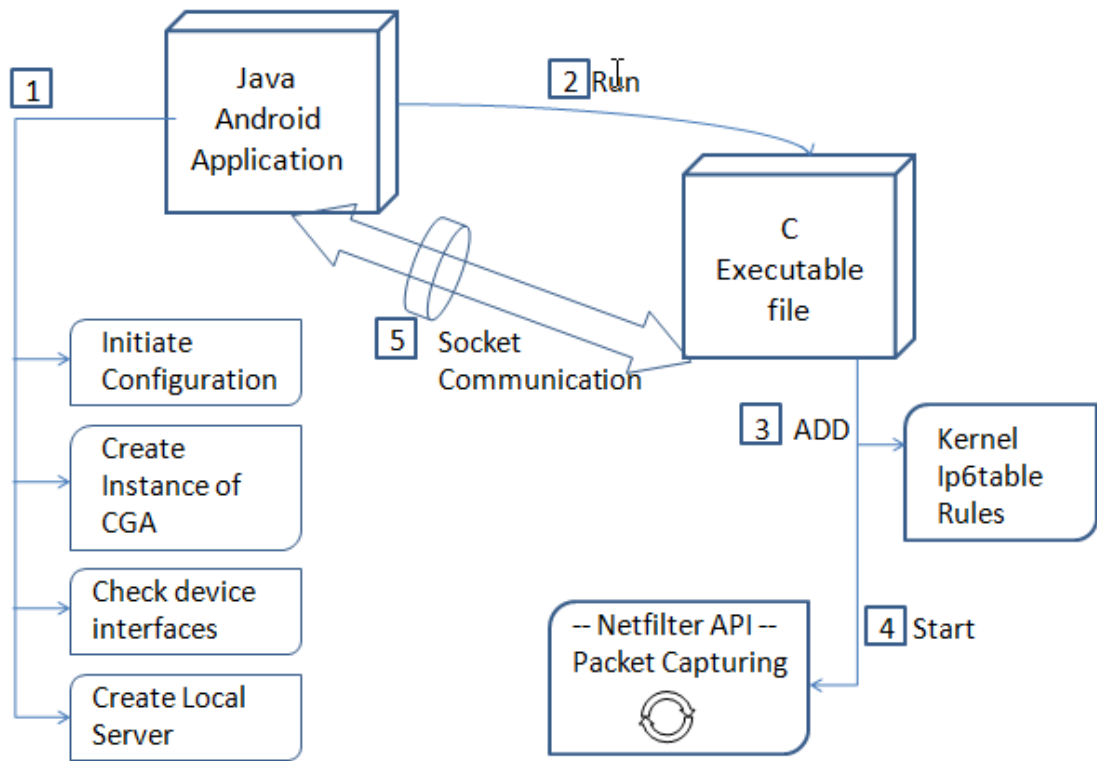


Figure 3.9: Component Relationship model

### 3.4 Problems Faced and Constraint

During our progress we faced a lot of problems listed as below:

- Android platform is built upon Linux platform, and you can run Linux commands on Android if you have a root access to the Android kernel. To have such an access and permission to control internal system's files, the device must be rooted. Rooting a device will void its company guarantee and put the device in risk if anything went wrong during the rooting process.
- Other implementations of SEcure Neighbor Discovery (SEND) such as TrustRouter and NDProtector use Python for their implementations, they used the *nfqueue-bindings* project that aims to provide access to *libnetfilter\_queue* in high-level languages like Python .Android applications can be built using python language. We tried to implement our project using python but the *nfqueue-bindings* project libraries did not run upon Android.



- Implementing parts of the Android application using a native code language such as C is not a strait forward process, and it is not recommended to be used unless it is a need. The NDK toolkit which is used to compile the native code has some drawbacks, and does not generally result in a noticeable performance improvement, but it is always increases the application complexity, as noted in the official Android developer website [25].
- To test the SEND application we created a small wireless IPv6 network in the university lab. We configured a router to provide an IPv6 network and thus to announce a prefix to hosts by sending RA messages. When we tried to make the Android device connect to the wireless network, the device tried to connect and shows a message “Obtaining IP address” but it never connected. This is because Android devices are hardcoded to connect to IPv4 networks [29]. And it never connects to an IPv6-only network. We searched the internet and find a method to force the android device to connect to IPv6 network. The method worked fine but we had to configure the device to use static Address. The device connected and our application starts capturing packets, but we could not force the device to change its static IP address to the new Generated CGA address.

