

Technical Document

**CSCI321 – Project (Android Packet Sniffer)**

Table of Contents

1. [1. Document Overview 3](#_Toc518676510)
2. [2. Introduction 3](#_Toc518676511)

[1.1 What is Packet Sniffing?](#_Toc518676512)

[1.2 What is TCPDump?](#_Toc518676513)

[1.2.1 How to setup TCPDump in Linux Environment](#_Toc518676514)

[1.3 What is libpcap?](#_Toc518676515)

[1.3.1 Creating our own Binary like TCPDump](#_Toc518676516)

[1.3.2 Difficulties in creating the Binary](#_Toc518676517)

[1.4 Type of IP Packets](#_Toc518676518)

[1.4.1 IP Packet Header](#_Toc518676519)

[1.4.2 ICMP Packet Header](#_Toc518676520)

[1.4.3 ARP (Address Resolution Packet)](#_Toc518676521)

[1.5 Types of Protocols](#_Toc518676522)

[1.5.1 Transport Control Protocol (TCP)](#_Toc518676523)

[1.5.2 User Datagram Protocol (UDP)](#_Toc518676524)

1. [3. Implementation libpcap that binary PCBIN uses 11](#_Toc518676525)

[3.1 Main Functions of libpcap Library](#_Toc518676526)

[3.2 Future Updates](#_Toc518676527)

1. [3. Application Description 13](#_Toc518676528)

[3.1 Introduction](#_Toc518676529)

[3.2 Operational Examples](#_Toc518676530)

[3.3 System Requirements](#_Toc518676531)

[3.4 Functional Requirements](#_Toc518676532)

[3.4.1 Start/Stop Packet Sniffing](#_Toc518676533)

[3.4.2 Display Information](#_Toc518676534)

[3.4.3 Save Packets to File](#_Toc518676535)

[3.4.4 Filter the Packets](#_Toc518676536)

[3.5 Non-functional Requirements (Quality of life Service)](#_Toc518676537)

1. [4. UML Diagram 15](#_Toc518676538)

[4.1 Use Case Diagrams](#_Toc518676539)

[4.1.2 Second Iteration](#_Toc518676540)

[4.2 Sequence Diagrams](#_Toc518676541)

[4.3 Activity Diagrams](#_Toc518676542)

[4.4 Other UML Diagrams](#_Toc518676543)

1. [5. User Interface Designs 21](#_Toc518676544)
2. [6. Software Development 22](#_Toc518676545)

[6.1 Software Design Process](#_Toc518676546)

[6.2 Development Tools and Environment](#_Toc518676547)

[6.3 Schedule of Project](#_Toc518676548)

[6.4 Roles & Responsibilities](#_Toc518676549)

[6.5 Version Control](#_Toc518676550)

1. [7. Application Test 23](#_Toc518676551)

[7.1 Test Cases](#_Toc518676552)

1. [8. Conclusion 23](#_Toc518676553)

Group Member List

|  |  |
| --- | --- |
| Name | Student ID |
| Soh Yu Xuan | 5498636 |
| Timothy Chin | 5498399 |
| Kenneth Huang | 5498442 |
| Kendrick Tan | 5026556 |

# 1. Document Overview

The document describes the design process which is used in creating the application, it also includes UML Diagrams to illustrate the design of the system that can be used to show the functionalities of the application. Some form of implementation may also be described. Other than functionalities there will also be test cases to show that the application have the appropriate quality for use.

# 2. Introduction

Computers used to be large clunky machines which were the innovation of the IT sector but now, almost everyone if not all, has a “small computer” with them all the time, a smartphone. One of the duties of a network administrator is to monitor the network. This can be done with a computer with either with an external network card or with an internal one. However, the computer was something that was needed, adding weight and hassle. What if one could monitor and sniff packets from their device whom they carried with them everywhere? One could simply be in a location, capturing packets in their vicinity for analysis, without having to lug around a laptop right into their mobile device.

## 1.1 What is Packet Sniffing?

“A sniffer (packet sniffer) is a tool that intercepts data flowing in a network. If computers are connected to a local area network that is not filtered or switched, the traffic can be broadcast to all computers contained in the same segment. This doesn’t generally occur, since computers are generally told to ignore all the comings and goings of traffic from other computers. However, in the case of the sniffer, all traffic is captured when the sniffer software commands the Network Interface Card (NIC) to stop ignoring the traffic. The NIC is put into promiscuous mode, and it reads communications between computers within a particular segment. This allows the sniffer to seize everything that is flowing in the network, which can lead to the unauthorized access of sensitive data. A packet sniffer can take the form of either a hardware or software solution.

## 1.2 What is TCPDump?

TCPDump is a free software designed by Van Jacobson, Sally Floyd,Vern Paxson and Steven McCanne in 1988. It uses the libpcap library in C to read packets from a network interface card. TCPDump prints the contents of network packets, it can also read the packets from a network interface card (NIC) or from a saved packet file, also it can write packets to standard output or to a file. For a more specific purpose we can use it to intercept and display communications of another user, with the necessary privileges of a system acting as a router or gateway through which those unencrypted traffic like telnet or HTPP passes we can use TCPDump to view login IDs, passwords, URLs and also the content of which the websites that were being viewed or even any other unencrypted information. When using TCPDump on a network with high volumes of traffic filters can be used like BPF-based to limit the number of packets that will be seen by TCPDump, which makes the output more usable. However, in order for TCPDump to be used, superuser privileges is required as packet capturing mechanisms on those systems requires the privileges. Which is an essential part of our project whereby we need a rooted phone as well as superuser privilegs.

## 1.2.1 How to setup TCPDump in Linux Environment

wget http://www.tcpdump.org/release/libpcap-1.8.1.tar.gz

tar -xf libpcap-1.8.1.tar.gz

sudo apt-get update

sudo apt-get install flex

sudo apt-get install bison

cd libpcap-1.8.1

./configure –prefix=/usr

Make

Sudo make install

Following the above step will compiled the libpcap giving libpcap.a and libpcap.so file.

The .a file are static libraries while .so are dynamic libraries. The difference between the two is, in a static libraries, if you used code stored inside them, it is taken from them and embedded into your own binary while in dynamic libraries, the code is not taken and embedded into your own library. It is referenced so the binary will depend on them and the code from the so file is loaded at runtime.

## 1.3 What is libpcap?

libpcap is the abbreviation of packet capture library which is an application programming interface (API ) that is used to capture network traffic. The libpcap is written in c so therefore in order for android which uses JAVA, we must use a wrapper to translate a library existing interface to the compatible interface. The wrapper enable android which uses JAVA to use libpcap C code.

Example of some wrapper program readily available would be jPcap,jNetPcap,JPcap and pcap4j.

These are all wrapper libraries made by other developers to enable C code to be use in JAVA.

## 1.3.1 Creating our own Binary like TCPDump

We planned to achieve this by using libpcap (API) and write the general code which is similar to TCPDump. After that, we cross compile it into an ARM compatible structure and use the ARM binary in our applications. Why do we use ARM? As most mobile uses ARM processors instead of intel processors. The difference between linux ARM and linux x86 is that internally their binary code is different thus software done on x86 must be pre compile to match the target architecture.

## 1.3.2 Difficulties in creating the Binary

Although these libraries are readily available for us to use in android we face one problem.

One of the problems we faced was that if we want to sniff packets in promiscuous mode, we would need root permission. Getting root permission would not be a problem if we were to work in environments such as LINUX or WINDOWS on the computer.

