

**ISTANBUL TECHNICAL UNIVERSITY ★ INFORMATICS INSTITUTE**

**NETWORK PACKET CAPTURING  
FOR WINDOWS OPERATING SYSTEM**

**M.Sc. THESIS**

**Yücel AYDIN**

**Department of Applied Informatics**

**Cyber Security Engineering and Cryptography Programme**

**JUNE 2017**



**ISTANBUL TECHNICAL UNIVERSITY ★ INFORMATICS INSTITUTE**

**NETWORK PACKET CAPTURING  
FOR WINDOWS OPERATING SYSTEM**

**M.Sc. THESIS**

**Yücel AYDIN  
(707141011)**

**Department of Applied Informatics**

**Cyber Security Engineering and Cryptography Programme**

**Thesis Advisor: Assoc. Prof. Dr. Enver ÖZDEMİR**

**JUNE 2017**



**İSTANBUL TEKNİK ÜNİVERSİTESİ ★ BİLİŞİM ENSTİTÜSÜ**

**WINDOWS İŞLETİM SİSTEMLERİNDE AĞ PAKETİ YAKALAMA**

**YÜKSEK LİSANS TEZİ**

**Yücel AYDIN  
(707141011)**

**Bilişim Uygulamaları Anabilim Dalı**

**Bilgi Güvenliği Mühendisliği ve Kriptografi Programı**

**Tez Danışmanı: Doç. Dr. Enver ÖZDEMİR**

**HAZİRAN 2017**



Yücel AYDIN, a M.Sc. student of ITU Informatics Institute student ID 707141011, successfully defended the thesis entitled “Network Packet Capturing for Windows Operating System”, which he prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

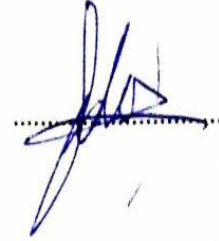
**Thesis Advisor :**    **Assoc. Prof. Dr.Enver ÖZDEMİR**  
Istanbul Technical University



**Jury Members :**    **Prof. Dr. Lutfiye DURAK ATA**  
Istanbul Technical University



**Assis. Prof. Dr. Erdinç ÖZTÜRK**  
Istanbul Commerce University



**Date of Submission : 05 May 2017**

**Date of Defense :     05 June 2017**





*To my spouse Latifah,*



## **FOREWORD**

I would like to express my deep appreciation and thanks for my advisor. This work is supported by ITU Institute of Informatic.

June 2017

Yücel AYDIN



## TABLE OF CONTENTS

|                                                                | <u>Page</u> |
|----------------------------------------------------------------|-------------|
| <b>FOREWORD .....</b>                                          | <b>ix</b>   |
| <b>TABLE OF CONTENTS.....</b>                                  | <b>xi</b>   |
| <b>ABBREVIATIONS .....</b>                                     | <b>xiii</b> |
| <b>LIST OF TABLES .....</b>                                    | <b>xv</b>   |
| <b>LIST OF FIGURES .....</b>                                   | <b>xvii</b> |
| <b>SUMMARY .....</b>                                           | <b>xix</b>  |
| <b>ÖZET.....</b>                                               | <b>xxi</b>  |
| <b>1. INTRODUCTION.....</b>                                    | <b>1</b>    |
| 1.1 Purpose of Thesis .....                                    | 2           |
| 1.2 Background .....                                           | 2           |
| <b>2. WINDOWS INTERNALS.....</b>                               | <b>5</b>    |
| 2.1 User Mode and Kernel Mode .....                            | 5           |
| 2.2 User Space and System Space.....                           | 5           |
| 2.3 System Calls.....                                          | 6           |
| 2.5 Windows Architecture.....                                  | 8           |
| 2.6 Network Driver Interface Specification (NDIS) .....        | 9           |
| 2.7 The Path from NIC to Applications .....                    | 11          |
| 2.8 Journey of Network Packet in Internet.....                 | 11          |
| <b>3. NETWORK PACKET CAPTURING SYSTEMS.....</b>                | <b>13</b>   |
| 3.1 Berkeley Packet Filter .....                               | 13          |
| 3.2 Network Packet Capturing for UNIX Systems .....            | 15          |
| 3.3 Network Packet Capturing in Android Operating Systems..... | 19          |
| 3.4 WinPcap Windows Packet Capturing Library .....             | 20          |
| 3.5 Packet Encapsulation.....                                  | 26          |
| 3.6 Costs and Optimization of WinPcap .....                    | 26          |
| <b>4. EXPERIMENTAL RESULTS AND RISK ANALYSIS .....</b>         | <b>29</b>   |
| 4.1 Experimental Results.....                                  | 29          |
| 4.2 Risk Analysis.....                                         | 32          |
| <b>5. CONCLUSIONS AND RECOMMENDATIONS.....</b>                 | <b>33</b>   |
| 5.1 Practical Application of This Study .....                  | 33          |
| 5.2 Future Works.....                                          | 35          |
| <b>REFERENCES .....</b>                                        | <b>37</b>   |
| <b>APPENDICES .....</b>                                        | <b>39</b>   |
| APPENDIX A.1 .....                                             | 40          |
| <b>CURRICULUM VITAE.....</b>                                   | <b>43</b>   |



## ABBREVIATIONS

|                |                                          |
|----------------|------------------------------------------|
| <b>API</b>     | : Application Programming Interface      |
| <b>BPF</b>     | : Berkeley Packet Filtering              |
| <b>CSPF</b>    | : CMU/Stanford Packet Filtering          |
| <b>DPC</b>     | : Deferred Procedure Call                |
| <b>DLL</b>     | : Dynamic Link Library                   |
| <b>HAL</b>     | : Hardware Abstraction Layer             |
| <b>IP</b>      | : Internet Protocol                      |
| <b>LibPcap</b> | : Library for Packet Capturing           |
| <b>NDIS</b>    | : Network Device Interface Specification |
| <b>NIC</b>     | : Network Interface Card                 |
| <b>NPF</b>     | : Netgroup Packet Filtering              |
| <b>OS</b>      | : Operating System                       |
| <b>TCP</b>     | : Transmission Control Protocol          |
| <b>UDP</b>     | : User Datagram Protocol                 |
| <b>WinPcap</b> | : Windows Packet Capturing               |





## LIST OF TABLES

|                                                                            | <b><u>Page</u></b> |
|----------------------------------------------------------------------------|--------------------|
| <b>Table 3.1</b> : Comparison of libraries for Unix Operating System. .... | 18                 |
| <b>Table A.1</b> : PreRequirements for Application.....                    | 41                 |



## LIST OF FIGURES

|                                                                  | <u>Page</u> |
|------------------------------------------------------------------|-------------|
| <b>Figure 2.1</b> : Relations between user and kernel mode. .... | 5           |
| <b>Figure 2.2</b> : 32 Bit System Space. ....                    | 6           |
| <b>Figure 2.4</b> : System Calls. ....                           | 7           |
| <b>Figure 2.5</b> : Windows Architecture .....                   | 8           |
| <b>Figure 2.6</b> : NDIS Wrapper [9]. ....                       | 10          |
| <b>Figure 3.1</b> : Tree Model .....                             | 13          |
| <b>Figure 3.2</b> : Acyclic Control Flow (ACF) Method .....      | 13          |
| <b>Figure 3.3</b> : BPF Architecture .....                       | 15          |
| <b>Figure 3.4</b> : DashCap Architecture [14]. ....              | 17          |
| <b>Figure 3.5</b> : nCap Architecture [14]. ....                 | 17          |
| <b>Figure 3.6</b> : Packet Loss Comparison [13]. ....            | 19          |
| <b>Figure 3.7</b> : LibPcap Work Flow .....                      | 20          |
| <b>Figure 3.8</b> : WinPcap Work Flow .....                      | 24          |
| <b>Figure 3.9</b> : Netgroup Packet Filter .....                 | 25          |
| <b>Figure 3.10</b> : Encapsulation .....                         | 26          |
| <b>Figure 4.1</b> : Network Packet Capturing by packet.dll ..... | 30          |
| <b>Figure 4.2</b> : Network Packet Capturing by wpcap.dll .....  | 31          |
| <b>Figure 5.1</b> : Packet Capturing with WinPcap .....          | 34          |
| <b>Figure 5.2</b> : Packet Capturing with LibPcap .....          | 34          |
| <b>Figure A.1</b> : packetCapture Application. ....              | 40          |



# **NETWORK PACKET CAPTURING FOR WINDOWS OPERATING SYSTEM**

## **SUMMARY**

Today there are several network security systems used by the end users at the application layer. Network security systems are designed to detect the attacks or to measure the intensity of the network traffic. The first step for the network systems is to detect the malicious traffic coming to the network or going from the network. Detecting is possible with analyzing the packets whether they are malicious or normal network packets with the user level applications. Diagnosis tools can be created by inspecting of each packet in the network.

In this study, I present the detailed methods to capture the network traffic in Windows Operating Systems. The second chapter is devoted to explain the parts of the Windows OS' kernel in charge of network. Network packet capturing process for different kind of operating systems and Winpcap used by the most popular network analysis tools are discussed in the third part of the study. The evaluations and analysis with a simulation of packet capturing are presented in the fourth chapter. A detail explanation of the network packet capturing application written in C++ programming language and future works are presented in the conclusion part.

It is through network packets that computers can communicate with each other within a network. Any input from the outside world to the computer is input to the computer through a hardware device. The packages come in the same way via a computer network cable and enter the operating system from the network card.

The Windows operating system restricts the operations that the user can do, unlike open source operating systems. In open source operating systems, users can interact directly with the kernel that makes up the operating system.

When network packages need to be captured and analyzed, application developers place one packet capture driver in the operating system kernel and also develop new libraries to allow user-level applications to contact this driver. The speed and accuracy of network packet capture systems directly impact the applications that will analyze the network packets. Because; There may be content that will damage the system in any unreachable network package and the analysis system may not be aware of this package.

The performance of network packet capture systems is directly related to the buffering and filtering systems used in the system. If a good buffer and filter are not used, many network packets are transmitted directly to the operating system without being caught.

There are many network packet capture systems developed in open source operating systems. However, most of the normal and server computers used by many users and companies nowadays include the Windows operating system. Because of this, a network packet capture system was required for Windows operating system. As a result of the work done, WinPcap network packet capture library, which is preferred by many applications today, has been developed.



# WINDOWS İŞLETİM SİSTEMLERİNDE AĞ PAKETİ YAKALAMA

## ÖZET

Günümüzde uygulama seviyesinde yapılmış birçok ağ güvenlik sistemi bulunmaktadır. Bunlar saldırganı tespit etmek, saldırı trafiğinin yoğunluğunu tespit etmek gibi birçok amaç için tasarlanmıştır. Yapılan her uygulamada ilk aşama bilgisayara veya ağa gelen trafiğin tespit edilmesidir. Bunun hayata geçirilmesi için ise trafiği oluşturan paketlerin elde edilmesi şarttır. Elde edilen paketlerin içeriği yorumlanarak gerekli önleme yöntemleri ortaya koyulmaktadır.

Bu çalışma Windows işletim sistemlerindeki ağ trafiğini elde edebilecek bilgileri içermektedir. Çalışmanın ikinci bölümünde Windows sistemlerindeki ağ trafiğini anlayabilmek için sistemi oluşturan işletim sisteminin içerisindeki yapılar incelenmiştir. Üçüncü bölümde ise bugüne kadar farklı işletim sistemlerinde ağ trafiğinin elde edilmesi için yapılmış uygulamaların detayları verilmiş ve Windows sistemlerinde uygulanan ve birçok kullanıcı tarafından tercih edilen ağ analiz programları tarafından kullanılan WinPcap alt yapısı incelenmiştir. Dördüncü kısımda Ağ paketlerinin yakalanması simule edilerek hazırlanan analiz ve değerlendirmelere yer verilmiştir. Sonuç kısmında ise Windows sistemlerinde kullanılacak C++ programlama dili ile yazılmış uygulamanın detayları ve ileride kullanım şekilleri açıklanmıştır.

Bir ağ içerisinde bilgisayarların birbirleriyle irtibat kurup iletişim sağlaması ağ paketleri sayesinde olmaktadır. Ağ paketi, kullanıcı veya işletim sistemi tarafından oluşturulmuş ham bilginin yada verinin iletişimin sağlanması için gerekli ek bilgilerin eklenmesiyle oluşmaktadır. Bilgisayara dış dünyadan gelen her türlü girdi bir donanım sayesinde bilgisayara giriş yapmaktadır. Paketler de aynı şekilde bir bilgisayara ağ kabloları vasıtasıyla gelmekte ve ağ kartından işletim sistemine giriş yapmaktadır. Ağ kartının kilobyte seviyesinde hafızası bulunması nedeniyle paketler ağ kartı tarafından ön tespit yapıldıktan sonra işletim sistemi hafızasına kaydedilmektedir. Ön tespit olarak ağ kartı sadece paketin içerisinde bulunan ethernet bilgilerini alıp gerekli hash algoritmasıyla işleyip elde ettiği bilgi ile ethernet bilgilerinin son hanesi olan frame kontrol bilgisiyle karşılaştırma yapmaktadır. Uygun olmayan paketleri ağ kartı işletim sistemine iletmeden elemektedir.