### 3.5 How to Run the Application

In order to run the Android application on an Android device, you should first build the executable C file on a Linux operation system, then copying the generated executable file to a folder in the Android device. Finally, you install the Android application and run it.

You should make sure of the next notes in order for the application to run successfully:

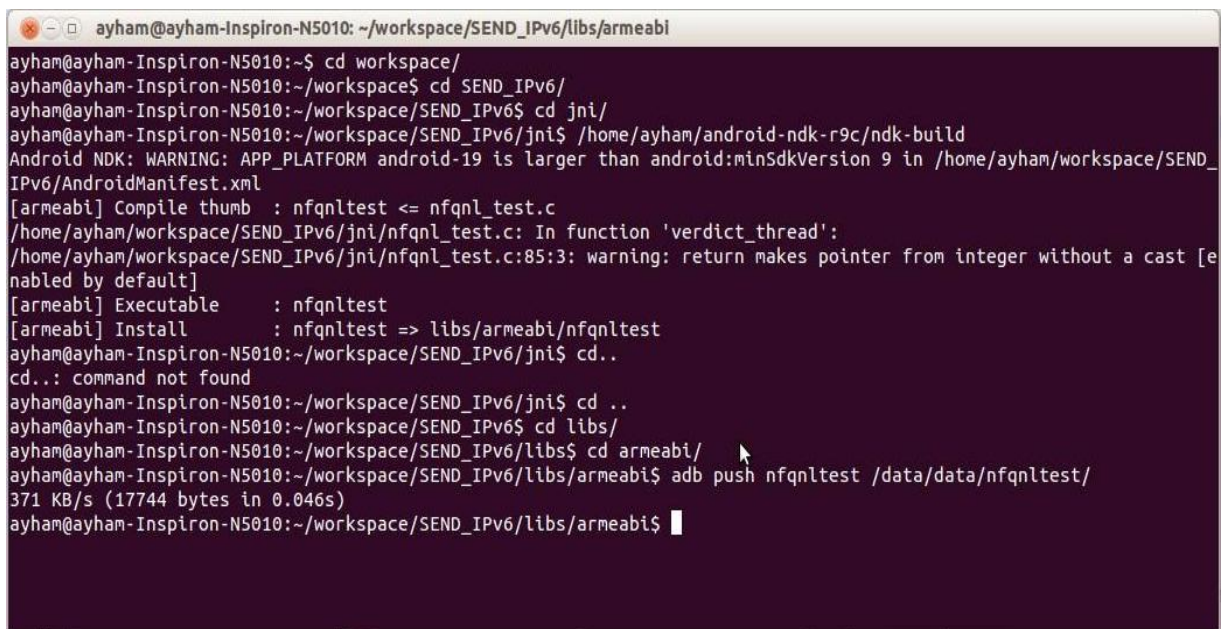
First, you will need to check if your Android system kernel is compiled with support for *libnetfilter\_queue*. Follow the following steps,

- Connect your Android device to your computer.
- Enter command “adb pull /proc/config.gz” to get the config.gz file from your Android device.
- Extract config.gz file, you’ll get a file named config. This is actually your Android Linux kernel build configuration file.
- Search for *CONFIG\_NETFILTER\_ADVANCED*, *CONFIG\_NETFITLER\_NETLINK* and *CONFIG\_NETFILTER\_NETLINK\_QUEUE* in config file, make sure they’re not commented out.

If your Android build is not complied with these features, you’ll need to compile customized kernel build to use *libnetfilter\_queue*.

The Android device should be rooted, and has `iptables` program with support for the `NFQUEUE` target.