In android any access to protected resources or services is guarded by the application permission framework, all access in native code needs to be analyzed, and the required permissions should be identified. Whatever permissions the native code may need should be published for developers, so that they can include these permissions in their applications Manifest file. Native code should not rely on code that need access as this is not available on standard android. [4]

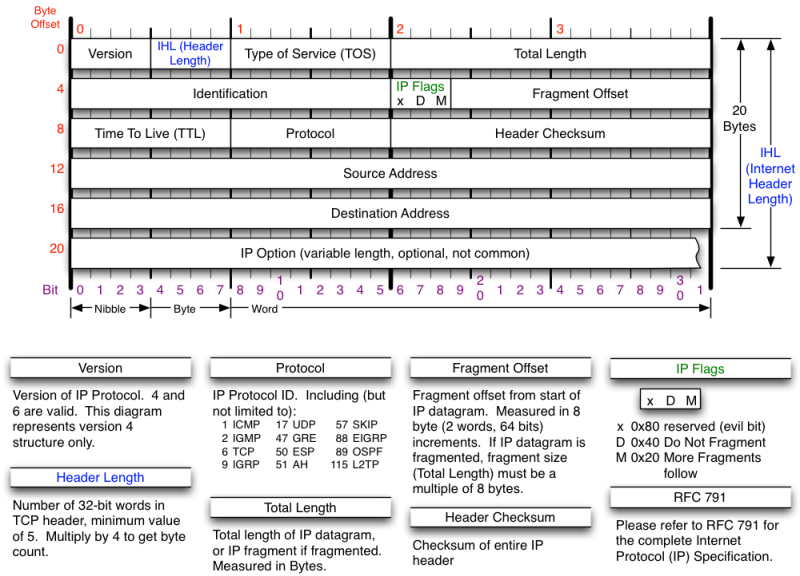
The above sentence is a research that we did when we came across that using a wrapper library that need route access violate that rule .Therefore we went to look for another alternative which is to use a ARM binary library for ARM architecture . If we were to use a binary in android we can give root to process and therefore bypassing the no rooting to native code problem.

The next problem would be that even though we created our own ARM binary, the binary should have no problem executing the code in the NEXUS 5 device. However, when we tried to execute the binary, it was denied permission even though the phone is rooted and ran in a rooted process. No available solution has been found for this problem for now.

## 1.4 Type of IP Packets

1. TCP/IP
2. UDP
3. ICMP (INTERNET CONTROL MESSAGE PROTOCOL)
4. IGMP (INTERNET GROUP MANAGEMENT PROTOCOL)
5. IGRP ( INTERIOR GATEWAY ROUTING PROTOCOL)
6. ESP (ENCAPSULATING SECURITY PAYLOAD)
7. AH (AUTHENTICATION HEADER)

## 1.4.1 IP Packet Header

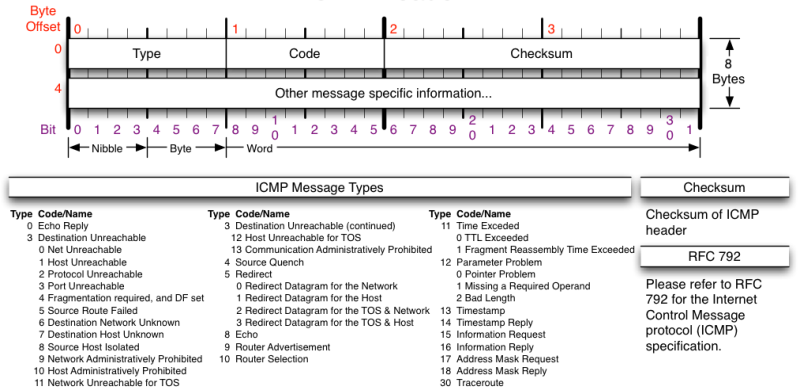
**[5]**

IP header usually about 20 bytes. The source address and destination address mean from where the ip packet comes from and where the packet is trying to reach.

The identification field is used to uniquely identify the group of fragments of a single ip datagram.

The Types of service has now been redefined by RFC 2474 for Differentiated service and Explicit Congestion Notification (ECN). Differentiated service (DSCP) is used when technologies require real time data streaming is needed. ECN is defined in RFC3186 and allows end to end notification of network congestion without dropping packets. It is an optional feature which is only effective if there is underlying network.

## 1.4.2 ICMP Packet Header

**[5]**

Currently our program is unable to sniff ICMP packets due insufficient time. However, our finished application should be able to ICMP packet would be sniff with our binary, PCBIN.

ICMP is not a transport protocol that sends data between systems. It is used to troubleshoot internet connections by network admin.

**IGMP (INTERNET GROUP MANAGEMENT PROTOCOL)**

Used by host and adjacent router on IPV4 network to establish multicast group membership.

**IGRP (INTERIOR GATEWAY ROUTING PROTOCOL)**

Distance vector interior gateway protocol. Used by routers to exchange routing data within an autonomous system.

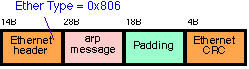
**ESP (ENCAPSULATING SECURITY PAYLOAD)**

Provides data confidentiality and authentication. Authentication mechanism authenticates only IP datagram portions of the IP packet.

**AH (AUTHENTICATION HEADER)**

Provides a mechanism for authentication only. It authenticates IP header and their payload.

## 1.4.3 ARP (Address Resolution Packet)

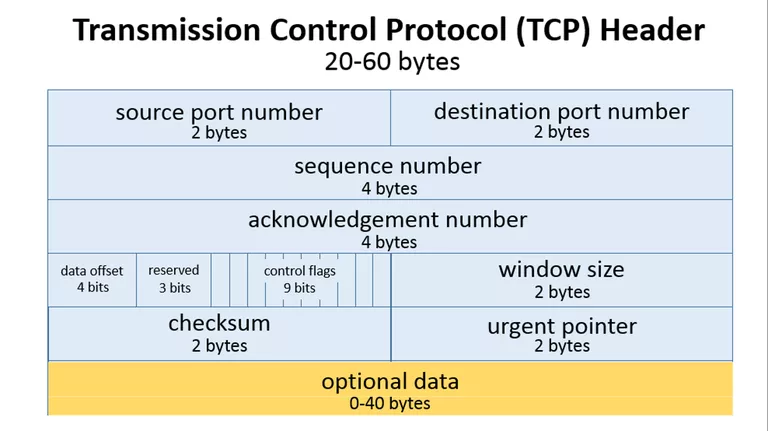
**[6]**

ARP is a protocol used by Internet protocol (IP) which broadcast packet to map IP network address to MAC Address. It is used by data link protocol. The ARP request message is sent by using the ethernet broadcast address to all system. Only the person who has the request IP address will reply. The rest will silently discard the packet.

## 1.5 Types of Protocols

In the Transport Layer, there are 2 protocols that are used, Transport Control Protocol and User Datagram Protocol.

## 1.5.1 Transport Control Protocol (TCP)

[2]

TCP header usually requires a minimum of 20 bytes anything.

As shown above the source port takes up around 2 bytes, destination port takes up around 2 bytes,

Sequence number takes up 4 bytes. Sequence number is used to mark the ordering of a group of message. Acknowledgment number takes up 4 bytes are used by the sender and receiver to communicate the sequence number of messages.

Data offset field takes up 4 bits and store the total size of a TCP header in multiples of four bytes. A header using the optional TCP field has a data offset of 5, while a header using the maximum-sized optional field has a data offset of 15.

Reserved uses 3 bits and the value is always zero.

Control flags uses 9 bits and TCP use 6 standard and 3 extend control flags to manage data flow in specific situations. Each flag is 1 bit in size. The flags are SYN, ACK, FIN, URG, PSH, RST, ECE, CWR, NS flags.

SYN aka synchronous flag is used as a first step in establishing a 3 way handshake between 2 hosts.

ACK aka acknowledgment is used to acknowledge the successful receive of a packet. It is use to tell the sender it has receive the initial packet.