Windows işletim sistemi diğer işletim sistemlerinin aksine kullanıcının yapabileceği işlemleri kısıtlamaktadır. Diğer işletim sistemlerinde kullanıcılar işletim sistemini oluşturan çekirdek ile doğrudan etkileşim sağlamaktadır. Bu özellik doğal olarak diğer işletim sistemlerinde ağ paketleri ağ kartından çekirdeğe geçtiğinde kullanıcılar tarafından direkt olarak elde edilip gerekli analizler yapılabilmektedir. Windows işletim sisteminde ise işlem bu kadar basit değildir.

Windows işletim sistemi kullanıcı bölgesi ve çekirdek bölgesi olacak şekilde iki ana yapıya ayrılmıştır. Normal kullanıcılar sadece kullanıcı bölgesinde yetki verilen işlemleri yapmaktayken çekirdek ile direk etkileşimleri bulunmamaktadır. Eğer kullanıcıların veya kullanıcı bölgesinde işleyen bir uygulamanın çekirdek ile etkileşim sağlaması gerekli ise çekirdek içerisinde bu etkileşimi sağlayacak bir sürücünün bulunması gereklidir. Sürücü çekirdek içerisinde gerekli işlemleri gerçekleştirirken

kullanıcı bu sürücü ile yine bu etkileşim için programlanmış dinamik bağlantı kütüphaneleri (dbk) yardımıyla gerekli irtibatı sağlamaktadır.

Her ağ kartı işletim sistemi içerisinde kendisine has olacak şekilde bir ağ sürücüsü ile birlikte bulunur. Bu sürücü vasıtasıyla ağ kartı işletim sistemine gerekli işlemleri yaptırır. Ağ kartı gelen paketleri sistem hafızasına kopyaladıktan sonra ağ sürücüsünü uyararak yeni bir paketin geldiğini haber verir. Ağ sürücüsü ise üst seviye sürücülerinden ağ paketlerini kullanacak tüm sürücülere paketin bulunduğu bilgisini verir ve üst sürücüler gerekli işlemleri yaparlar.

Ağ paketlerinin yakalanıp analiz yapılması gerekliliğinde ise uygulama geliştiriciler işletim sistemi çekirdeğine bir adet paket yakalama sürücüsü yerleştirir ve ayrıca kullanıcı seviyesindeki uygulamaların bu sürücü ile irtibatını sağlamak için yeni kütüphaneler geliştirirler.

Ağ paket yakalama sistemlerinin hızı ve doğruluğu aynı zamanda analiz yapacak uygulamaları doğrudan etkilerler. Çünkü; elde edilemeden geçen herhangi bir ağ paketi içerisinde sisteme zarar verecek içerik bulundurabilir ve bu paketten de analiz sisteminin haberi olmayabilir.

Ağ paket yakalama sistemlerinin performansı sistem içerisinde kullanılan tamponlama ve filtremele sistemleriyle doğrudan ilgilidir. İyi bir tampon yada filtre kullanılmaz ise bir çok ağ paketi yakalanmadan doğrudan işletim sistemine iletilmiş olmaktadır. İlk zamanlarda tampon olarak iki adet sabit boyutlu tampon çekirdek içerisine yerleştirilmiştir. Ağ kartından gelen paketler önce bir tamponda depolanmıştır ve bu tampon dolduğu zaman tüm veriler diğer tampona aktarılmıştır. Kullanıcı seviyesindeki uygulamalar ise doğrudan bu tampon içerisindeki bilgilere ulaşmıştır. Ancak bu sistem günümüz ortamında pek uygulanabilir olmamıştır. Günümüzde kullanılan ağ trafiği çok hızlı ve yoğun bir şekilde işlemektedir. Bir saniye içerisinde milyonlarca paket sistem içersine gelmek de veya sistemden uzaklaşmaktadır. Eğer kullanıcı seviyesindeki uygulama gelen paket hızı kadar bir hızla paketleri tampon içerisinden alamazsa ilk tampon içerisine gelen yeni paketler yakalanamamaktadır. Bu dezavantajı gidermek için sirküler tampon sistemi geliştirilmiştir. Bu sistem de sadece bir adet tampon kullanılmış ve bu tamponun boyutu dinamik olarak değiştirilmiştir. Ağ kartından gelen paketler olduğu sürece tampon boyutu artarken kullanıcı seviyesindeki uygulamalar paketleri aldıkça boyut azalmıştır.

Ağ paket yakalama sistemlerindeki diğer bir kritik unsur ise filtreleme sistemidir. Çoğu kullanıcı veya uygulama ağ paketlerini analiz ederken tüm paketleri yakalamak istemez. Sadece gerekli olan paketlerin yakalanması sistem için yeterlidir. Bu ayırt etme işlemi ise filtreleme sistemiyle mümkün olmaktadır. Yakalanacak paket sayısının azaltılması vasıtasıyla ayrıca tampon içerisinde yerleştirilecek paket sayısı azaltılmak da ve tamponlama sistemi de rahatlatılmaktadır. Filtreme sistemi kullanıcı uygulamasının hangi paketlerin yakalanması gerektiğini tanımlamasıyla başlar. Bu tanımlama gerekli kütüphaneler vasıtasıyla çekirdek içerisindeki sürücüye iletilir. Sürücü ise ağ paketleri daha ağ kartı içerisindeyken bu filtreleme işlemini yapar ve sadece kullanıcı uygulamasının istediği paketler tampon içersine kopyalanır.

Açık kaynak işletim sistemlerinde geliştirilen birçok ağ paket yakalama sistemi bulunmaktadır. Ancak günümüzde birçok kullanıcı ve şirketin kullandığı normal ve



server bilgisayarlarının çoğu Windows işletim sistemini içermektedir. Bu sebepten dolayı Windows işletim sistemi için de bir ağ paket yakalama sistemi gerekliliği doğmuştur. Yapılan çalışmalar sonucunda günümüzde de bir çok uygulamanın tercih ettiği WinPcap ağ paket yakalama kütüphanesi geliştirilmiştir. Wireshark, TCPDump vb. Birçok popüler uygulama da kendi sistemlerinde bu kütüphaneyi kullanmaktadır.

WinPcap kütüphanesi Fulvio Rises ve Loris Degionani isimli iki italyan bilim insanı tarafından geliştirilmiştir ve daha sonraki yıllarda optimize çalışması yapılmıştır. WinPcap kütüphanesi üç ayrı ancak birbiriyle irtibatlı parçalardan oluşmaktadır. Bu parçalar Çekirdek Sürücüsü, Packet.dll ve Wpcap.dll parçalarıdır.

Çekirdek sürücüsü kullanıcı tarafından tanımlanmış filtreye uygun olarak filtreleme yapıp kullanıcının istediği paketleri sirküler tampon içerisine kopyalamaktadır. Aynı zamanda işlem kolaylığı ve hız sağlamak maksadıyla sürücü içerisinde bir istatistiksel makine tutmaktadır. Bu makine gelen ve giden paketlerle ilgili istatistiksel bilgileri kullanıcının istemesi durumunda derhal sağlamak maksadıyla hazır etmektedir. Diğer bir sürücü görevi ise gelen giden tüm paketlerinin içeriği ile ilgili tüm bilgileri kullanıcı seviyesine çıkmadan direkt olarak hardisk içerisine kaydetmektedir.

Packet.dll unsuru ise Windows işletim sistemi içerisindeki kullanıcı bölgesi ile çekirdek bölgesi arasında irtibatı sağlamaktadır. Çekirdek içerisindeki sürücüye gerekli direktifleri göndererek kullanıcı uygulamasının istediği işlemleri gerçekleştirmektedir.

Wpcap.dll unsuru üst seviye işlemler için geliştirilmiş bir kütüphanedir. Çekirdek tarafından kullanılacak filtrenin oluşturulması, kullanıcı uygulamaları ile irtibatların sağlanması vb. işlemler bu kütüphane tarafından sağlanmaktadır.

Windows işletim sisteminin incelenmesinin yanında çalışmanın üçüncü bölümünde diğer işletim sistemlerindeki ağ paket yakalama sistemleriyle ilgili çalışmalar incelenmiştir.

Günümüzde Windows işletim sisteminin yaygın olarak kullanılmasının yanında bireysel kullanım bazında mobil sistemlerin gelişmesiyle Android işletim sistemlerinin kullanılması da artmıştır. Android işletim sistemlerinde ağ paketinin yakalanması ile ilgili çalışmalar yapılmıştır. Bu çalışmalar Android işletim sisteminin linux tabanlı bir işletim sistemi olması sebebiyle açık kaynak işletim sistemleriyle ilgili geliştirilmiş sistemler kullanılmıştır. Linux işletim sisteminde ağ paketi yakalamak konusunda popüler olan libpcap kütüphanesinin Android işletim sistemine uyarlanması çalışmaları yapılmıştır. LibPcap kütüphanesinin C programlama dili ile yazılmış olması ve Android işletim sisteminde Java programlama dili kullanımı geliştiriciler için problemler çıkartmıştır. Bu problemin aşılması ise Android işletim sistemlerinde kullanılan programlama dilleri arasında dönüşüme imkan sağlayan Java Native Interface kullanılmıştır.

Yapılan çalışma ile aynı zamanda gerekli test çalışmalarının yapılabilmesi için C++ programlama dili kullanılarak WinPcap kütüphanesi tabanlı bir uygulama geliştirilmiştir. Geliştirilen uygulama ile açık kaynak işletim sistemlerindeki ağ paket yakalama sistemleri karşılaştırılmış ve gerekli sonuçlar ilgili bölümlerde verilmiştir. Genel olarak bakıldığında Windows işletim sistemi için geliştirilen WinPcap

kütüphanesi son optimize çalışmalarıyla birlikte açık kaynak sistemlerine göre üstünlük göstermekte ancak açık kaynak sistemlerinde sistem geliştirilmesi için birçok araştırma yapıldığı ve yapılacağı görülmektedir. Geliştirilen uygulama Windows işletim sisteminde simule edilmiş ve 10 Ghz kadar ağ trafiği altında çalışabilirliği ve kullandığı kaynak miktarı gözlenmiştir.

## 1. INTRODUCTION

The advance of technology connected the people throughout the world by internet. Also, there are huge number of network infrastructure connecting several people, commercial companies, military and government. Internet itself allowed for many security threats to occur. Therefore, network security has become so much important to the world because; sensitive information can be reached through the internet. A data from source computer to the destination computer should be secured well. Otherwise; an attacker can obtain the data or send another data from the channel.

An efficient network monitoring tool is a crucial need for all kind of network systems. Network monitoring is the collection of information for network management purposes. An efficient network monitoring tool should provide a real time monitoring for administrators. And also a network monitoring tool should have comprehensive capabilities for analysis purposes in order to see the traffic clearly and quickly.

Important features of network monitoring are real-time capturing which is to show the traffic in the monitoring application when the packet arrived the computer, displaying type of information such as list of coming packets, protocol distribution charts etc. Meanwhile, packet capturing is very crucial component for network monitoring systems. Most of the networks nowadays are working with high traffic load and in this kind of traffic capturing all packets without losing even one is the most essential functionality in network monitoring systems. Because; the packets that were not captured by the system could have viruses, worms and others that can affect the network.

With the improvements in the technology, several packet capturing systems have been produced by the vendors in either hardware based or software based. Even though the hardware based systems provide high capturing rates, deployment of hardware for all systems in the network needs huge amount of budget. Therefore, scientists has focus on software based systems more than hardware systems [1].

## **1.1 Purpose of Thesis**

The main objective for this study to elaborate the internal infrastructure of the systems built for packet capturing purposes and to create a basic packet capturing system to be used by network intrusion systems.

## **1.2 Background**

Network Monitoring tools are called sniffers. Monitoring tools are the helpers for administrators to manage and administer the network. Traditionally, these tools are used for evaluating network related problems, network intrusion systems and network traffic logging. While network monitoring tools are passive listeners, network intrusion systems can respond the malicious traffic.

Usually, monitoring applications put the network card of a computer into promiscuous mode. This enables the computer to listen all subsection traffic in the network. Also a filter can be produced from the information inside headers and it can be used for capturing only a specific packet such as; capturing packets coming from port 80 [2].

There are two basic network monitoring approaches, active monitoring and passive monitoring. Active monitoring is to inject control packets inside the network. It costs additional traffic inside the network. Active monitoring provides full control regarding monitoring interval, packet size and path to be monitored. A ping search can be considered as active monitoring method.