To compile and build the executable C file, go to `jni` folder under the application directory and execute the command “`ndk-build`”, this will use the make file named “`Android.mk`” to compile and generate the executable file. To copy the file to the Android device, go to the `/libs/armeabi` folder and execute the command “`adb push nfqnltest /data/data/nfqnltest/`” which will copy the executable `nfqnltest` file to the correct folder in the device. Figure 3.10 shows the terminal window after executing these commands.



```
ayham@ayham-Inspiron-N5010: ~/workspace/SEND_IPv6/libs/armeabi
ayham@ayham-Inspiron-N5010:~$ cd workspace/
ayham@ayham-Inspiron-N5010:~/workspace$ cd SEND_IPv6/
ayham@ayham-Inspiron-N5010:~/workspace/SEND_IPv6$ cd jni/
ayham@ayham-Inspiron-N5010:~/workspace/SEND_IPv6/jni$ /home/ayham/android-ndk-r9c/ndk-build
Android NDK: WARNING: APP_PLATFORM android-19 is larger than android:minSdkVersion 9 in /home/ayham/workspace/SEND_IPv6/AndroidManifest.xml
[armeabi] Compile thumb   : nfqnltest <= nfqnl_test.c
/home/ayham/workspace/SEND_IPv6/jni/nfqnl_test.c: In function 'verdict_thread':
/home/ayham/workspace/SEND_IPv6/jni/nfqnl_test.c:85:3: warning: return makes pointer from integer without a cast [enabled by default]
[armeabi] Executable     : nfqnltest
[armeabi] Install        : nfqnltest => libs/armeabi/nfqnltest
ayham@ayham-Inspiron-N5010:~/workspace/SEND_IPv6/jni$ cd..
cd..: command not found
ayham@ayham-Inspiron-N5010:~/workspace/SEND_IPv6/jni$ cd ..
ayham@ayham-Inspiron-N5010:~/workspace/SEND_IPv6$ cd libs/
ayham@ayham-Inspiron-N5010:~/workspace/SEND_IPv6/libs$ cd armeabi/
ayham@ayham-Inspiron-N5010:~/workspace/SEND_IPv6/libs/armeabi$ adb push nfqnltest /data/data/nfqnltest/
371 KB/s (17744 bytes in 0.046s)
ayham@ayham-Inspiron-N5010:~/workspace/SEND_IPv6/libs/armeabi$
```

Figure 3.10: Linux commands to compile and copy the C executable file to Android device

## Chapter 4: Results and Conclusions

We implemented SEND on Android; we could capture the IPv6 packets from the kernel and send them to user space, where our Android application works on filtering and modifying them. We added all SEND options to achieve proof of address ownership and message protection. Figure 4.1 and 4.2 shows the *LogCat* output in the Java ADT eclipse for both C executable file and Java application respectively. The *LogCat* figures show the captured packets data, and some extra logging information.