FIN aka as Finished. Tell the receiver that sender has no more data to send. Therefore this is the last packet sent to receiver.

URG aka urgent. This flag is to the receiver to process the urgent packet first before processing other packet.

PSH stands for push. It is similar to URG flag. It is used to tell the receiver to process the packets with PSH flag instead of buffering them.

RST stands for reset flag. This flag is sent when a packet is sent to a particular host that was no expecting it.

ECE is responsible for indicating if a TCP peer is ECN capable

CWR flag stands for Congestion window reduced is used by sending host to indicate it received a packet with ECE flag set.

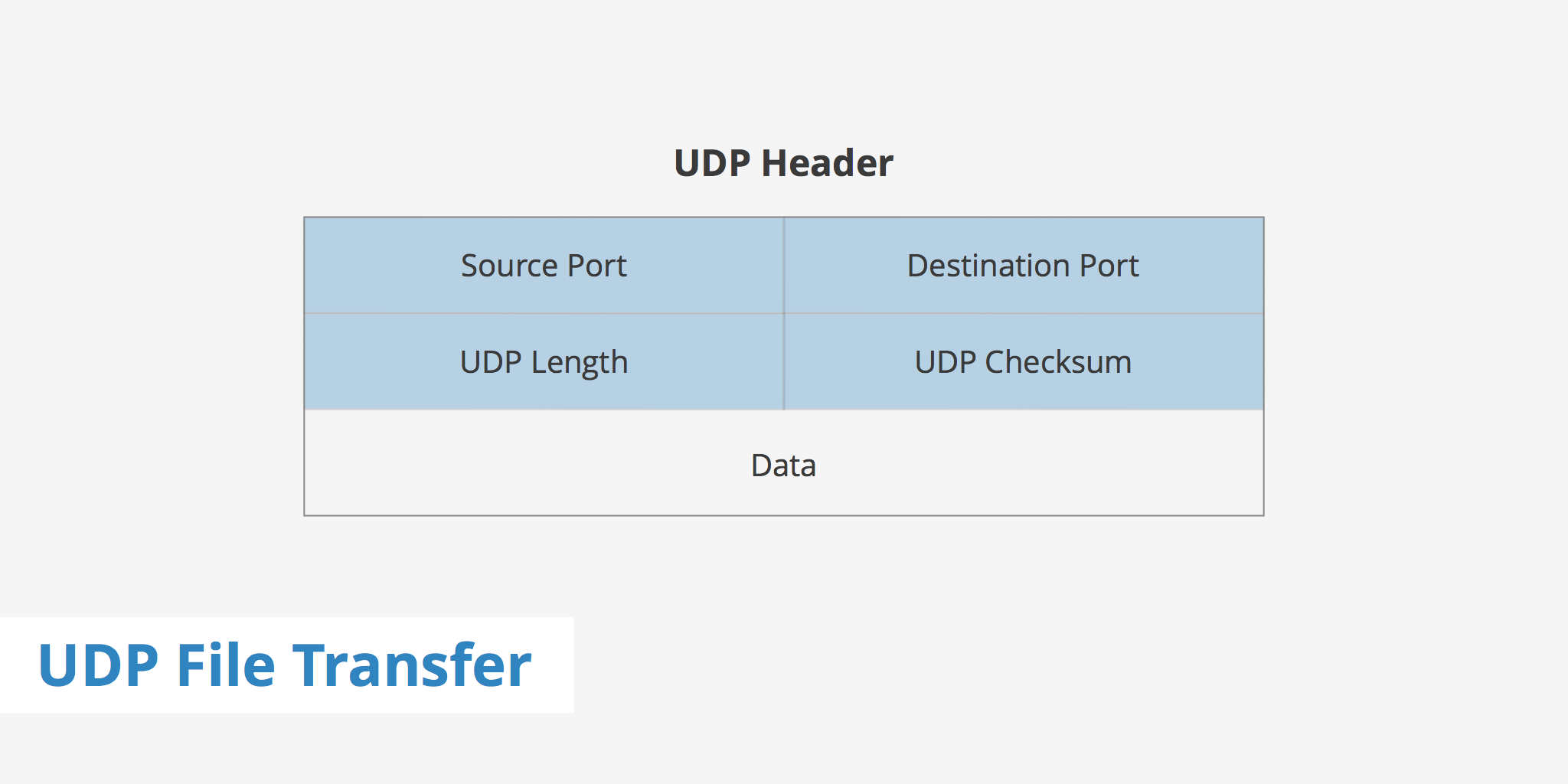
NS stands for Nonce Sum, it is an experimental flag used to help protect accidental malicious concealment of packets from sender.

Window size is a value to regulate how much data is send before an acknowledgment from receiver is required. This value cannot be too small or too big as being too small will slow performance while too big will cause network link to be saturated making it unusable for other application

Checksum is used to help receiver check if the data receive has been corrupted or tempered with.

Urgent pointer field is often set to zero or ignored. [3]

## 1.5.2 User Datagram Protocol (UDP)



[1]

UDP header is much smaller than TCP. It consist of 8 bytes in total.

As shown above it only has four fields. Source port, Destination Port, UDP length and UDP checksum.

UDP length is used to identify the length of the header itself.

UDP checksum is the same as TCP checksum. It is used to check if a UDP packet data is corrupted or been tampered with.

# 3. Implementation libpcap that binary PCBIN uses

## 3.1 Main Functions of libpcap Library

1. pcap\_lookupdev

* Finds the default device on which to capture the packet.
* Pcap\_lookupdev returns a pointer to a string giving the name of a network device suitable for use with other main function to create a packet sniffer.
* However in our program we would like to use our own interface that we specified therefore this function is not needed.

1. Pcap\_create(device, errbuff)

* Used to create a packet capture handle to look at the packet on the network

1. Pcap\_set\_promisc(handler,int )

* To set promiscuous mode on a non-activated handle. If the int is any non-zero value, promiscuous mode will be set otherwise it will not be set.

1. pcap\_activate(handler)

* Using pcap\_create just creates the handler in order to use the handler , we would need to activate it

1. pcap\_datalink

* Get the link layer header type

1. pcap\_lookupnet ()

* Get the netmask and network number of IPv4 for a device

1. pcap\_compile()

* Used to compile the string str in to filter program.Program is a pointer to a bpf\_program struct .optimize controls whether optimization on the resulting code is performed.
* Return 0 on success and -1 on failure.

1. pcap\_setfilter()

* Used to specify a filter program which is apointer to a bpf\_program struct ,usually the result of a call to pcap\_compile()
* Return 0 on success -1 on failure.

1. pcap\_loop()

* Loop mean that if will keep sniffing packet until it is specified to stop.
* It loops a specified function that is placed on the third argument

1. pcap\_close()

* Closes the handler once it has completed its job.

## 3.2 Future Updates

* Enable sniffing of all other packet for eg (ARP packets, DNS and ICMP packets)
* Implement it together with libnet library to send packet to other device.

(Libnet is required to break wep)

# 3. Application Description

This section will give an overview of the application design, development, and Implementation

## 3.1 Introduction

Our Application is a development of a network sniffer on android platform. Although it might have similarities to that of a Computer version, the user needs to know that although the application might have the same functionalities as that of the Computer Version, there are a lot more limitations that will be faced on the Android platform version.

## 3.2 Operational Examples

The application would be used mostly by network administrators, students who take Computer Science (Security) courses as the knowledge required to use the application might be different from normal users.

How a network administrator can use this application for example is monitor day to day network traffic to check for anomaly and irregularities so as to check for security loopholes or vulnerabilities that might occur as each system will need to be updated constantly as technology is constantly improving the number of loopholes or vulnerabilities will increase.