Passive Monitoring is to observe the existing network traffic without injecting any packets in the network. There is no control over monitoring action for passive monitoring. For passive monitoring there is no choice except underlying network protocol.

Most of the traffic in a network consist of three underlying protocol Internet Control Message Protocol (ICMP), Transmission Control Protocol (TCP) and User Datagram Protocol (UDP).

ICMP is underlying protocol for famous ping utility. The advantage of ICMP is simplicity and ease of use. Some of the protocols need to establish connection ,which is called handshake ,before sending data. ICMP protocol doesn't have any handshake requirement.

TCP is the underlying protocol for many internet applications such as WWW and File Transfer Protocol(FTP). Unlike ICMP, TCP needs handshake. It is a three way process. First the source host sends a request to initiate the connection which is called TCP SYN (Synchronization). Destination host replies the request as TCP ACK (Acknowledged). Lastly, source host sends a SYN ACK packet to the destination host.

A TCP packet consists of 40 bytes header (20 Byte IP Header + 20 Byte TCP Header). The TCP protocol consumes more bandwidth than other protocols because of its large header size.

The applications which need high reliability use TCP protocol. Because; TCP has functionality to rearrange the packets in the right order at the destination host. Also TCP provides retransmission if the packets reach the destination with loses.

UDP packet consists of 28 Byte header. It is a connectionless protocol unlike TCP. It doesn't need any handshake. UDP consumes less bandwidth than TCP because of its 28 Bytes header [3].

All computers in a network gets the network packets with their own Network Interface Cards (NICs). Network packets travel through several components inside the operating systems from NIC to the applications that we use in our computers. Also when an application needs to interact with outside, operating system creates a network packet through the application to the NIC.

Network analysing tools such as Snort should capture network packets before analysing them. The most important part in this process is buffering and filtering of the related packets. Buffering is keeping the packets inside the memory before sending them to the other part of the operating system. Due to the high amount of the data, if the packets were sent from a part to another part without buffering it would cost high amount of microprocessor. Therefore, some of the packets may be dropped before reaching the applications by the operating system due to the bottleneck for the microprocessor.

There are several kind of network protocols reaching from outside to the internal networks. Most of the network monitoring applications want to track specific protocols and also delivering all packets to the applications needs a strong microprocessor. Therefore, filtering of specific network protocol has a critical importance for network monitoring applications.

There are several libraries providing this functionality for users but when comparison is done, WinPcap is the most preferred one by the diagnosis tools.

The first known packet filtering and network monitoring system was CMU/Stanford Packet Filtering (CSPF) [4]. It provided access to the data link layer and was used by most of the other applications as a starting point for their systems.

McCane and Van Jackson improved CSPF in 1993 and released the Berkeley Packet Filter (BPF) [4]. Basically their improvement was dropping the network packets according to user defined filter while the packet was still in NIC. Today, most of the Berkeley Software Distribution (BSD) operating systems such as UNIX use BPF as a default network packet capturing system.

The Mach Packet Filter, PathFilter are some examples for studies aimed to improve the filter inside the BPF.

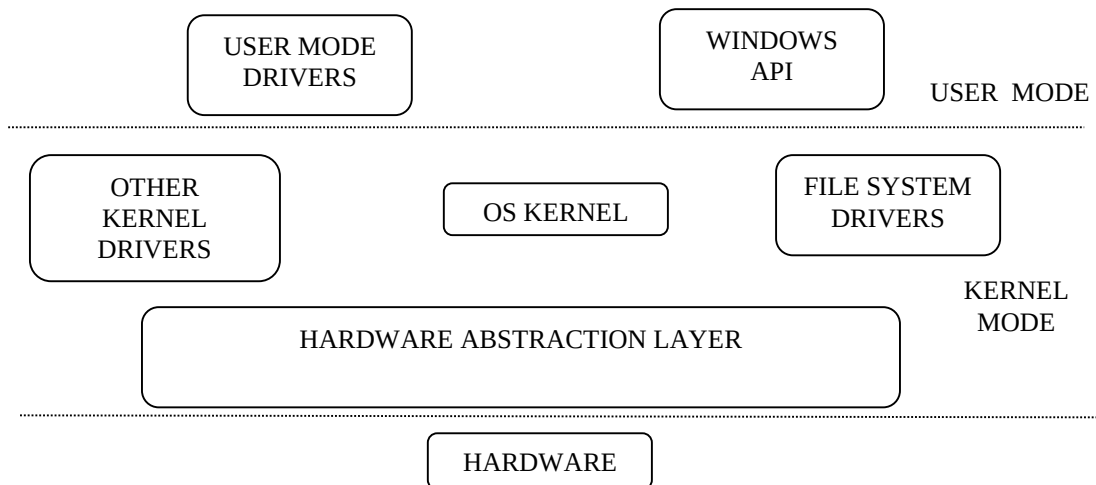
A few studies focused on the other aspects of packet filtering, such as buffering and copying. A study proposed to use a shared buffer or a bigger buffer to limit the copy time for network packets. WinPcap which was built for Windows Operating Systems is the most robust library with its efficient buffering system .

## 2. WINDOWS INTERNALS

### 2.1 User Mode and Kernel Mode

A processor has two different modes in windows operating systems. According to the code, the mode of processor is shifted. User applications run in user mode while core operating system components run in kernel mode. Many drivers also run in kernel mode [5].

Windows allocates virtual address space for each application that runs in user mode. This property protects one application from the other one. If one application crashes, it affects only that application not the other one. All codes running in kernel mode use only one virtual address space. One code can affect or destroy the other one. If one code crashed all operating system would crash.



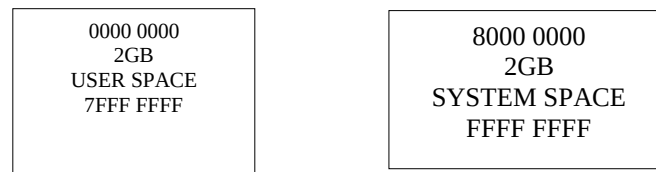
**Figure 2.1 :** Relations between user and kernel mode.

### 2.2 User Space and System Space

As we mentioned before, each code running in user mode uses different spaces. These spaces are called USER SPACE. But many drivers and system components run in kernel mode and they also use different spaces. These spaces are called SYSTEM SPACE.

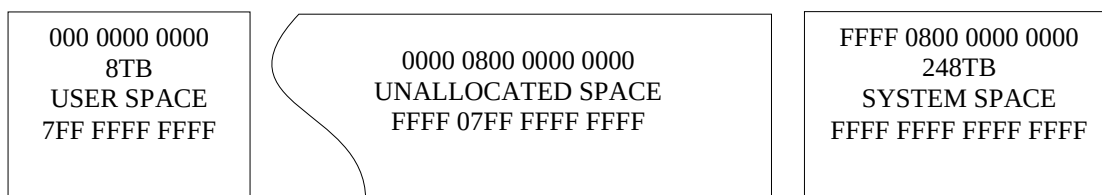
Normally 32 bit Windows has  $2^{32} = 4\text{GB}$  total space. As default, 2GB memory is allocated for user space and 2GB for system space. This default option can be changed

before running the operating system i.e. before booting. For example; 3GB space can be allocated for user and 1GB for system space [6].



**Figure 2.2 : 32 Bit System Space.**

For 64 bit systems there are  $2^{64} = 16$  Exabyte space totally. 8TB space is allocated for user and 248 TB are allocated for system [6].



**Figure 2.3 : 64 Bit System Space.**

Codes run in user mode have only access to user space not to system space but codes run in system space have access to both system and user spaces. Therefore, codes or components that run in system space should be designed very carefully to read or write to user space. They can interrupt processes running on the user space.

All pages in user space can be transferred to disk and bring back but in system space some pages (paged pool) can be transferred, some pages (non paged pool) cannot. The places for paged and non paged pools are created in system space dynamically [6].

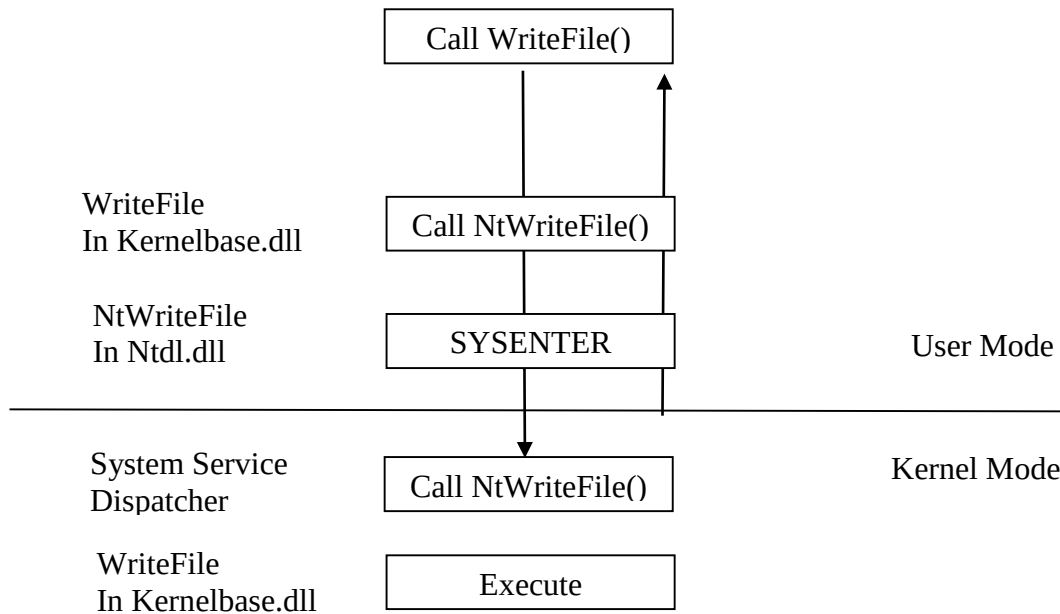
## 2.3 System Calls

When a Windows application creates a function which needs to work in the kernel it comes first to the kernelbase.dll library to translate the function to the kernel understandable function. Then the function is transferred to the ntdll.dll library. This library includes all kernel functions with a specific number. This point is the entrance point to the kernel mode from user mode [7].

Ntdll.dll creates a system call to change the CPU mode from user to kernel mode. It transmits the function number to the System Service Dispatcher inside the kernel. Dispatcher includes a table whose inputs are the numbers coming from system calls and



the outputs are the related kernel function. The system call is sent to the related execution exe file inside the kernel. After execution the result turns back to the user level application through the same way. Figure 2.6 illustrates an example for writing a file [7].



**Figure 2.4 : System Calls.**

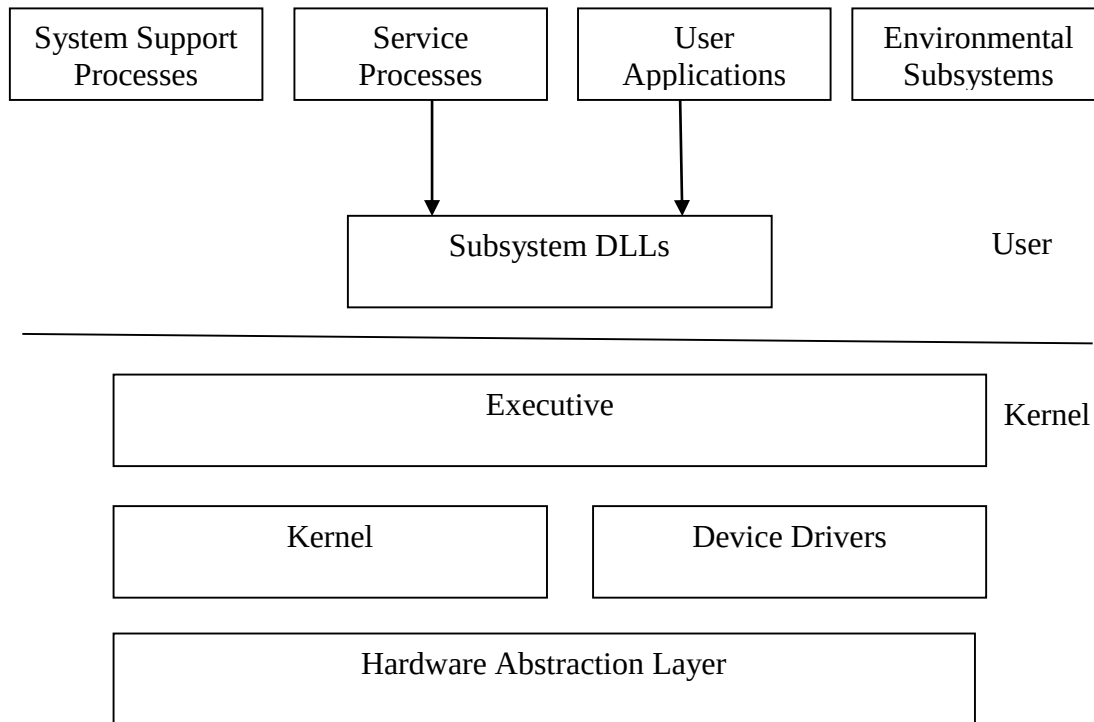
## 2.4 Mini Drivers, Miniport Drivers and Driver Pairs

(Miniport Driver, Port Driver) This structure is shown at the left side is a driver pair. Although they are different drivers, they are seen as one driver by all of the elements in the network. Miniport Driver focuses on general works for a driver while port driver handles with specific works [8].