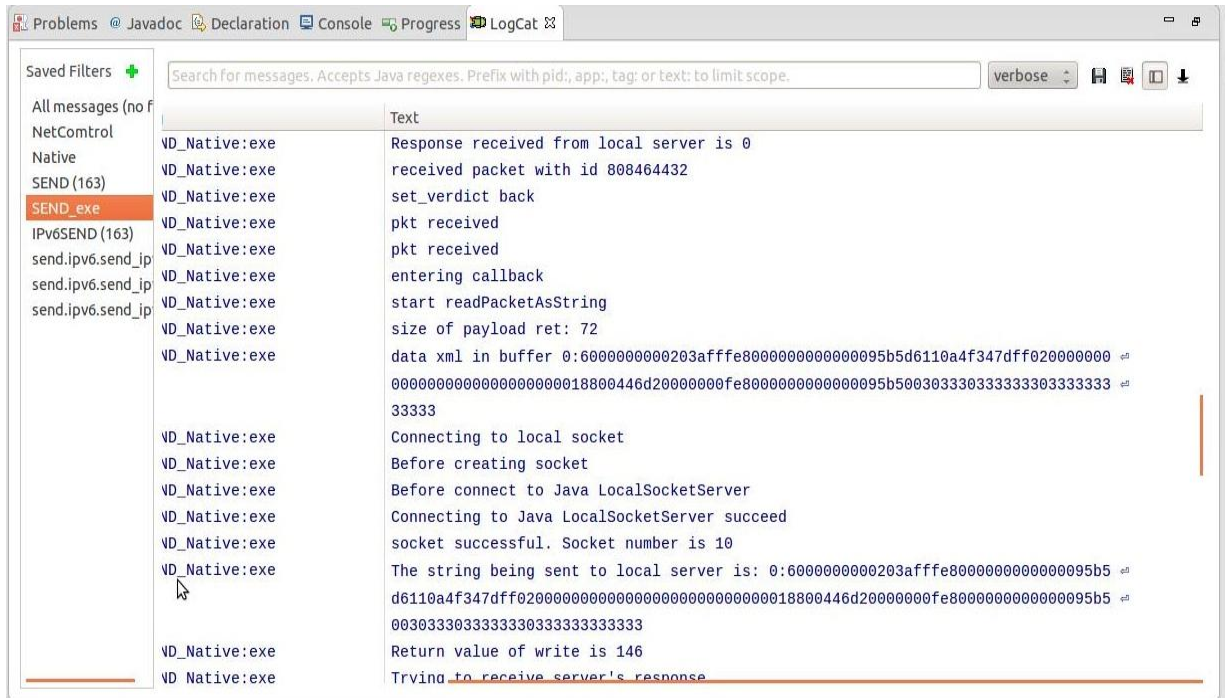


Figure 4.1: eclipse LogCat output for running C executable file

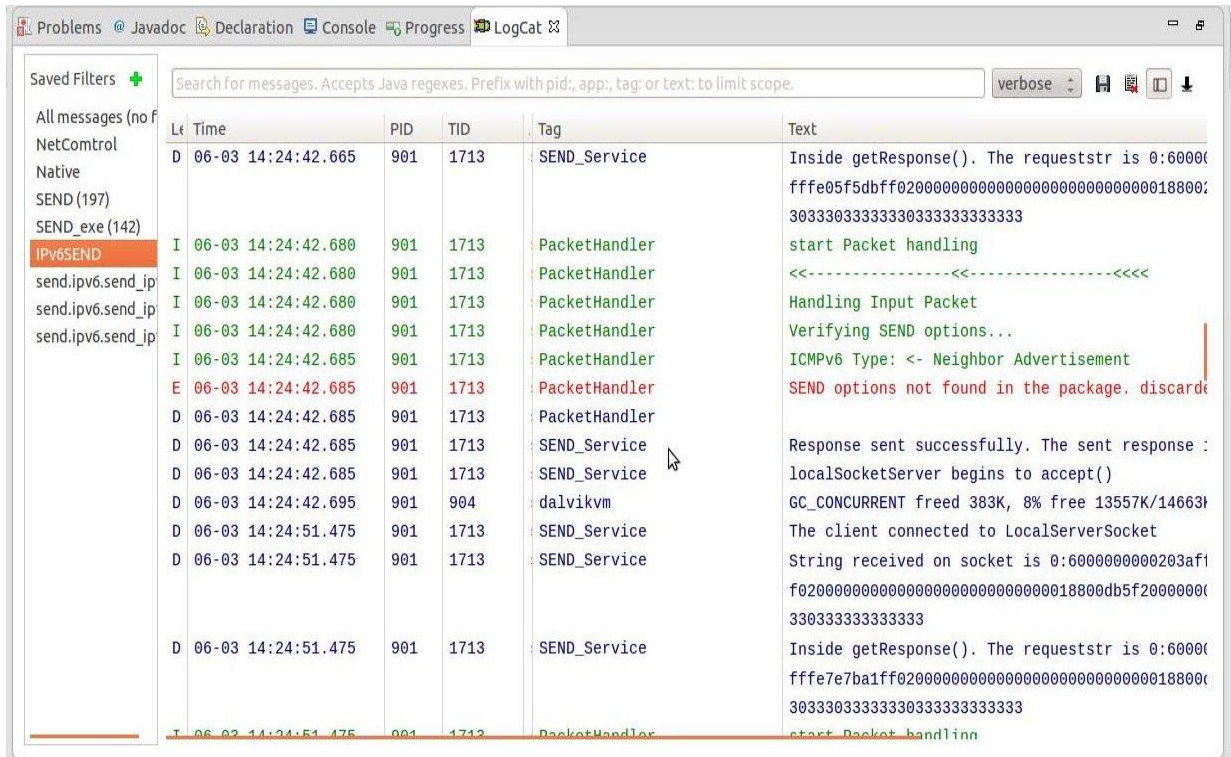


Figure 4.2: eclipse LogCat output for Android Java application

## 4.1 Results

We will provide a CGA computational cost results and running the application results.