For students, it mostly lies in their studies where they may or may not need this when pursing their studies as it is an application for those keen in android application development as well as network related studies

## 3.3 System Requirements

* Android 4.0 (Ice Cream Sandwich) – Android 6.0 (Marshmallow)
* Chipset Qualcomm MSM8974 Snapdragon
* Wi-Fi 802.11 a/b/g/n/ac
* Compatibility with Nexmon

## 3.4 Functional Requirements

Under this section, we will be describing the functional requirements of our application.

## 3.4.1 Start/Stop Packet Sniffing

When users open the application, they will be able to select when they wish to start and stop the capturing of packets. The application works by utilizing a binary that will be started in a process with SuperUser permission. The process runs in a thread and stops only when the stop button is pressed. The packets are captured using the binary and are redirected to a text file.

## 3.4.2 Display Information

The application works by reading in the text file that is at the same time being written to by the sniffing thread. As data enters the text file, another thread is used to read in the packet data to the application. Another thread at the same time will get the data read in and display to the gui portion of the application.

Users will be able to view the packets captured by utilizing 3 threads, writing to a text file, reading a text file and updating to the display all being done concurrently.

## 3.4.3 Save Packets to File

After the user is done capturing packets, they will be able to save the data to a file which they can then use another application to view the data at their own time or for system administrators they will be able to keep records of the type of data that has been going through the network.

This packet information is saved in a ‘.pcap’ format contains more information that can be displayed on a mobile device. It would be better analyzed on a computer.

## 3.4.4 Filter the Packets

Our application would be designed to filter the types of packet that the user wishes to see, according to certain devices/websites/IP addresses/packet type.

## 3.5 Non-functional Requirements (Quality of life Service)

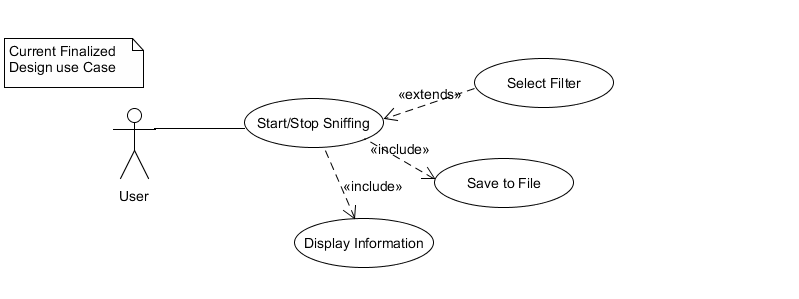
This section describes the non-functional requirements of our application that is designed to help users navigate through the application with as little difficulty as possible

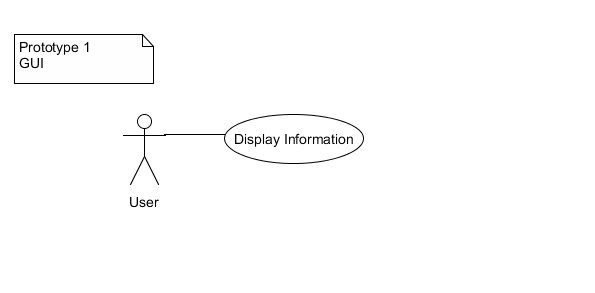
# 4. UML Diagram

## 4.1 Use Case Diagrams

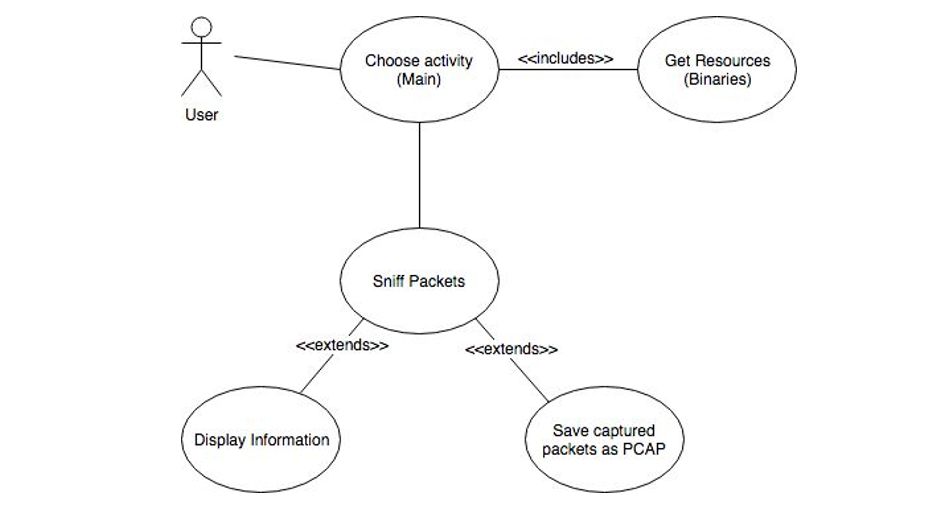
First Iteration:

First Design of Application:



First GUI Design of Application:

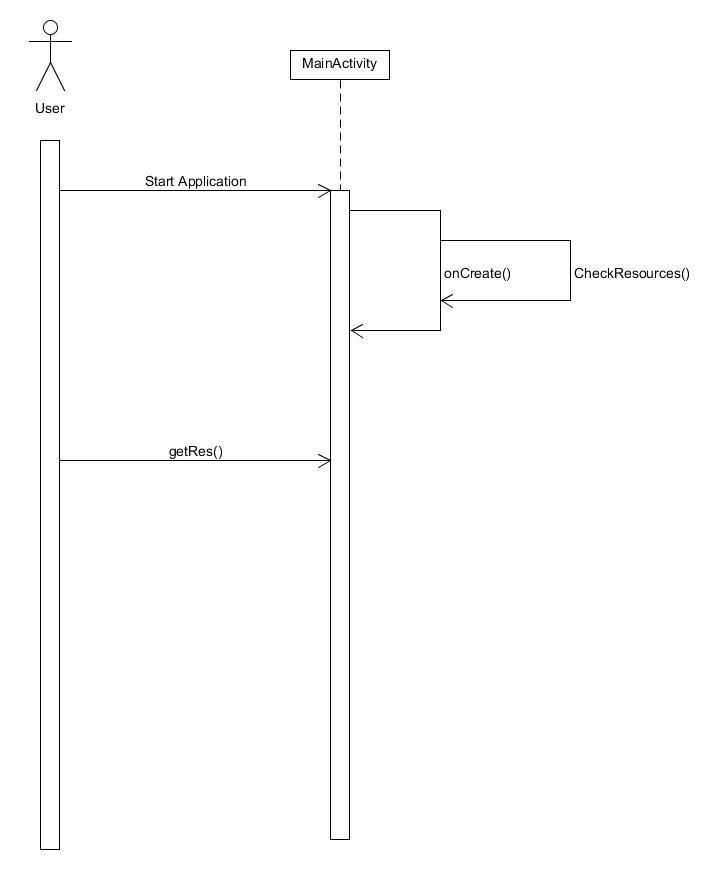
## 4.1.2 Second Iteration



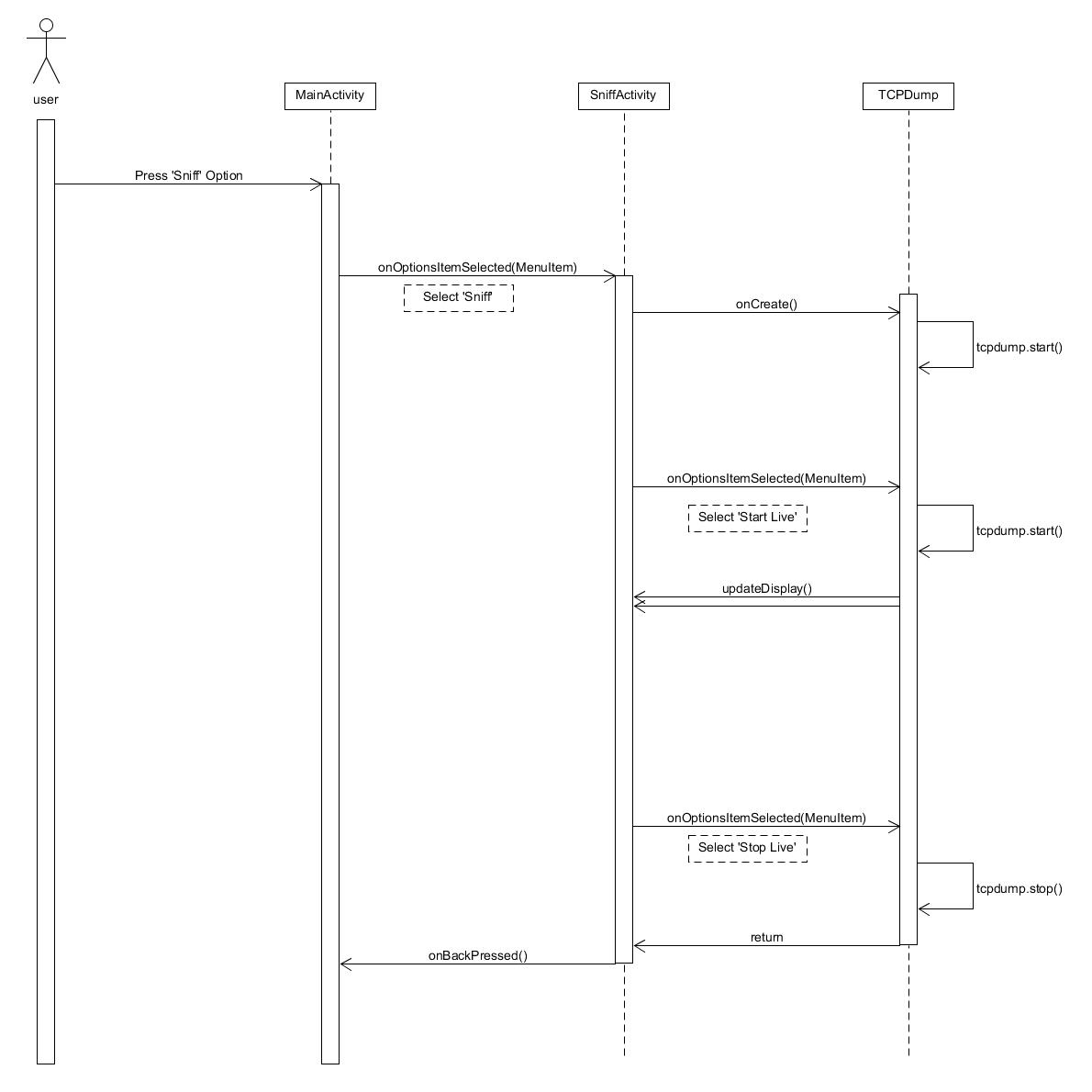
## 4.2 Sequence Diagrams

First Iteration:

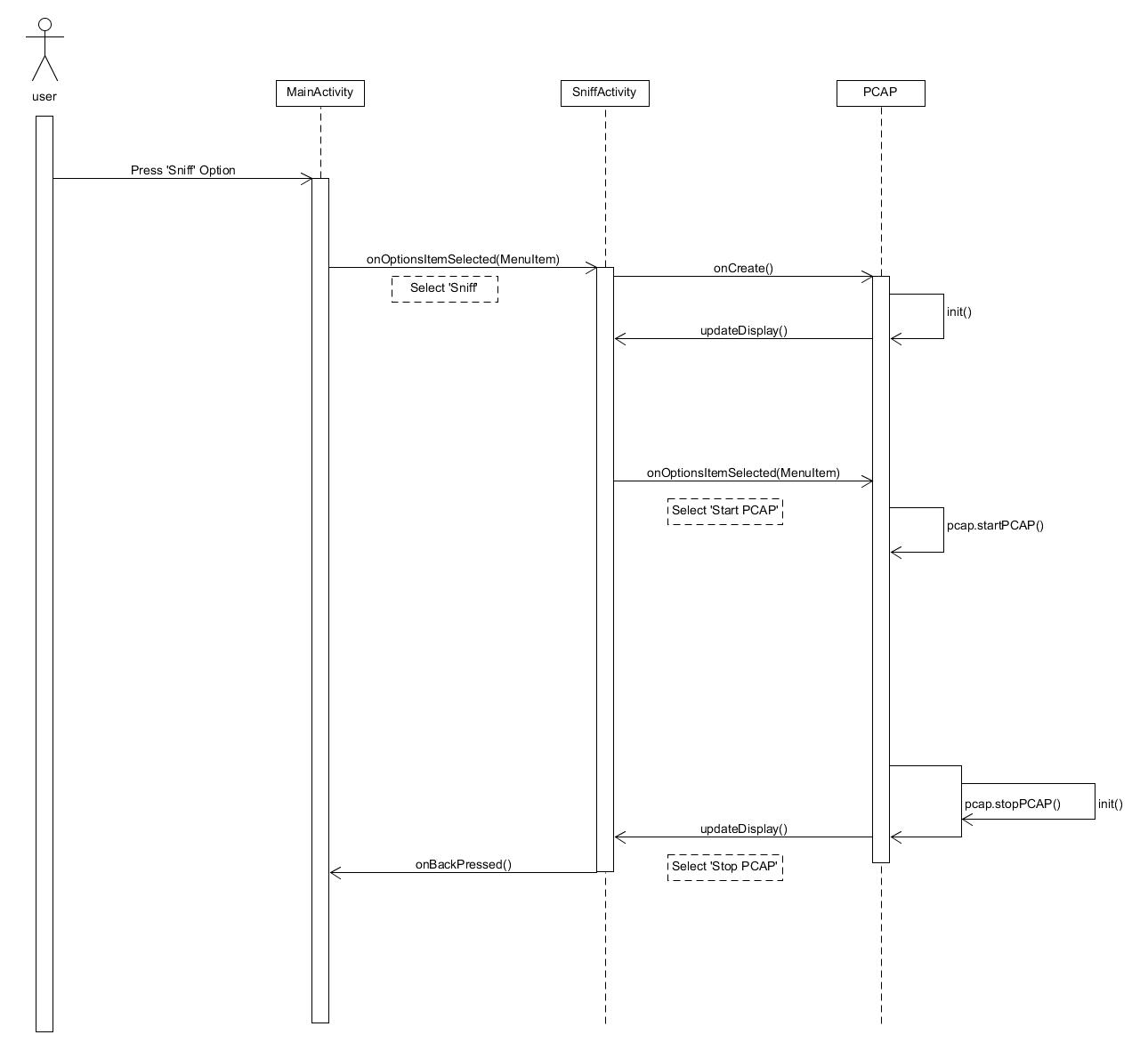
App initialize sequence diagram:



Start live sequence diagram:



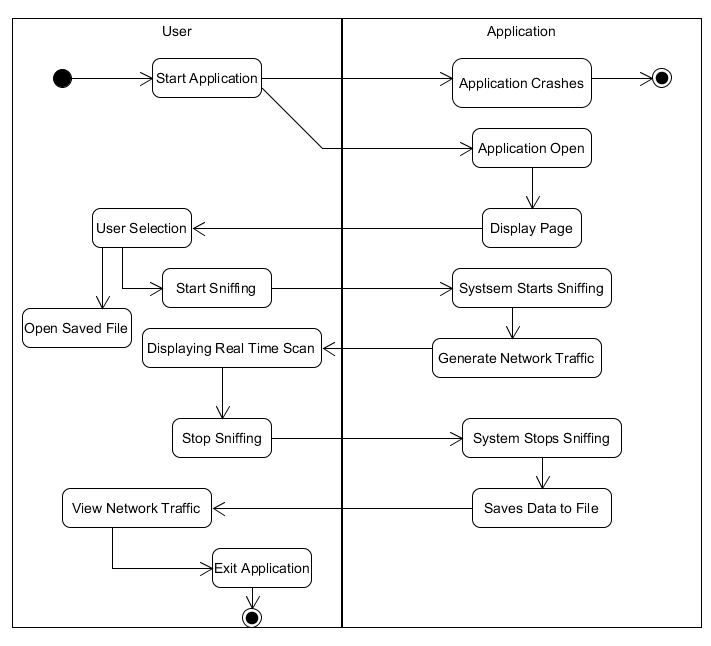
Start pcap sequence diagram:



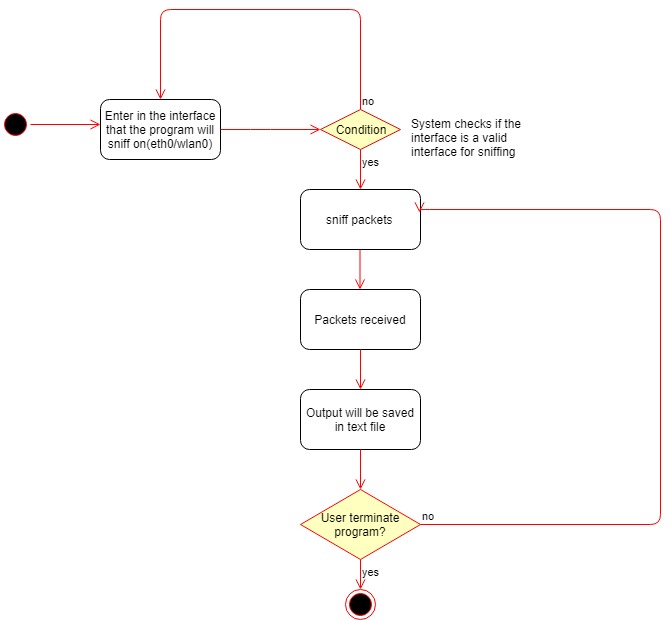
## 4.3 Activity Diagrams

First Iteration:

Android app activity diagram:



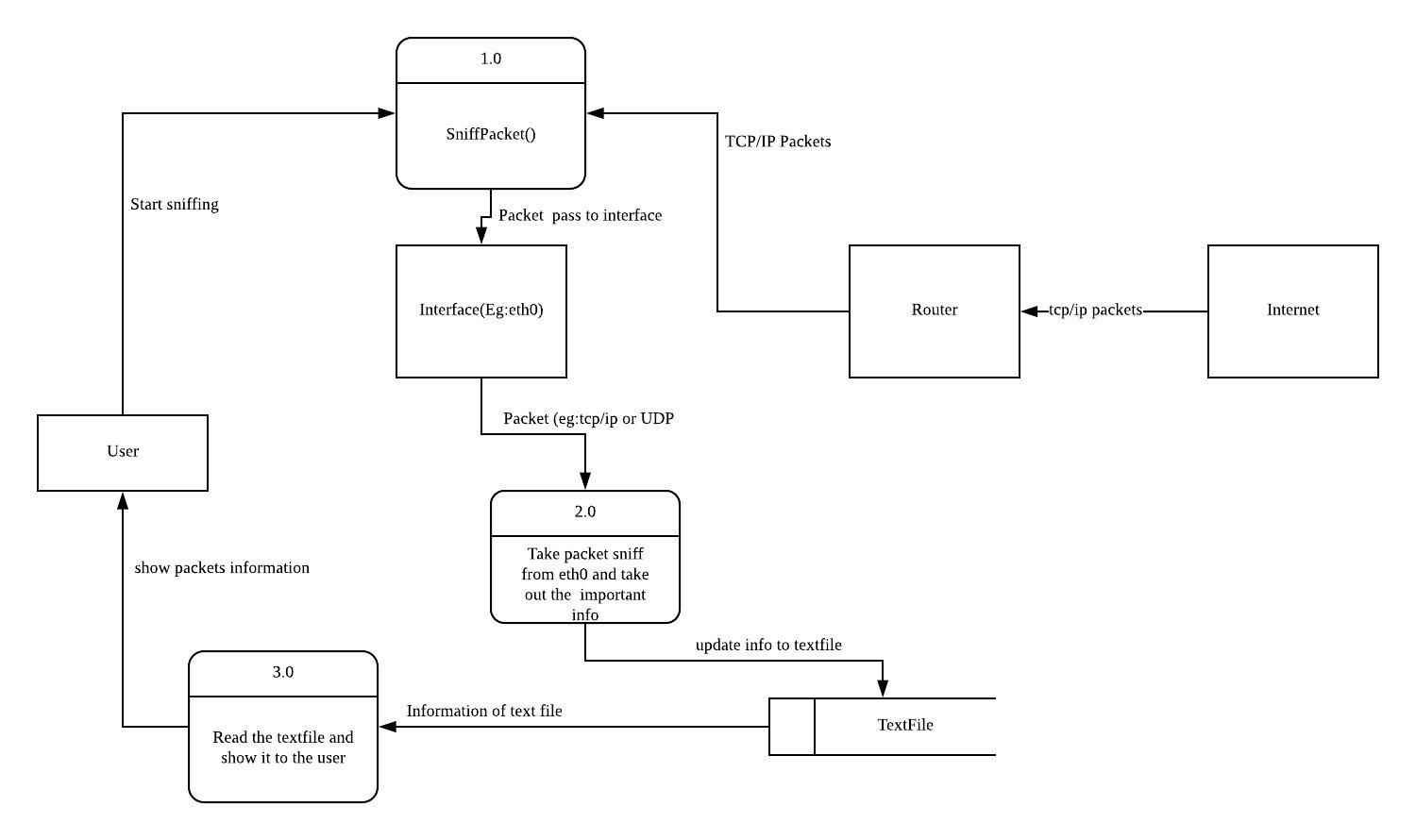
Pcbin activity diagram:



## 4.4 Other UML Diagrams

First Iteration:

Data Flow Diagram for pcbin



# 5. Application Program

**5.1 Application Program Technical Aspect Introduction**

**Before looking at the codes there are some technical aspect that needs to be introduced so that the user will be able to understand how our application works, what are the limitations, and possible problems.**

**Some of the information below was extracted online from websites like Wikipedia, stack overflow**

**5.1.1 List of Application Program Technical Aspect**

**The list below shows what is required for our application for it to be able to run**

**Android Platform**

**This section talks about the device used by our application as well as introduction to the device and possible constrains faced.**

* Operating System:

The application is run based on the Operating System (OS) of a mobile device known as Android, it is currently being used in most mobile devices such as Samsung, the only other known OS would be iPhone OS (iOS).

* Android Version:

Users need to know that their Mobile devices have different versions as the device must undergo updates consistently due to maintenance and fixes. Currently the Android Version most up to date is version 9.0 Pie. However, our application runs on Version 4.0 Ice Cream Sandwich to Version 6.0 Marshmallow, so this means newer devices might not be able to support our application which brings us to the first hurdle, acquiring a device for use.