When a driver is loaded, it first implements GsDriverEntry function. The function makes some initial synchronization and calls the DriverEntry function. DriverEntry function fills the device object with the functions that related driver will use. Device Objects consists of Unload and Major Function that is the need to handle with IRPs [8]. Having experiences with Windows, the developers understand that some functions are all the same for most of the drivers and also creating these functions for some vendors who want to create a specific driver for themselves can be hard. Because of these reasons Microsoft built driver pairs. Driver pairs consist of two parts. First part is miniport driver that conducts common works and the second part is port driver that conducts the works that special vendors want to [8].

The way for implementing with driver pairs is first to give all works to the miniport driver and then if miniport driver can't handle with request, it asks help from port driver [8].

## 2.5 Windows Architecture



**Figure 2.5 : Windows Architecture**

Windows has a hierarchical system to implement all complex works. Normally, it is restricted to call a native windows system directly. When a service process or a user wants to call a kernel service it is only possible by the help of subsystem dynamic link libraries. These DLLs translate a user level function into a native system call.

Windows executive implements the services in the kernel, such as memory management, process and thread management, security, I/O, networking and inter process connection [7].

Kernel implements low level operating system services, such as thread scheduling, interrupt and exception dispatching and multiprocessor synchronization. Kernel also provides objects which will be used by executive.

Device drivers translate user or device defined I/O functions into related I/O request packets which the kernel can understand.

Hardware Abstraction Layer (HAL) is an interface between kernel, device drivers and hardware. HAL isolates the kernel, device drivers and executive from different kinds of hardware. It provides functionality for kernel, device drivers and executive to implement their works independent from the hardware [7].

I/O system in Windows includes several executive components to manage hardware and to provide interaction between hardware and applications. These components are I/O Manager, Plug and Play Manager and Power Manager.

I/O Manager is the main actor in the I/O system. It provides communication between applications, systems and devices.

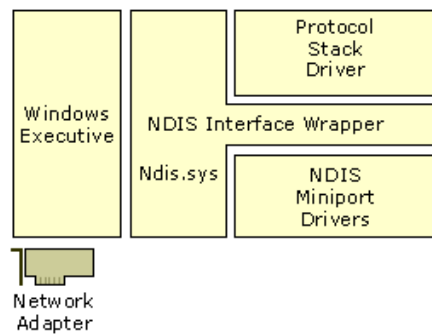
A device driver is a software module that translates high level commands, such as read and write into device understandable commands. They take the commands from the I/O manager and transmits the related device. Once the commands are done, device driver informs the I/O manager.

A user application can issue one or multiple I/O requests. If the request was synchronous I/O the request would reach the device and the device will implement it. After completing the work, the device will inform the application. After informing, the application can continue its work. If the I/O request type is asynchronous the application doesn't have to wait the device to continue its work [7].

## **2.6 Network Driver Interface Specification (NDIS)**

NDIS is the system that provides the communication between network card driver and protocol drivers. It provides the capability for protocol drivers to send network packets to the network drivers and receive network packets from network drivers independent from the model the network adapter and operating system.

NDIS itself is also a driver and Ndis.sys file is located under the Windows folder. Actually this driver is a wrapper and provides protocol drivers to communicate with NIC driver. NIC drivers should be written for Windows O.S. as NDIS compatible so that Ndis.sys can understand NIC driver language. NDIS converts the NIC language to the language that protocol drivers can understand.



**Figure 2.6 : NDIS Wrapper [9].**

NDIS support three kinds of network drivers:

Network Interface Card Driver: NDIS can manage NICs directly. NIC driver is the interface between NIC and upper layers. Its jobs:

- a. To send packets to the network,
- b. To handle with interrupts,
- c. To reset the NIC,
- d. To halt the NIC.

NIC Drives can be miniport or full NIC. Miniport drivers can only send and receive the packets. NDIS does the low level hardware operations. Such as synchronization for miniport drivers, establishing connection with OS. But Full NIC can do these works by itself.

Interface Drivers are the second one. They are placed between NIC drivers and protocol drivers. The reason using these drivers is to translate some kinds of media types that NIC doesn't understand to the proper types.

Protocol drivers are the last ones. These drivers execute the protocol works. Such as; TCP/IP protocol driver.

The version of NDIS changes according to version of Windows O.S:

|       |            |
|-------|------------|
| Win95 | : NDIS 3.0 |
| Win98 | : NDIS 5.0 |
| WinXP | : NDIS 5.1 |

Win7 : NDIS 6.20  
Win8 : NDIS 6.30  
Win10 : NDIS 6.50 [9].

Netgroup Packet Filter in WinPcap is also a protocol driver. NPF is not working as a synchronic drive. When an operation is need by applications in the user level they are activating the NPF and also when NIC catches a packet it is activating the NPF.

## **2.7 The Path from NIC to Applications**

NICs have a limited size of memory (a few Kbytes) inside it and this memory should deal with sending and receiving the network packets [10]. Also NICs check the packets when they are still inside the NIC memory in order to discard the improper packets, such as short Ethernet frames.

When the packet is valid, the NIC requests a bus controller role to transfer the packet from its memory to the main memory of the operating system. After transferring the packet, the NIC creates a hardware interrupt which will trigger the NIC driver.

NIC driver creates a Deferred Procedure Call (DPC) to inform upper layer drivers about that a new packet has been received. If there is a network capturing driver in the system, it will receive the packet after filtering process to deliver to the applications in the user level [10].

## **2.8 Journey of Network Packet in Internet**

A bus network is connecting of each computer or network device to a single cable. Internet is also a bus network which consists of a control node and many nodes all around the world. Each node in the internet has a unique address. When a packet is sent by any node, it will be sent all nodes due to the broadcasting logic. The Network Card for each host examines whether source address of packet is matching with its address or not. If not, the packet will be discarded by the host. On the other hand, if it matches, network card will deliver the packet to the operating system.

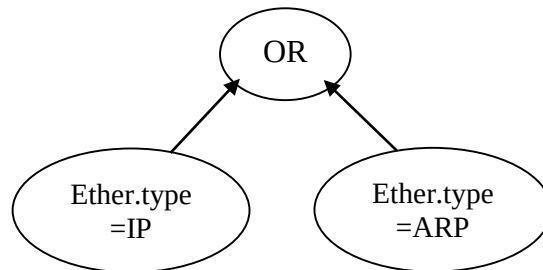
Nowadays, network cards have four modes: the broadcast, the multicast, direct and promiscuous modes. Direct and promiscuous modes are the basic modes for network cards. If the network card switches the promiscuous mode, it will process all packets in the network to the operating system [11].

### 3. NETWORK PACKET CAPTURING SYSTEMS

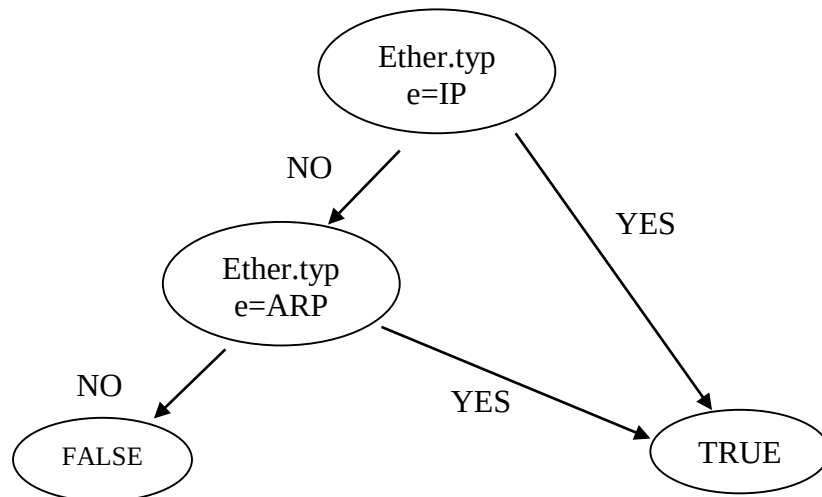
#### 3.1 Berkeley Packet Filter

For a network analysing system good buffering can be measured with the number of packets that are dropped before processing the user space while good filtering can be measured with the number of the packets that are processed to the buffer in accordance with the related user defined filter. A good performance for capturing applications is possible with having a well designed filtering machine.

There are two methods to create a filtering machine. First one is a Boolean expression tree and second one is a directed acyclic control flow, which is used in Berkeley Packet Filter (BPF). Examples for methods can be seen at Figure 3.1 and 3.2.



**Figure 3.1 : Tree Model**



**Figure 3.2 : Acyclic Control Flow (ACF) Method**

Tree Model has some disadvantages for capturing process:

- It produces more memory traffic.

- It computes the formulas even if it is not necessary. For example; it should compute ether. Type=IP even if it doesn't need to compare them.

BPF uses acyclic control flow method. ACF doesn't need to compute all packets for each comparison.

Most UNIX versions provide user level functionality to capture network packets from the network traffic. Every time a user captures a packet and then it should be copied from the kernel to the user level. This causes a bottleneck for operating system because; every time creating a system call to copy packets can cost a huge usage of CPU.

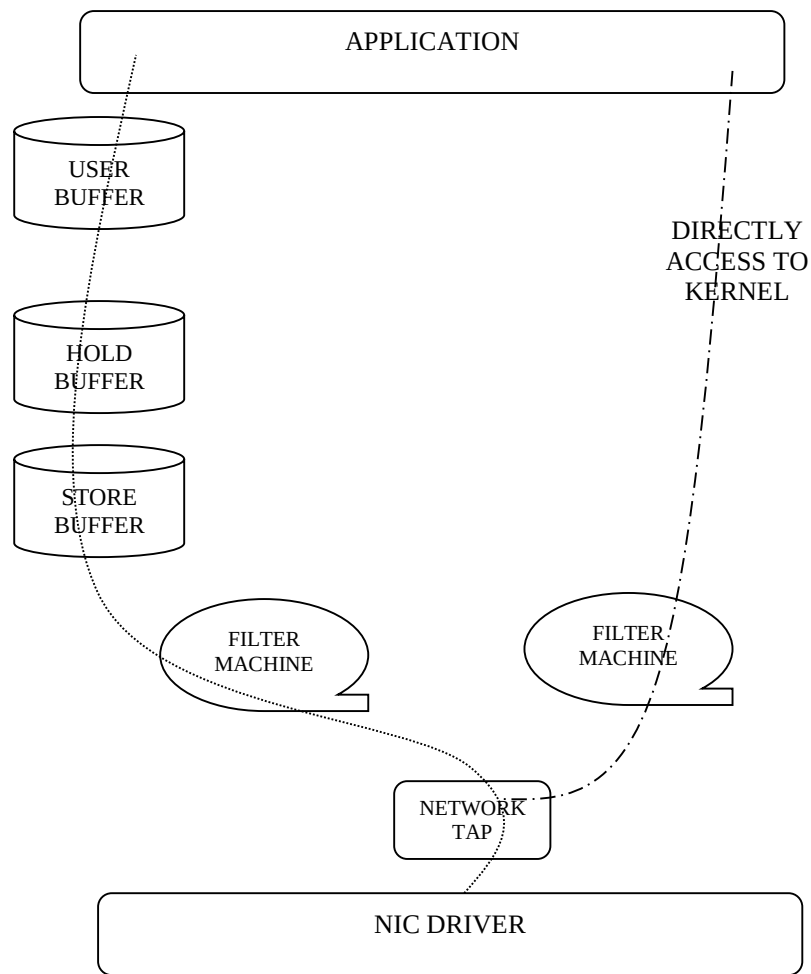
Berkeley Packet Filter is a kernel agent located inside the kernel. Its main function is to set a filter inside the kernel and decrease the numbers of the packets that will be copied to the user level. This kernel agent provides 20 times faster packet capturing than the normal packet capturing processes [12].

The first idea to reside a filter inside the kernel came from a study in CMU/Stanford. They placed a packet filter in a UNIX kernel in 1980[12]. This filter was sufficient for the time being computers but not the computers that we are using today.

Normal traffic in a UNIX environment when link level driver gets a packet it sends it to the protocol stack. But if there is a BPF which tracks the communication, link level driver delivers packet to the BPF first. When the BPF gets the packet it copies the packet to the kernel buffer if the packet is valid for user defined filter.