### 4.1.1 CGA Computational Cost Results

We tested the CGA algorithm Class on a dual core processor Samsung Galaxy Tab3 Android device with 1.2 GHz CPU speed. We called the function that creates the CGA addresses with different parameters and calculated the time it took to generate the address. Table 3.1 shows the result we got. We repeated the test fifty times for each case and calculated the average time. Figure 4.3 shows a snapshot from the application during CGA computational cost testing.

Test #	Sec value	Key Length	Avg. Time in Millisecond
1	0	1024	500 msec
2	0	2048	2428 msec
3	1	1024	211961 msec
4	1	2048	3317456 msec

Table 3.1 CGA Address Algorithm, average generation time

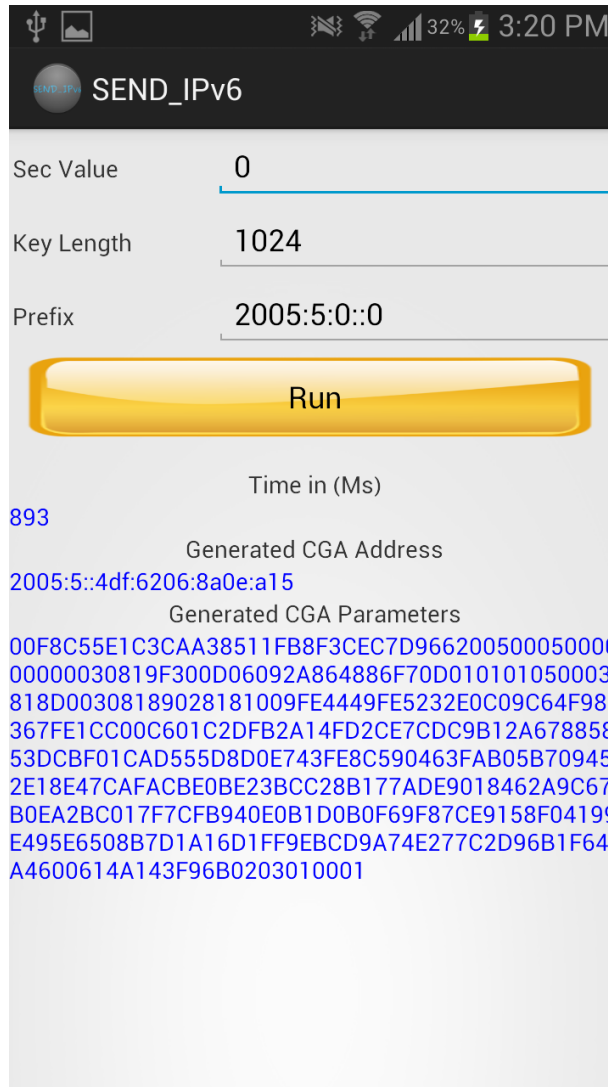


Figure 4.3: snapshot from CGA computational cost testing.

#### 4.1.2 Application Testing Results

Since Android has its limitations with IPv6-only network connections [29], we could not connect dynamically to the configured IPv6 network in the university lab, so we connected to the network by using static address, and thus we could not test the entire application as a whole, so we tested each one of its functions alone. The implementation works well and the application does its work correctly, except for configuring the new CGA Address to the device interface. The device does not take the new IPv6 address. This is because we connected with static IP address configuration. Figures 4.4 and 4.5 show the Android application logging page, the page shows some logging information after starting SEND service.