* Chipset:

Mobile Phones have a design like that of a Computer Central Processing Unit (CPU) known as chipset. However, the design of a Chipset is like that of a Motherboard of a Computer, it accepts all the components to sit upon it and connect with each other such as Camera, Bluetooth, Wi-Fi, touch screen with CPU/Flash Storage/Ram. Due to the earlier mentioned Android Version, it affects the chipset as well, as the chipset used by our application is called Qualcomm MSM8974 Snapdragon. It is a type of chipset used in a limited number of devices which is another hurdle as the list of devices with said chipset is limited and thus limits the device that can be used.

**Network**

**This section talks about network and how it is related to our application as well as the possible constrains**

* IEEE 802.11

a set of Media Access Control (MAC) and physical layer (PHY) specifications implementing wireless local area network (WLAN) it is the world’s most widely used wireless network standards it is used by Mobile Devices to talk to each other and access the internet.

* WNIC

Wireless Network Interface Controller (WNIC) is a network interface controller which connects to a wireless radio-based computer network rather than using wired. It works on the Layer 1 and 2 of the Open Systems Interconnection Model (OSI Model). It is usually applied to IEEE 802.11 adapters.

Modes of Operation are Infrastructure and Ad Hoc

For Infrastructure the WNIC need a wireless Access Point, all data is transferred using the access point as a central hub, all nodes in an infrastructure connect to an access point. All nodes connecting to the access point must have the same Service Set Identifier (SSID) as the access point. And if any kind of wireless security is enabled on the access point (WEP/WPA) they must share the same keys or other authentication parameters

For Ad Hoc Mode, the WNIC does not require an access point but rather can interface with all other wireless nodes directly. All the nodes in an ad hoc network must have the same channel and SSID

In a 802.11 WNIC, the MAC Sublayer Management Entity (MLME) can be implemented either in the NIC’s hardware or firmware or host-based software that is executed on the main CPU, and a WNIC that implements the MLME function in hardware or firmware is called a FullMAC WNIC or HardMAC NIC, and a NIC that implements it in host software is known as SoftMAC NIC. FullMAC chips are typically used in mobile devices.

* 802.11 Frames

Data Frames which are considered the pack horses, where data is being hauled from station to station, data frames carry higher-level protocol data in the frame body, before the frame body itself it has frame control, duration ID, address 1 (receiver), address 2 (sender), address 3 (filtering), Seq-ctl, and an optional address 4 the type of data frames are Data which is moving the frame body from one station to another and Null which consist of a MAC header followed by the FCS trailer.

Control Frames used in conjunction with data frames to do area clearing operations, channel acquisition, carrier-sensing maintenance functions, positive acknowledgement of received data. Control frames assist in the delivery of data frames and all control frames use the same frame control field.

* Management frames which does functions regarding supervisory, such as joining and leaving a wireless network, move associations from access point to access point. An identity of a network station can be broken down into three components, the first is mobile stations in search of connectivity must first locate a compatible wireless network to use for access, second, the network must authenticate mobile stations, third mobile stations must associate with an AP. For all management frames structure the MAC header is the same in all the management frames. And for the frame body most of the data contained in the frame body is fixed fields (fixed-length) or information elements (variable-length). 802.11 management frames have MAC headers with 3 addresses fields in it, for 802.11a/b/g it has 24-byte MAC Header, 802.11n has 28-byte (extra byte is HT control field)

So, what are fields? Fields also have authentication algorithm number with values 0 (open system authentication), 1 (shared key authentication), 2-65535 (reserved). Apart from authentication algorithm fields contain an authentication transaction sequence number used to track progress through the authentication exchange as well as beacon interval used to set the number of time units between beacon transmission. The beacon transmission contains capability information to advertise the network’s capabilities with each bit being used a s a flag to advertise a particular network function. Other things in field are current AP Address (MAC address of the access point which they are associated), Listen interval (number of beacon intervals that stations wait between listening for beacon frames), association ID (stations associated with an access point are assigned association ID to assist with control and management functions), timestamp (synchronize between stations in a BSS), reason code (disassociation or deauthentication frames, when the sender has not properly joined the network part of the frame is a 16-bit reason code field), status code (success/failure of operation).

For fields in a management frames it may have up to 10, there is a duration field which have 4 rules, contention-free period of duration: 32768, frames transmitted to a broadcast/multicast destination duration: 0, if more fragments bit is 0 duration: SIFS + ACK, if more fragments bit is 1 duration = fragment+3x SIFS+2x ACK.

Management frames generic information element contains a ID Number, length and variable-length component. Within the information elements there is also Service Set Identity (SSID) which allows network managers to assign an identifier to the service set, whenever stations attempting to join a network may scan an area for available networks and join the network with specific SSID. SSID is the same for all the basic service areas composing an extended service area. Apart from SSID there are also others such as supported rates which consist of a string of bytes that uses seven low-order bits for the data rate, most significant bit indicates whether the data rate is mandatory and up to eight rates may be encoded in the information element. Management frames also has their types like Beacon (important part of many network maintenance task), Probe Request (mobile stations use it to scan an area for existing 802.11 network), Probe response (it is sent if the request encounters a network that has compatible parameters), IBSS announcement traffic indication map (ATIM) is when the station has a buffered frames for a receiver in low-power mode, This is sent during the delivery period to notify recipient it has buffered data, disassociation and deauthentication, association/re-association request/response, authentication.

* HardMAC(FullMAC) & SoftMAC

HardMAC describes a WNIC which implements the MAC Layer in hardware & SoftMAC refers to a WNIC which does not implement the MAC layer in hardware, rather it expects the drivers to implement the MAC Layer.

Advantages of SoftMAC is the potentially lower hardware costs, upgrade to newer standards by updating the driver only, possibility of correcting faults in the MAC implementation by updating the driver only, one other advantage that applies only in the Linux Kernel at least is that many different drivers for different types of WNIC can all share the same MAC Implementation. However, despite all the advantages, not all WNIC uses SoftMAC.

Mac80211 is the framework within the Linux Kernel for implementing SoftMAC Drivers. It implements the cfg80211 callbacks which would otherwise have to be implemented by the driver itself and implements MAC Layer function which means that it goes between cfg80211 and the SoftMAC Drivers

The Main Advantage of HardMAC is that since MAC functions are implemented through the hardware, they contribute less CPU load.

For HardMAC, the drivers have to implement the cfg80211 interfaces fully themselves.

**Monitoring and Promiscuous Mode**

**This section explains the 2 modes used mainly in our application of Sniffing Data Packet as well as possible limitations faced when using these 2 modes for our application.**

* Promiscuous Mode

It is a mode for a network interface controller (NIC) or wireless network interfaces controller (WNIC) that causes the controller to pass all traffic it receives to the central processing unit for both wired and wireless network, rather than passing only the frames that the controller is specifically programmed to received and this mode is used for network sniffing. However, one issue is that not all wireless driver supports promiscuous mode in addition to this, if the network has any form of encryption such as WEP, WPA/WPA2, packets might be dropped because even if it can capture all traffic on the local network it might not be able to decrypt hence the reason for the packets to be dropped.

However, for promiscuous mode there is a chance in which it might not work. One reason is due to the Adapter and Driver as it is dependent on the implementation of promiscuous mode. If a network is protected with WEP the adapter might be able to show third-party packets, but for WPA/WPA2 it is required that a type of decryption which is like that of Wireshark does. This results in a need to generate “fake Ethernet” header which is based on 802.2 header, and SNAP header if present in data packets which then decryption is required. in order for it to be shown which is a major issue for our application as it runs of android instead of on a Computer and currently Wireshark is unavailable on Android and even if there are alternatives it might not be possible that it runs the same as how Wireshark does it. And most wireless access points are protected using WPA/WPA2 which makes it more difficult for our application when we do the network sniffing. Another issue is that they might not have bothered implementing promiscuous mode even for open networks. Promiscuous mode will not always allow you to see traffic while client isolation is in play.