There are two buffers inside kernel: Store and Hold buffers. Their size is 32 Kbytes and this memory for them is allocated at the beginning of the capturing process [12]. Store buffer keeps the packets coming from filter machine while Hold buffer keeps the packets that will go to the user level buffer.





**Figure 3.3 : BPF Architecture**

### 3.2 Network Packet Capturing for UNIX Systems

When the first network was established over ARPANET in 1969, the traffic was low and it was easy to monitor this traffic. Today the situation is a little bit challenging. Due to the large computer networks the network traffic is huge and the need for a new diagnosis tool is occurred. Once a network card receives a network packet it checks its destination MAC address. If the MAC address matches with its own MAC address, network card copies the packet to the system memory in the kernel space and then it checks its Ethernet header. According to header packet is copied to the related protocol stack. When IP stack receives a packet it conducts some tests to clarify whether the packet changed on the way or the destination address for the packet is its address. After tests protocol header are removed and packet comes to transport layer. This process continues until the packet reaches to the application layer.

Libpcap is an open source library providing functionality to capture network packets over the network. McCane, Leres and Jacobson created the library in 1993 to improve a platform independent API to capture network packets. The library first was created by C programming language. But today there are some wrappers for other programming languages to use the library.

Functions in library is:

Pcap\_lookupdev is to show the network devices name

Pcap\_open\_live is to open the selected interface, which includes maximum bytes of packets to capture and time for copying from kernel to user level.

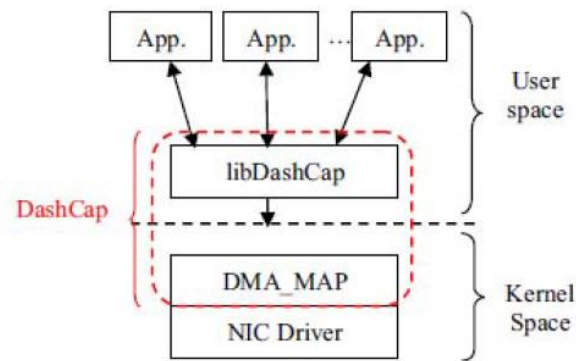
Pcap\_next is to capture packets in a loop.

Libpcap library captures the packets in a loop and copies them to the user application. The packets consist of Ethernet header, IP header, Protocol header and data. It is a work for programmers to design raw network packets for their own purposes.

Different kind of new solutions proposed their own packet capturing systems for UNIX Systems by using libpcap library. They also used new technologies such as circular buffer to increase the performance of their network monitoring tools. DashCap, nCap and Beyond Device Polling are among the existing solutions.

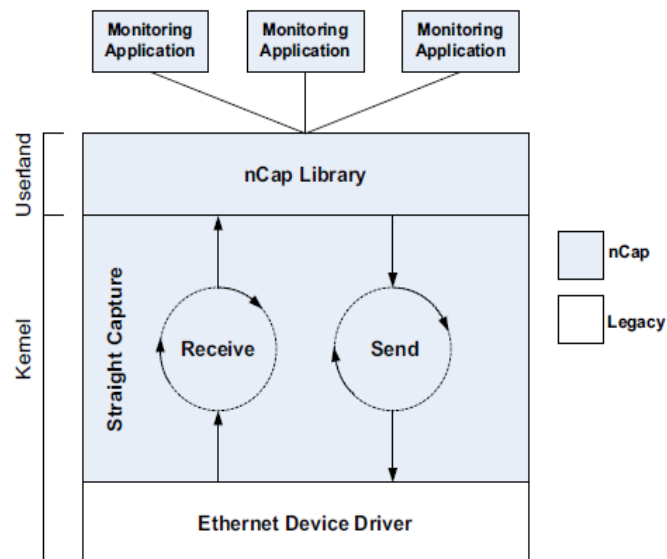
In high speed networks, there is huge possibility losing packets while sending from NIC to kernel then to the user level. DashCap proposal was to include two component into system to increase the performance. These components are DMA\_MAP in the kernel and libDashCap in the user level. Both components provided functionality to increase the performance to capture packets in high traffic. DMA\_MAP is a kernel and NIC driver dependent component.

DashCap also uses circular buffer technology. Their system consists of two different buffer, receive (Rx) and transmit (Tx) buffer. While Rx buffer gets the packets from DMA\_MAP, Tx buffer sends the packets into the address space inside memory with calling mmap function in UNIX. With this functionality, user level applications can reach packets from memory without any other system call. DashCap is able to capture high packets rates without losing any packets up to 700 KPPS( Kilo Packets per Seconds) [13].



**Figure 3.4 :** DashCap Architecture [14].

nCap has been designed to provide functionality for capturing packets with wire speed and sending packets at least 1 Gbit speed. Also, it has been designed to give more control over system components for users. It has two circular buffers inside to kernel where incoming and outgoing packets are placed. Applications in user level have authorization to customize the buffers directly from user level [13].



**Figure 3.5 :** nCap Architecture [14].

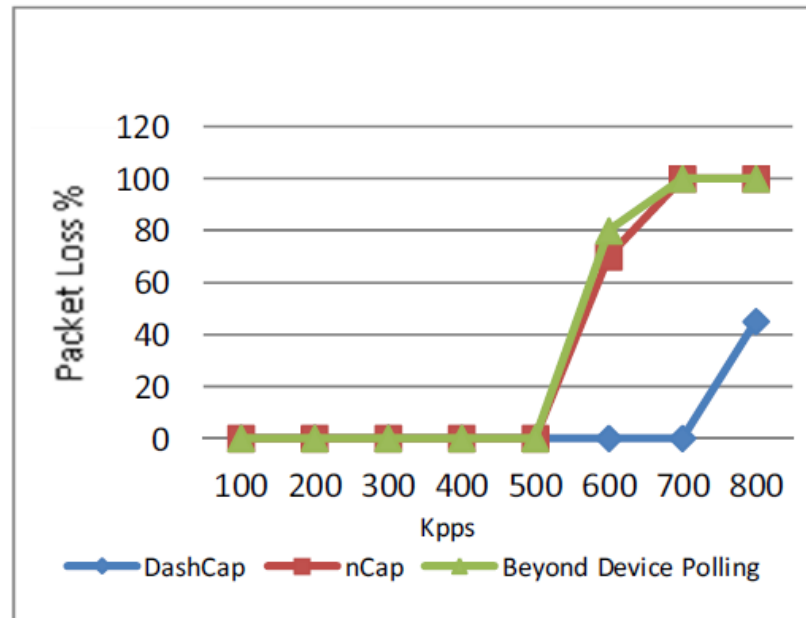
Beyond Device Polling has been proposed to deal with the overhead of operating system in high speed networks. The Device polling was implemented where polling is a technique for handling the devices including network cards to perform the tasks for packet capturing. In the design the packets are not queued inside the kernel. Users applications can capture packets directly from NIC by socket system calls. Due to the the using socket system calls there is no overhead in the system [13].

**Table 3.1:**Comparison of libraries for Unix Operating System.

| Existing Solution            | Architecture                                                                                                                                                                                                                                                                                                                                          |                                                                                                                                                                       |
|------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                              | Kernel Space                                                                                                                                                                                                                                                                                                                                          | User Space                                                                                                                                                            |
| <b>DashCap</b>               | DMA_MAP can Access the NIC driver by registering in /dev file system, opening this device through it and then calling its system calls.                                                                                                                                                                                                               | libDashCap is implemented in user space for its simplicity of implementation and error handling.                                                                      |
| <b>nCap</b>                  | Device driver is responsible for controlling the Ethernet device and creating two circular buffers where incoming and outgoing packets are set as well as enable applications to act upon the two buffers and their indexes directly from user-space.                                                                                                 | nCap library allows the applications to control the two buffers and their indexes directly from userspace by means of memory mapping without any kernel intervention. |
| <b>Beyond Device Polling</b> | Device polling is implemented where polling is a technique for handling devices including network cards that perform the tasks:<br>-When network device receives a packet it creates an interrupt to request kernel attention.<br>It is a good solution to enhance both packet capture performance and system responsiveness under high traffic load. | Libpcap-mmap use the mmap() system call for passing packets to user space, lessen the time spent moving the packet from the kernel to user-space.                     |

From the performance point of view, DashCap provides the best performance regarding packet loss in high speed networks. Beyond Device Polling and nCap can tolerate packets up to 570 KPSS. After reaching maximum level, their packet loss rate is increasing 100%. DashCap can tolerate packets up to 770 KPSS [13].

Another solution proposed to decrease the packet loss rate for high speed networks is to use link lists in the applications. Link lists can decrease the usage of memory so that overhead may be less. Link lists provides increasing and decreasing the size of list without fixing the size of memory. This means that any packet insertion and deletion can be handled efficiently without fixing the size of the memory in advance.



**Figure 3.6 :** Packet Loss Comparison [13].

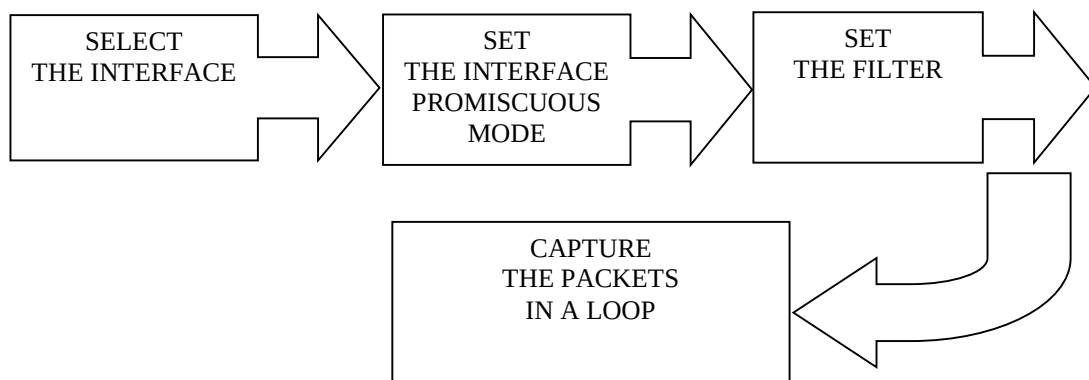
### 3.3 Network Packet Capturing in Android Operating Systems

The more a system is used the more it is being attacked by intruders. One of the popular systems nowadays is Android Operating System. According to the reports Android has 9.2% portion in the spreading malicious traffic and also attackers can reach personal information from the Android devices easily [15]. From this observation we can understand that providing a secure environment for Android systems is one of the primary tasks for mobile security developers.

Android OS is created by Google Company and its framework system is Linux also it is written with Java Programming language.

Internal of the Android includes three layers. At the top it has an application program which provides interface for the users. At the bottom there is Linux kernel. Between Linux kernel and application program there is application framework which provides communication between top and bottom.

The network traffic in Android OS is not huge but the processing capability of the system is not good enough. So, packet capturing interface for Android should have very good performance. LibPcap library which is used on Linux systems to capture network packets can be used also on Android systems for the same purpose with a little bit exceptions. With the help of LibPcap library network packets can be captured and saved in a memory card. Basic work flow for packet capturing with the library can be seen at Fig.3.2.



**Figure 3.7 : LibPcap Work Flow**

Because of the security issues Android System doesn't let to interact with Linux Kernel directly from application program. LibPcap library was written with C++ programming language while Android System is Java Based system. But interacting with the kernel this C++ library should be used. This is a problem for developers and to deal with that issue Java Native Interface in Android System can be used. JNI is a Java Developer Kit (JDK) providing interface for other programming languages with Java [15].

### **3.4 WinPcap Windows Packet Capturing Library**

UNIX System lets the users to interact with the network directly with the help of some system calls to capture network packets. But Windows operating systems protect low level components from the high level applications. There is no significant system calls like UNIX systems to provide interaction with the network directly. If an interaction with the low level is needed a driver should be included inside the Windows system.

There are some drivers providing interaction with the network for Windows system. But they all have some limitations. NETMON API is only working on Windows 2000 operating system, it is not open source and it doesn't have filtering and raw packet sending functionality. PCAUSA was produced to provide UNIX compatible packet capturing driver for Windows. Its user interface is not well designed and filtering functionality is not user friendly [16].

WinPcap packet capturing platform is the first open source library with the several functionalities inside it. Therefore, most popular network analysis applications use WinPcap library at the bottom of their applications. The designers of WinPcap are Fulvio Rises and Loris Degiooni who are Italians.

Basically WinPcap was designed on BPF structure to provide UNIX compatibility to the applications. A detailed structure of the WinPcap can be seen at Fig.3.3.

In Windows systems WinPcap can be used with:

1. Network and Protocol Analyzers
2. Network Monitoring systems
3. Network Logger Systems
4. Traffic Generator
5. Network IDS
6. Network Scanner
7. Security Tools

Filtering is very important for packet capturing performance. Most of the applications generally want to catch a specific packet.

WinPcap filtering starts from the user defined filter. In user level the filter that user defined is converted to the pseudo instruction.