Figure 4.4: some logging information from Android Application

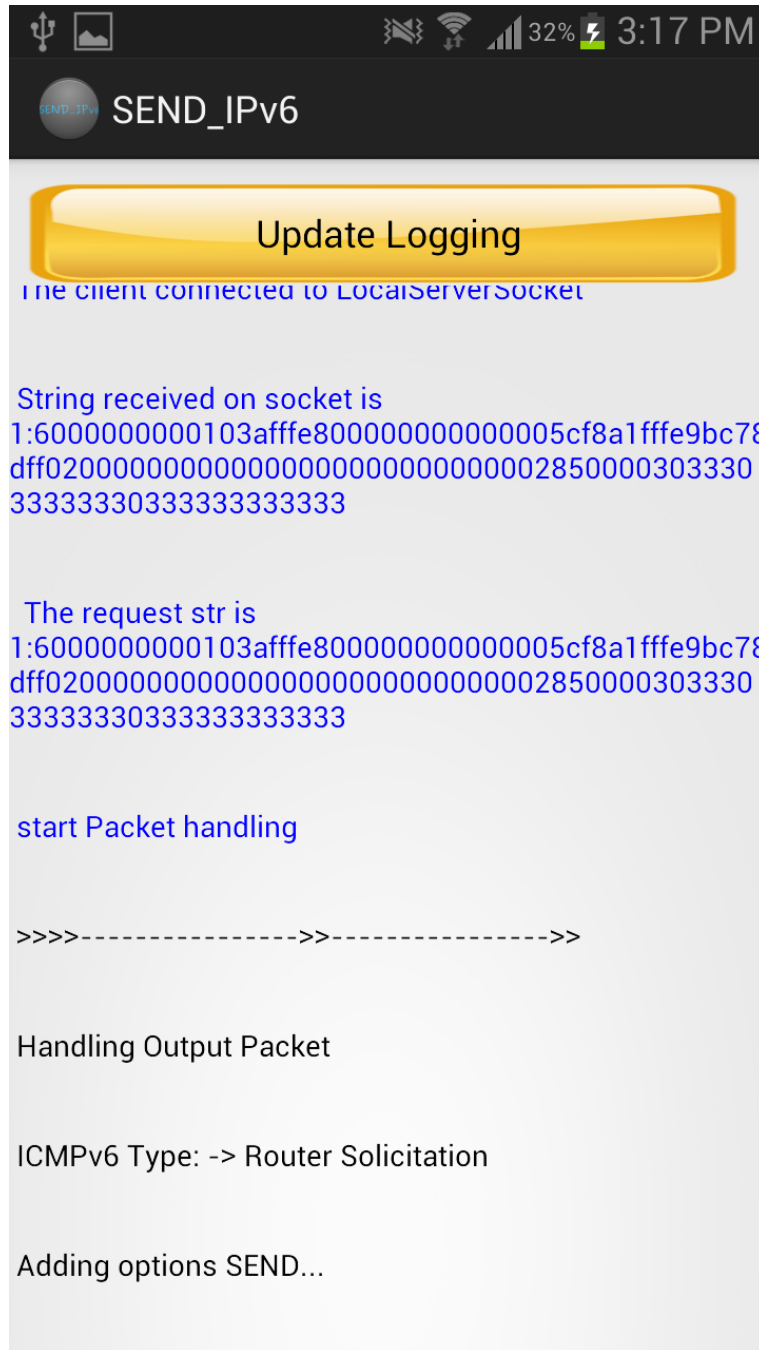


Figure 4.5: some logging information from Android Application



## 4.2 Conclusions

The Neighbor Discovery Protocol (NDP), which is an IPv6 Protocol that roughly corresponds to IPv4 ARP, is vulnerable to set of threats if not secured. The Secure Neighbor Discovery (SEND) extensions counter security threats to NDP by offering proof of address ownership, message protection, and router authorization. The fast growing of smart phones that connect to Internet for almost every service provided by companies around the world, and with huge companies binding their services to IPv6; comes the needs for implementing the Secure Neighbor Discovery extension on smart phones like Android.

SEND deployment is challenging task, especially on a limited resources devices like smart phones. There is a need for deep understanding of SEND new messages and options, how it works, and how to analysis packets and modify them. Also there must be a huge knowledge about Android devices, especially there architecture components and Kernel specifications.

Android devices has a serious limitations with IPv6 networks, the devices for the current time is hardcoded to connect to IPv4 networks, which limit the ability to dynamically connect to IPv6-only networks.