* Monitor Mode

It is a mode which allows for a WNIC to monitor all traffic received from the wireless network, unlike promiscuous mode, monitor mode allows packets to be captured without having to associate with an access point or ad hoc network first. Monitor mode applies only to wireless network. Limitation is that usually the wireless adapter is unable to transmit in monitor mode and is restricted to a single wireless channel which may be dependent on the driver, firmware and features of chipset. Also, it does not check to see if the Cyclic Redundancy Check (CRC) values are correct for packets captured which means that a chance for the packets to be corrupted is present.

**Nmap (Network Mapper)**

**Some of the below information is extracted from insecure.org by Gordon Lyon**

Apart from knowing the types of modes that the NIC needs be in to Sniff packets there are also other features in which is used in our application, one such feature is called Nmap. It is originally written by Gordon Lyon, where it is used to discover hosts and services on a computer network, which is thus built into a map of the network.

Nmap works by sending specially crafted packets to the target hosts and then analyses the responses from the target. Nmap features other than probing of the network it also includes host discovery and service as well as Operating System Detection. Which is a feature in our application in that every user that uses our application will want to know other than the type of packets being sent, are the types of Operating System that is being used to send these packets. Nmap can adapt to network conditions such as latency and congestion during scan. The features of Nmap also include Port Scanning, such as checking for open ports, Version detection.

Features of Nmap is that it is flexible that supports dozens of advanced techniques for mapping out network filled with IP Filters, firewall, routers and other obstacles, including port scanning mechanisms that work for both TCP & UDP. Nmap is powerful as it has been used to scan up to hundreds of thousands of machines which is considered a huge network, it is also portable as mentioned above that it is flexible as it supports most of the operating systems including Linux, Microsoft Windows, FreeBSD, OpenBSD, Solaris, Mac OS X, Sun OS and more. It is also supported by traditional command line as well as Graphical User Interface (GUI) and Nmap have Binaries available which we used in our application.

**NexMon**

**Some of the below information is extracted from Hackaday by Elliot Williams,**  [**arXiv:1601.07077**](https://arxiv.org/abs/1601.07077)**[cs.OH]**

* Nexmon is what we based our project on as it uses a Nexus 5 Phone with Broadcom BCM 4339 WiFi Chipset to do monitoring as well as packet-injecting as NexMon is Open. NexMon gives full control over a Wi-Fi Chip for research purposes is limited by firmware, which means it is hard to evolve communication protocols and test schemes in practical environments. Monitor mode as mentioned above allows eavesdropping on all frames on a wireless communication channel. This includes uses such as network packet analyses, security research and testing of new medium access control layer protocols, it is offered by SoftMAC drivers that implement the media access control sublayer management entity (MLME) in the driver rather than in the Wi-Fi Chip. On smartphones, mostly FullMAC (HardMAC) chips are used to reduce power consumption as the tasks do not need to wake up the main processor. Even though it is possible in FullMAC situations, it is generally not implemented in today’s Wi-Fi firmware’s used in smart phones which is one of the limitations that our project is facing, that is why in such situations we are bringing monitor mode to Nexus 5 Smartphones to enhance the interoperability between applications that require monitor mode and the BCM4339 Wi-Fi Chips.

**Some of the below information is from a dissertation by Matthias Thomas Schulz**

**Nexutil & ioctls**

**In this section we talk about nexutil which is being used by our applications and include some details on what it can do**

* Nexutil is one of the utilities/libraries required for controlling the firmware as well as imitate a monitor interface to unmodified analysis and penetration testing (TCPDump)
* Talking to the firmware for many applications it is helpful to configure a firmware during runtime or extract information for debugging purposes one such means is to use ioctls to control the firmware and send events from the firmware to the host. To initiate transfers from the firmware, a user-space program known as nexutil can initiate a synchronous data exchange with the firmware by calling ioctls in the firmware. Each iotcl contain a command number, pointer to buffer to exchange data and length of the buffer, ioctls can either only set data or get data back from the firmware. For the two directions set and get nexutil offers two parameters -s<command\_number> and -g<command\_number> and may pass either integers, strings, raw data from the Standard input or based64 encoded raw data to the firmware. While ioctls are always initiated by the host, Wi-Fi firmware can also create an event and send it as a message to the host, where it is handled in the driver.
* FullMAC Wi-Fi Chips consist of two processers, one is ARM Processor that performs not time critical tasks (implemented in the driver for SoftMAC). The second processor implements a programmable state machine (PSM) which runs in the D11 core which is responsible to quickly process MAC-Layer Events. The D11 core then decides which frames should be dropped or which should be answered by acknowledgements. In Broadcom Wi-Fi Chips the D11 core does real-time MAC Processing as well
* Nexutil (handle receptions)

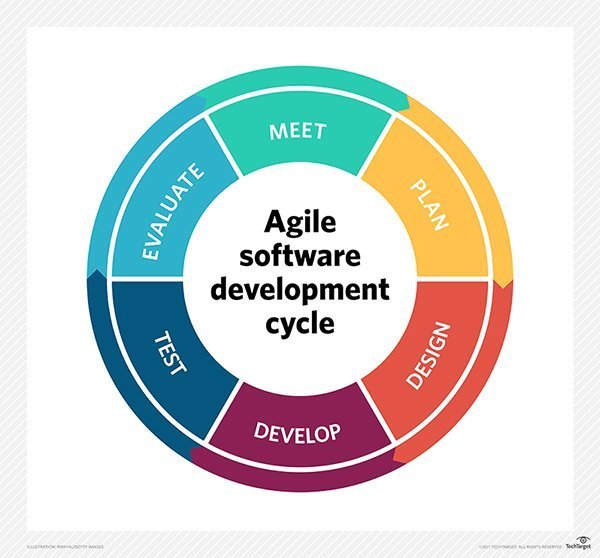
When monitor mode is active (calling nexutil -ml), this unfunction will call a wlc\_monitor function that will extract the received statistics and write them into a structure which will then pass both the statistics and frame to the wlmonitor function, which is a function that is hooked to implement monitor mode with radiotap headers

# 6. User Interface Designs

(application interface designs to be added)

# 7. Software Development

## 6.1 Software Design Process



Agile (Scrum) is used to develop our application, it provides the ability to create and respond to change to succeed in an uncertain and turbulent environment. Agile is the most applicable due to the unpredictability of the application due to rooting and specific modes based on the WLAN cards of the type of devices that we plan to incorporate into. With Agile we can makes changes as and when necessary to help overcome such issues.

## 6.2 Development Tools and Environment

* Android Studio
* UMLet
* Other possible tools we used

## 6.3 Schedule of Project

## 6.4 Roles & Responsibilities

|  |  |
| --- | --- |
| Team Members | Roles |
| Soh Yu Xuan | Team Leader, Lead Programmer |
| Timothy Chin | Assistant Programmer / Lead UI |
| Kenneth Huang | Assistant Programmer / Lead Tester |
| Kendrick Tan | Project Management / Documenter |

## 6.5 Version Control

Github is used as the version control software during the development of this project. Both source code and documents were managed using Github.

# 7. Application Test

## 7.1 Test Cases

# 8. References

[1] <https://www.keycdn.com/support/udp-file-transfer/>

[2] <https://www.lifewire.com/tcp-headers-and-udp-headers-explained-817970>

[3] <https://www.keycdn.com/support/tcp-flags/>

[4] <https://opensourceforu.com/2013/09/what-a-native-developer-should-know-about-android-security/>

[5] <https://nmap.org/book/tcpip-ref.html>

[6] http://www.erg.abdn.ac.uk/users/gorry/course/inet-pages/arp.html