Example: If the packet is IP and the protocol type is equal to 18

Then

Return true

These instructions are sent to the filtering machine inside the kernel.

The differences of WinPcap are hidden inside the kernel module. WinPcap architecture has a circular buffer inside the kernel that can copy a block of packets at one time. Implementing a circular buffer is harder than implementing a hold and store buffer as in BPF. There is no fix size buffer in circular buffer system. According to captured packets the buffer size can be raised and when the user buffer is free packets inside kernel buffer will be sent to user buffer. Then the kernel buffer size will get decreased. This provides more right for memory usage and more speed than BPF system.

Choosing a buffer size in user space is very crucial for network packet capturing systems. Having a small or large user buffer can change according to the developers' will. When the user buffer was chosen larger, the kernel buffer would wait until having enough packets to fill the user buffer. It means that the system calls from user to kernel will be less. It is a good scenario to use processor less. But if the user buffer is kept smaller the kernel buffer will send the packets as soon as possible. This is also good for real time capturing systems. WinPcap has option to configure user buffer size according to needs.

Copy time process is the number of hops for network packets. The more copy number the more system will have overheating due to the memory usage. WinPcap has 2 copy time processes because it copies the packets first from network driver to kernel buffer then from kernel buffer to user buffer. Filtering is the first phase in the WinPcap structure so that unnecessary packets will not cause the memory usage and overheating will be less.

Real time packet capturing is to capture network traffic at the same time it happens. Packet capturing applications use huge amount of CPU when they are in loop. If the application is not functional enough lose of the network packets can occur. One of the methods to decrease packet losing is to use filters while the packets are still inside the NIC driver. Another method is to keep the number of packets which will be copied to memory less. In each copy function application or the kernel uses system calls that cause the usage of CPU.

WinPcap has a statistical functionality to make the usage of CPU less. In this function statistical information is kept inside the kernel such as amount of data per second. With only one system call the application can get the result from the statistical machine.



WinPcap consists of three different components. One of them works inside the kernel as a protocol driver to communicate with the NDIS. NPF.sys file can be found under the System32/drivers folder after the installation of WinPcap which is the kernel driver. Main purpose of the driver is to get packets from NIC and deliver them to the user level.

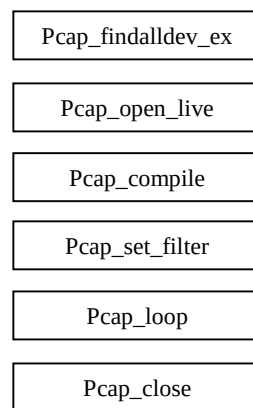
Second component which is located in user level is packet.dll component. It is an API for Windows systems and its duty is to be an interface between user and kernel level. Some low level functions are performed by packet.dll such as getting the name of interface or the net mask of the interface etc. Because; each Windows version has different kind of NDIS version up of the NIC driver, NPF driver and packet.dll should be configured according to the NDIS. These components work OS dependent.

The last component in WinPcap is Wpcap.dll which is a high level component and is not OS dependent. The component provides high level functionalities such as setting the size of user buffer or producing a filter etc. The component can communicate directly with the protocol driver to execute the capturing process.

A faster capturing process can be measured according to the performance of the device driver inside the kernel. Once the NIC driver gets the packet it will process the packet to the upper layer. The device driver should capture the packets before this process. In UNIX systems a special system call can be created to direct the NIC driver to send a copy of the packets to the capturing driver but in Windows systems it is not possible to modify the operating system or NIC driver. The solution for this problem is network tap. Network Tap is a protocol driver created by WinPcap at the top of NDIS to get one copy from each packet. Tests are implemented to show the performance of WinPcap including the comparison with UNIX BPF system in [16]. Tests show that the usage of the CPU never reaches 100% level with WinPcap. While BPF can only capture half of the packets with high usage of CPU, WinPcap can capture all packets on the network traffic. When the evaluating of the dumping machine which saves captured packets to the disk directly is done it is a bottleneck for both UNIX and Windows systems. Some packets were dropped because of the high CPU usage or the lack of kernel buffer space. Another observation is the performance of the statistical machine which can monitor and bring statistical information about packets inside the kernel. Observation was that making monitoring inside the kernel costs less CPU usage whose reason is the less switching between user and kernel level [16].

Kernel buffer size is also important piece for capturing performance. In Windows systems increasing the kernel buffer size can decrease the number of packets that are dropped but in UNIX systems the performance is not changing after changing the kernel buffer size.

The basic work flow of Winpcap:

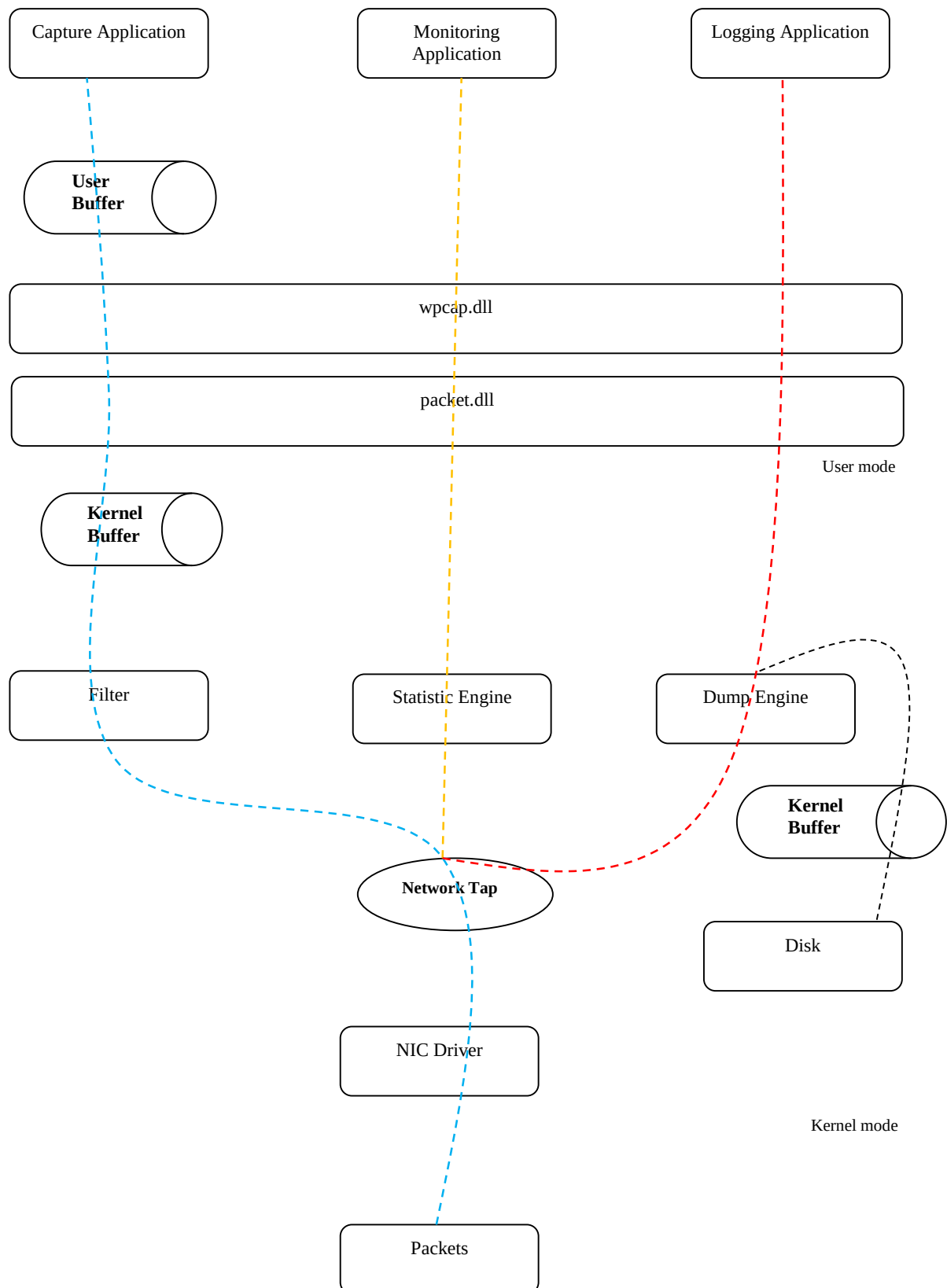


**Figure 3.8 : WinPcap Work Flow**

To list the network adapters, WinPcap has “pcap\_lookupdev” function.

To open a connection it has a “pcap\_open\_live” function. This function uses five inputs. Network Adapter, Buffer Size, Promiscuous Mode, 0, Error Buffer.

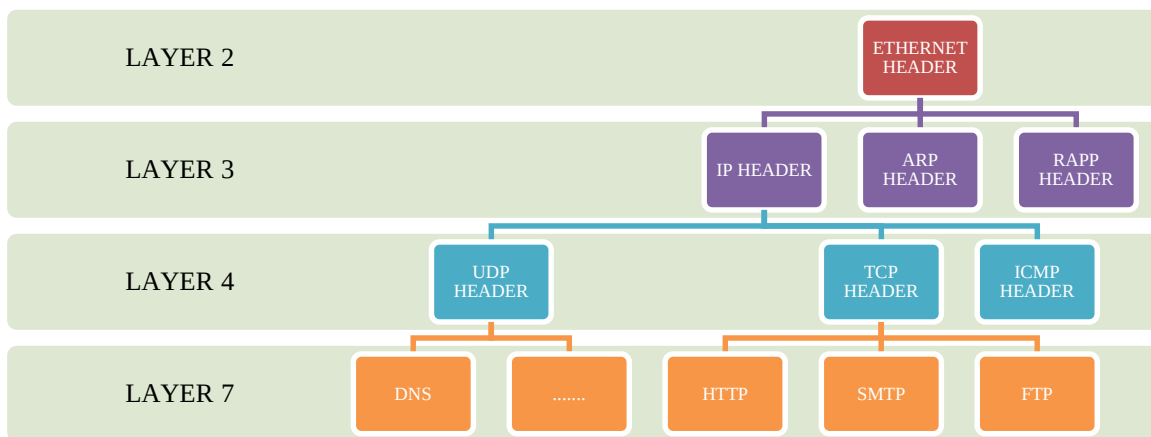
To filter the traffic, there are two functions. “pcap\_compile” and “pcap\_filter”. First one is to convert a filter string to the program understandable format and second one is to set the filter.



**Figure 3.9 : Netgroup Packet Filter**

### 3.5 Packet Encapsulation

Packets encapsulating is to add a header to packets in each network layer. It starts from Application layer until the Physical Layer. In order to analysis the packets a network analyzing tool should examine all headers. Filtering machine inside the kernel can clarify the kind of packets such as only tcp packets or ip packets will be captured. According to filter type packets arrives the user level and in user level the application according to its design can separate the headers from the packets. Packets are delivered by a stream that includes Ethernet, IP and etc. headers inside it. In the application showed in appendix-1 designed to capture destination addresses inside the udp and tcp packets. To separate this result from all stream after first 14 bytes which is Ethernet header 20 bytes of the stream were taken and the destination address from the IP header was delivered to the database.



**Figure 3.10 : Encapsulation**

### 3.6 Costs and Optimization of WinPcap

Filtering is the one of the main component for network capturing applications. Therefore, the performance of the filtering can affect the whole capturing process.

There is a relation between the number of clock cycles and the number of filtering instruction. The more building a complex instruction for filtering the more the CPU will be used by the application. For example; the instruction for capturing only IP packets can create 131 clock cycles while the instruction for capturing UDP Port 20 packets can create 585 cycles [4].

The second cost in design of WinPcap is copy time of network packets. The packets are copied twice in WinPcap. The first one is from NIC memory to the main memory which is performed by `NdisTransferData()` function. This function has two disadvantages. Firstly, the function can copy the packet from NIC to main memory if the whole packet exists in the NIC memory. If not, the function will cause a delay. Second disadvantage is when the function is performed, The NIC will need a bus master role to transfer the packet. This process also causes more clock cycle number.

The second copy function for copy from kernel buffer to user buffer also can increase the number of clock cycles. The number of clock cycles can vary according to the length of the packet and the size of the kernel buffer.

The number of clock cycles varies between 540 and 10500 for first copy function while the number of clock cycles varies between 259 and 8550 for second copy function [4].

One of the packet filtering costs is the number of clock cycles to obtain the timestamp of each packet. WinPcap is using `KeQueryPerformanceCounter` function that is the only function to obtain a time reference with microsecond level. This function uses huge amount of clock cycles due to its interaction with system time chip, which costs 1800 clock cycles per packet. 1800 clock cycles per packet is the maximum number of clock cycle usage in the packet capturing process.

Experiments showed that the optimization of filtering machine is possible with using a just in time machine to translate a filtering instruction to the 80x86 binary code. 8% improvement was observed with a complex filtering instruction and after WinPcap version 3.0, this property was implemented in the kernel [4].