## References

- [1] W. George, et al. "IPv6-Required", RFC 6540, April. 2012;<http://tools.ietf.org/html/rfc6540>
- [2] S. Deering and R. Hinden. "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, December 1998;  
<http://tools.ietf.org/html/rfc2460>
- [3] IPv6 Implementation Guide, Cisco IOS Release 15.2S, "Implementing IPsec in IPv6 Security"
- [4] S. Thomson, T. Narten, and T. Jinmel, "IPv6 Stateless Address Autoconfiguration", RFC 4862, Sept. 2007  
;<http://tools.ietf.org/html/rfc4862>.
- [5] J. Arkko et al., "Secure Neighbor Discovery (SEND)", RFC 3971, Mar. 2005;<http://tools.ietf.org/html/rfc3971>
- [6] T. Aura, "Cryptographically Generated Addresses (CGA)", RFC 3972, Mar. 2005;  
<http://tools.ietf.org/html/rfc3972>.
- [7] A. Alsa'deh and C. Meinel. "Secure Neighbor Discovery", IEEE SECURITY & PRIVACY Magazine, July/August 2012;  
[http://www.hpi-unipotsdam.de/fileadmin/hpi/FG\\_ITS/papers/Trust\\_and\\_Security\\_Engineering/2012\\_Alsadeh\\_SecurityPrivacy.pdf](http://www.hpi-unipotsdam.de/fileadmin/hpi/FG_ITS/papers/Trust_and_Security_Engineering/2012_Alsadeh_SecurityPrivacy.pdf)
- [8] T. Cheneau, "NDprotector";<https://amnesiak.org/ndprotector/>
- [9] A. Jayaraj, "What Is Scapy", Oct. 2013;  
<http://resources.infosecinstitute.com/what-is-scapy/>
- [10] R. Hinden. "IPv6 Addressing Architecture", RFC 4291, Feb. 2006;  
<http://tools.ietf.org/pdf/rfc4291.pdf>
- [12] T. Narten, "Neighbor Discovery in IPv6", RFC 4861, Sep. 2007;  
<http://tools.ietf.org/pdf/rfc4861.pdf>
- [13] S. Garbett, "IPv6 Neighbor Discovery Protocol", Nov. 2012;  
<http://ccieblog.co.uk/ipv6/ipv6-neighbor-discovery-protocol-ndp>
- [14] Arkko, J., "Effects of ICMPv6 on IKE", March 2003.
- [15] P. Nikander. "IPv6 ND Trust Models and Threats", RFC 3756, May. 2004;  
<http://tools.ietf.org/pdf/rfc3756.pdf>
- [16] A. AlSa'deh, H. Rafiee, and C. Meinel, "Stopping Time Condition for Practical IPv6 Cryptographically Generated Addresses," Proc. 26th IEEE Int'l Conf. Information Networking (ICOIN 12), IEEE, 2012, pp. 257–262.
- [17] <http://developer.android.com/about/index.html>

- [18] <https://github.com/TrustRouter/TrustRouter>
- [19] <https://github.com/TrustRouter/TrustRouter/wiki/Client-Implementation:-Overview>
- [20] <https://github.com/TrustRouter/TrustRouter/wiki/Router-Advertisements-and-SEND>
- [21] <http://sourceforge.net/projects/easy-send/>
- [22] <http://www.netfilter.org>
- [23] [http://man.cx/?page=ip6tables\(8\)](http://man.cx/?page=ip6tables(8))
- [24] [http://www.netfilter.org/projects/libnetfilter\\_queue/](http://www.netfilter.org/projects/libnetfilter_queue/)
- [25] <http://developer.android.com/tools/sdk/ndk/index.html>
- [26] A. Siddique, M. Akhtar, S. Tanweer, "Android Basic Architecture including Operating System using their Application", April 2012; [http://www.ijmra.us/project%20doc/IJMIE\\_APRIL2012/IJMRA-MIE900.pdf](http://www.ijmra.us/project%20doc/IJMIE_APRIL2012/IJMRA-MIE900.pdf)
- [27] N. Smyth, "Android 4.4 App Development Essentials", 4th Edition book, chapter 6, 2004.
- [28] <http://www.savarese.org/software/vserv-ipq>
- [29] <https://code.google.com/p/android/issues/detail?id=32630>