It was observed that the first copy function which was from NIC memory to main memory was using more CPU than the second copy function. Optimization of this function is possible with using a C library function to copy the packets piece by piece to main memory without waiting the completing the packet. This optimization decreased the number of clock cycles from 540 to 300 for 20 byte packets [4].

Optimization of timestamp related to the cost for packet capturing system is possible with the usage of TimeStamp Counter whose function is rdtsc for Windows operating system. Using this function decreases the number of clock cycles from 1800 to 270 per packet. This provides a good performance for system but the problem is that only Intel CPUs have this function inside them. Therefore, WinPcap Team disabled this optimization as default for their system [4].

## **4. EXPERIMENTAL RESULTS AND RISK ANALYSIS**

### **4.1 Experimental Results**

In this chapter, the comparison of packet.dll with wpcap.dll and the results of observations were presented.

As it is presented in the third chapter, packet.dll and wpcap.dll have different functionalities. Packet.dll is providing interaction with kernel driver, whereas wpcap.dll is conducting high level functions such as filtering. A capturing system which is able to capture all network packets can be designed by both using only packet.dll or using only wpcap.dll.

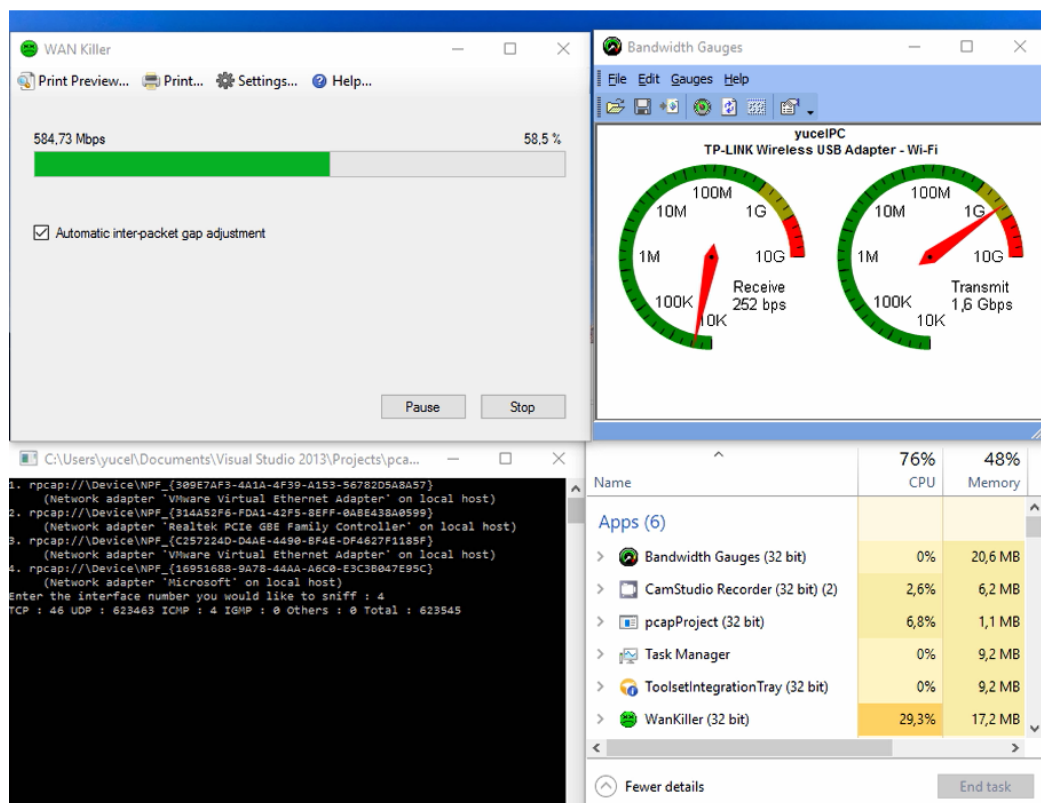
In a close environment, a virtual network traffic was created gradually. The traffic load was 100 MB first, then it was 1 GB and 10 GB network load was created lastly.

The main idea of this experiment was to observe the usage of microprocessor and memory by network packet capturing system and to compare the two winpcap dll files.

The test environment was designed carefully so that there will not be any other network traffic in the environment. Internet connection was closed and there was no modem connection which may cause occurring of some network packets.

Two different gauges which were showing the load of network traffic were used in the experiment. One gauge was showing outgoing traffic load, whereas other was showing incoming traffic.

Observation of microprocessor and memory usage was done by using Windows Task Manager. The system properties of host ,which the implementation was done, were Intel Core i5 CPU (Central Processor Unit), 4.00 GB Memory, 64 Bit Windows Operating System.

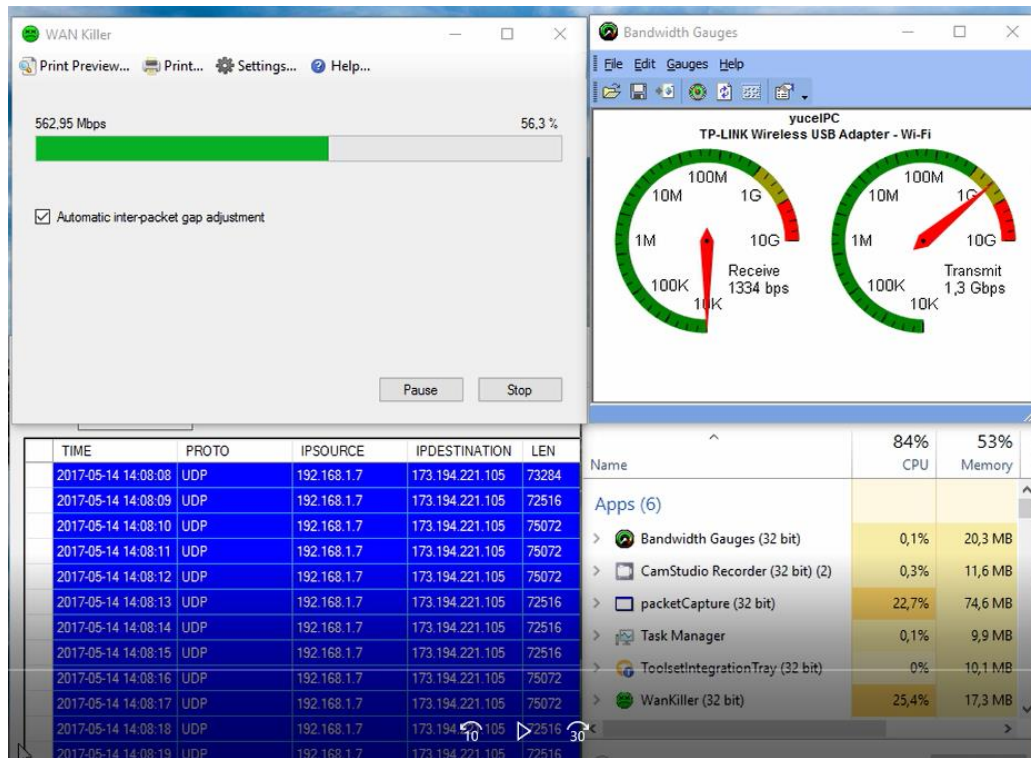


**Figure 4.1** : Network Packet Capturing by packet.dll

The simulation of network packet capturing by packet.dll can be seen at Figure 4.1. Under 100 MB network traffic load, the application was using 4.2% of CPU and 0.8 MB memory. When the load was 1 GB, the application was using 6.8% of CPU and 1.1 MB memory. Once the load was 10 GB, the application was using 7.1% of CPU and 1.5 MB memory.

It was clearly seen from the experiment that the application which had been created by using only packet.dll was not using too much CPU and memory. We can say that packet.dll doesn't cause the fail over of operating system by using too much CPU and memory.





**Figure 4.2 : Network Packet Capturing by wpcap.dll**

The simulation of network packet capturing by wpcap.dll can be seen at Figure 4.2. Under 100 MB network traffic load, the application was using 17.4% of CPU and 52.6 MB memory. When the load was 1 GB, the application was using 22.7% of CPU and 74.6 MB memory. Once the load was 10 GB, the application was using 25.8% of CPU and 94.2 MB memory.

It was clearly seen from the experiment that the application which had been created by using only wpcap.dll was using more CPU power and memory than packet.dll application. We can say that users who want to capture all network packets without filtering should use only packet.dll executable file in order to use less operating system sources.

## **4.2 Risk Analysis**

It is better to analyze the risks of all kind of network tools in order to understand the gaps of the systems. Gaps are the starter point of future works. Therefore, the points of network packet capturing system which are vulnerable to attacks are presented in this session.

The first attack point can be Network Interface Card because; the process is getting started in NIC. Once a new network packet arrives the NIC, NIC will inform the NIC Driver. If NIC can't inform NIC Driver, this new network packet will not be captured by packet capturing system.

The second attack point can be Network Driver Interspecification which is the intermedia driver inside the operating system. This driver is providing communication between NIC and capturing driver.

The third attack point is the network capturing system driver. In network packet capturing system, driver inside the kernel is conducting critical works.

Command and Controlling of Network Packet Capturing is possible with reaching device drivers and control them how we want.

Multipartite viruses are the viruses which can infect from boot sector of operating system to the file system and even device drivers [17]. Multipartite viruses can be used to control NIC driver or NDIS or Packet Capturing Driver.

## 5. CONCLUSIONS AND RECOMMENDATIONS

When the comparison is done it is seen that WinPcap library is the most powerful packet capturing system. It has different kind of functionality to work under high speed network traffic.

As we saw previous sections WinPcap has two dll files inside user level which are packet.dll and wpcap.dll. packet.dll includes simple functions while wpcap.dll has high level functions and more functionality. When we compare two dll with a simple packet capturing application we can see that wpcap.dll is using more CPU than packet.dll. The main reason for additional CPU usage for wpcap.dll is to use basic functions such as listing interface names and establishing connection with kernel driver wpcap.dll needs packet.dll.

Another observation was done to show the CPU usage of libPcap and WinPcap libraries. As explained before libPcap library can use 100% CPU sometimes and it costs dropping some network packets but with WinPcap CPU usage doesn't reach 100% anytime.

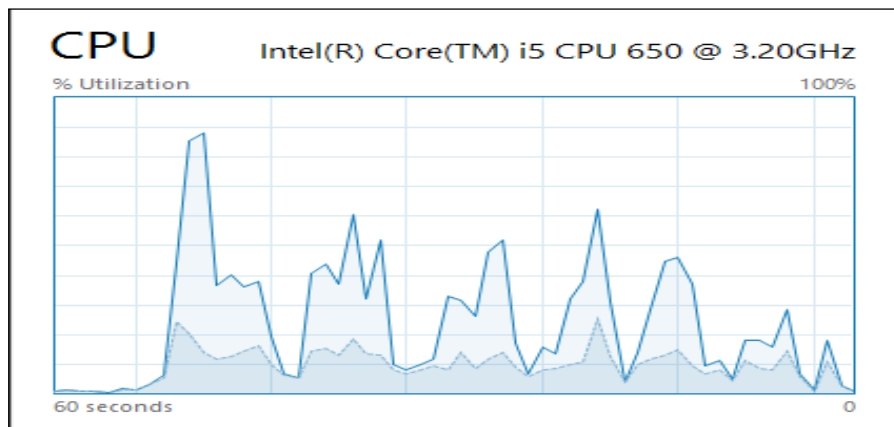
In the experiment several different web sites opened while the capturing programs were open and the usage of CPU was observed. In UNIX environment Wireshark application was used while Windows environment packetCapture application was used. As showed at Figure 5.1 and 5.2 in UNIX CPU usage was 100% two times while in Windows 100% usage was not observed.

### 5.1 Practical Application of This Study

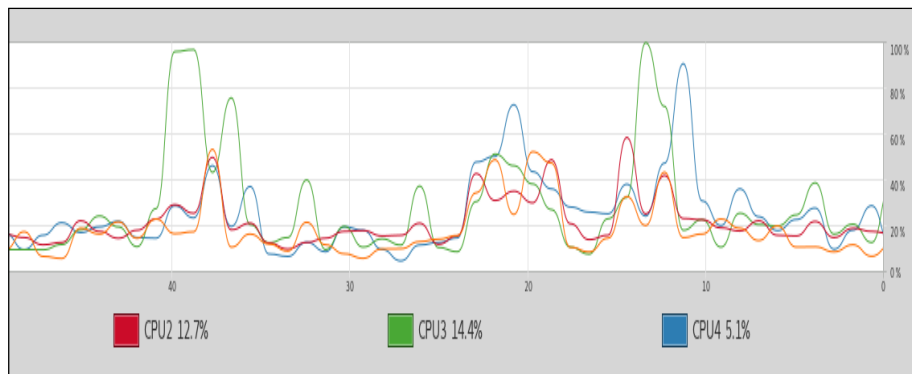
packetCapture application is a basic tool to capture network packets. It has functionalities to list network devices, to capture network packets in the chosen driver, to deliver destination IP addresses and to create a white list showing the IP addresses that will not be captured by the application. A capturing sample can be found at the Figure A.1.

Once find devices button clicks application shows the list of the interface in the host with the definition of interfaces. After choosing the interface wanted to be monitored one thread is executing the capturing the packets. Before saving the packet information to the database, the thread is checking whether the packet information is in the white list or not.

Two threads were used for the application. Second thread is delivering the packet informations from the database and saving the white list information to the other database while first thread continues the capturing phase.



**Figure 5.1 : Packet Capturing with WinPcap**



**Figure 5.2 : Packet Capturing with LibPcap**

## 5.2 Future Works

The basic understanding of network packet capturing systems were presented in this study. This study can be used as a starter point for several future works.

One of the open research area is to design more efficient network intrusion system. Network Intrusion Systems are designed to separate the malicious network traffic and normal network traffic. packetCapture application is creating a SQLite database with using performance of WinPcap library. Some statistical information is saved in the database such as country information, application port etc. These statistical information can be used to create a normal user behavior.

Another open research area is to improve the performance of the network packet capturing systems for open source operating systems. It is clear that in the future most of the operating systems will be open source and lightweight operating systems. For these systems it will be required to build new intrusion detection systems or firewall systems. Therefore, new lightweight packet capturing systems will be crucial.

The third open research area can be the security of device drivers. As it was seen in the risk analysis, network packet capturing systems are vulnerable to attacks due to the multipartite viruses. The prevention methods can be improved against the multipartite viruses.



## REFERENCES

- [1] **Alias, S. B., Manickam, S., & Kadhum, M. M.** (2013, December). A Study on Packet Capture Mechanisms in Real Time Network Traffic. In *Advanced Computer Science Applications and Technologies (ACSAT), 2013 International Conference on* (pp. 456-460). IEEE.
- [2] **Pande, B., Gupta, D., Sanghi, D., & Jain, S. K.** (2005, July). The Network Monitoring Tool—PickPacket. In *Information Technology and Applications, 2005. ICITA 2005. Third International Conference on* (Vol. 2, pp. 191-196). IEEE.
- [3] **Shamsi, J., & Brocmeyer, M.** (n.d.). Principles of Network Monitoring.
- [4] **Degioanni, L., Baldi, M., Risso, F., Varenni, G.** (2012, May). Profiling and Optimization of Software Based Network Analysis Applications.
- [5] **Url-1**<<http://msdn.microsoft.com/tr-tr/library/Windows/hardware/ff554836>>, date retrieved 17.08.2016.
- [6] **Url-2**<<https://docs.microsoft.com/en-us/windows/hardware/drivers/debugger/user-space-and-system-space>>, date retrieved 17.08.2016.
- [7] **Russinovich, M. E., Solomon, D. A., & Allchin, J.** (2005). Microsoft Windows Internals: Microsoft Windows Server 2003, Windows XP, and Windows 2000 (Vol. 4). Redmond: Microsoft Press.
- [8] **Url-3**<<http://msdn.microsoft.com/tr-tr/library/Windows/hardware/hh439643>>, date retrieved 02.09.2016.
- [9] **Url-4**<<https://technet.microsoft.com/en-us/library/cc958797.aspx>>, date retrieved 02.09.2016.
- [10] **Degioanni, L., Baldi, M., Risso, F., Varenni, G.** (2012, May). Profiling and Optimization of Software Based Network Analysis Applications
- [11] **Xiaoguang, A., & Xiaofan, L.** (2016, August). Packet Capture and Protocol Analysis Based on Winpcap. In *Robots & Intelligent System (ICRIS), 2016 International Conference on* (pp. 272-275). IEEE.
- [12] **McCanne, S., & Jacobson, V.** (1993, January). The BSD Packet Filter: A New Architecture for User-level Packet Capture. In *USENIX winter* (Vol. 93).
- [13] **Alias, S. B., Manickam, S., & Kadhum, M. M.** (2013, December). A Study on Packet Capture Mechanisms in Real Time Network Traffic. In *Advanced Computer Science Applications and Technologies (ACSAT), 2013 International Conference on* (pp. 456-460). IEEE.
- [14] **Dashtbozorgi, M., & Azgomi, M. A.** (2009, August). A high-performance software solution for packet capture and transmission. In *Computer Science and Information Technology, 2009. ICCSIT 2009. 2nd IEEE International Conference on* (pp. 407-411). IEEE.

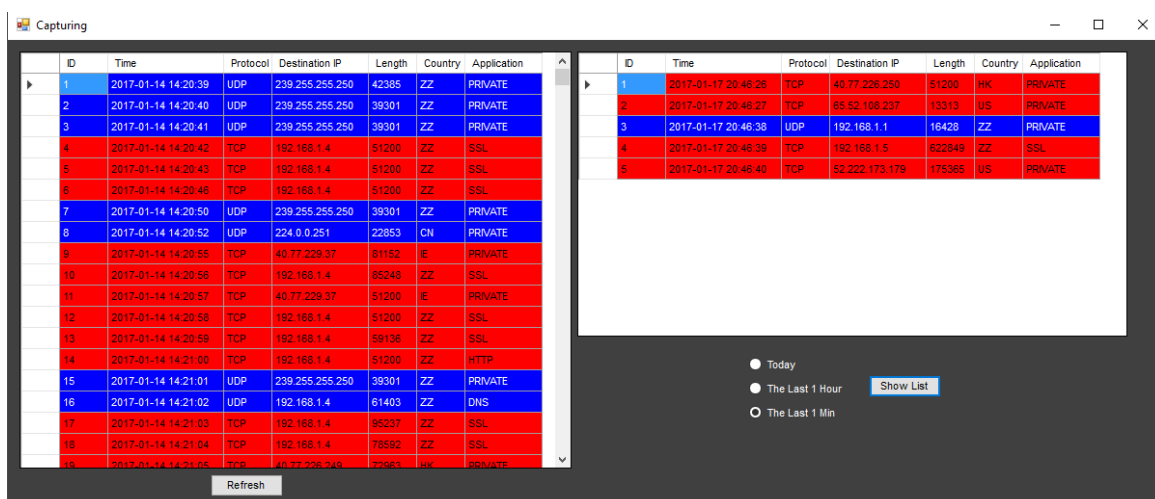
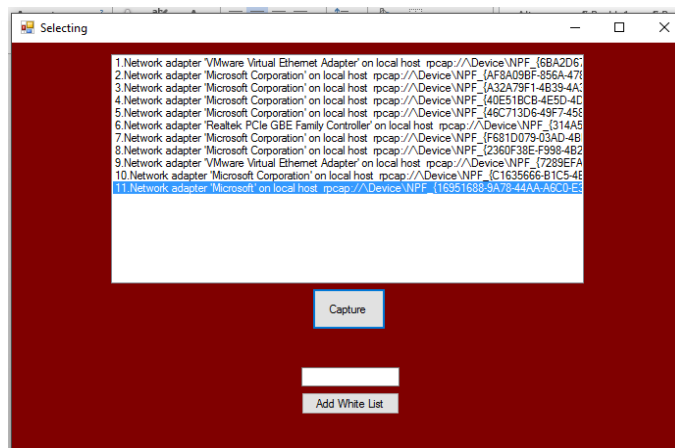
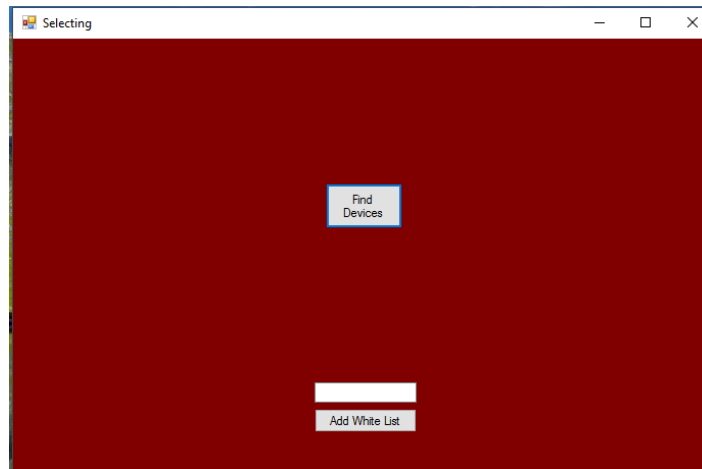
- [15] **Cheng, K., & Cui, Y.** (2012, May). Design and implementation of network packets collection tools based on the android platform. In Fuzzy Systems and Knowledge Discovery (FSKD), 2012 9th International Conference on (pp. 2166-2169). IEEE.
- [16] **Risso, F., & Degioanni, L.** (2001). Architecture for high performance network analysis. In Computers and Communications, 2001. Proceedings. Sixth IEEE Symposium on (pp. 686-693). IEEE.
- [17] **Habraken, J. W.** (2003). Absolute beginner's guide to networking. Que Publishing.



## **APPENDICES**

### **APPENDIX A.1 : Sample WinPcap Application**

## APPENDIX A.1



| ID | Time                | Protocol | Destination IP  | Length | Country | Application |
|----|---------------------|----------|-----------------|--------|---------|-------------|
| 1  | 2017-01-14 14:20:39 | UDP      | 239.255.255.250 | 42385  | ZZ      | PRIVATE     |
| 2  | 2017-01-14 14:20:40 | UDP      | 239.255.255.250 | 39301  | ZZ      | PRIVATE     |
| 3  | 2017-01-14 14:20:41 | UDP      | 239.255.255.250 | 39301  | ZZ      | PRIVATE     |
| 4  | 2017-01-14 14:20:42 | TCP      | 192.168.1.4     | 51200  | ZZ      | SSL         |
| 5  | 2017-01-14 14:20:43 | TCP      | 192.168.1.4     | 51200  | ZZ      | SSL         |
| 6  | 2017-01-14 14:20:46 | TCP      | 192.168.1.4     | 51200  | ZZ      | SSL         |
| 7  | 2017-01-14 14:20:50 | UDP      | 239.255.255.250 | 39301  | ZZ      | PRIVATE     |
| 8  | 2017-01-14 14:20:52 | UDP      | 224.0.0.251     | 22853  | CN      | PRIVATE     |
| 9  | 2017-01-14 14:20:55 | TCP      | 40.77.228.250   | 51152  | IE      | PRIVATE     |
| 10 | 2017-01-14 14:20:56 | TCP      | 192.168.1.4     | 85248  | ZZ      | SSL         |
| 11 | 2017-01-14 14:20:57 | TCP      | 40.77.228.37    | 51200  | IE      | PRIVATE     |
| 12 | 2017-01-14 14:20:58 | TCP      | 192.168.1.4     | 51200  | ZZ      | SSL         |
| 13 | 2017-01-14 14:20:59 | TCP      | 192.168.1.4     | 59136  | ZZ      | SSL         |
| 14 | 2017-01-14 14:21:00 | TCP      | 192.168.1.4     | 51200  | ZZ      | HTTP        |
| 15 | 2017-01-14 14:21:01 | UDP      | 239.255.255.250 | 39301  | ZZ      | PRIVATE     |
| 16 | 2017-01-14 14:21:02 | UDP      | 192.168.1.4     | 61403  | ZZ      | DNS         |
| 17 | 2017-01-14 14:21:03 | TCP      | 192.168.1.4     | 85237  | ZZ      | SSL         |
| 18 | 2017-01-14 14:21:04 | TCP      | 192.168.1.4     | 78592  | ZZ      | SSL         |
| 19 | 2017-01-14 14:21:05 | TCP      | 40.77.228.249   | 72963  | HK      | SSH/ATF     |

Figure A.1 : packetCapture Application.

**Table A.1 : PreRequirements for Application.**

| <b>Name</b>                     | <b>Version</b>            |
|---------------------------------|---------------------------|
| <b>Visual C++ CRT</b>           | <b>v12.0 (x86 or x64)</b> |
| <b>Microsoft .Net Framework</b> | <b>v4.5 Full</b>          |
| <b>WinPcap</b>                  | <b>v4.1.3</b>             |



## **CURRICULUM VITAE**

**Name Surname** : Yücel AYDIN  
**Place and Date of Birth** : Samsun / TURKEY – 13.05.1985  
**Address** : Esentepe M. Morgül S. No:5/2 Eyüp İstanbul  
**E-Mail** : aydinyuc@itu.edu.tr  
**B.Sc.** : Atatürk University Electrical and Electronic  
Engineering

### **List of Publications and Patents:**

#### **PUBLICATIONS/PRESENTATIONS ON THE THESIS**

- Aydın Y., Aslan A., 2016: Prevention Against Application Layer 7 DDOS Attacks. *9th International Cyber Security and Cryptography Conference Poster Publication* , October 25-26, 2016 Ankara, Turkey.