Military Technical College
Kobry El-Kobba,
Cairo, Egypt

11-th International Conference
on Aerospace Sciences &
Aviation Technology

SECURE INTRANET USING VIRTUAL PRIVATE NETWORKS

Ismail A. Ismail*, Moatasem M. Abd Allah**,
Gamal A. Osman***, Tamer M. Abo Neama****

ABSTRACT

The network security concepts are targeting to prevent or reduce the risk of different types of network threats, especially during the transfer of secured data among the network nodes. Many security mechanisms are needed to achieve the network security goals. One of the important concepts for securing the transmitted data among open networks is Virtual Private Network (VPN).

Some organizations don't trust on VPN security because it depends on commercial security products based on public encryption algorithms. For this reason, the objective is to make the VPN more trusted.

The presented model aimed to add another layer of security to VPN security which satisfies the privacy, confidentiality, integrity and access control needs by implementing kernel-mode network driver. The implemented driver can be used to intercept the data buffers of the transferred network packets and encrypts its payload by using proprietary encryption algorithms.

KEY WORDS

Network Security, Virtual Private Network (VPN), Network Driver Interface, Specification (NDIS), Transport Driver Interface (TDI), device driver.

* Associate Professor, Military Technical Collage, Cairo, Egypt.
** Dr., Military Technical Collage, Cairo, Egypt.
*** Dr., Technical Research Department, Cairo, Egypt.
**** Engineer, Technical Research Department, Cairo, Egypt.

INTRODUCTION

Security is required in any environment where information is not intended to be open to all. The importance of network security is increased nowadays, due to the dynamic increasing of network threats. Network security is targeting to provide confidentiality, integrity, and availability. There is a set of security mechanisms needed to achieve the network security goals, one of them is encryption that minimizes risks associated with transmitting sensitive information over public and private networks.

One of the important concepts for securing the transmitted data among open networks is Virtual Private Network (VPN) [1- 3]. Some organizations consider the VPN security is not trusted because it depends on commercial security products based on public encryption algorithms.
For that reason, the presented model adds an intermediate layer between network layer and the data link layer in the TCP/IP protocol suit. The added layer acts as a network interface driver intercepts all data packets transferred to the real network interface card (NIC).

The added driver could examine these data packets and have a chance to apply proprietary encryption algorithms. Providing data confidentiality involves making that data incomprehensible, as with encryption [4]. Also the driver can decide to copy, modify, or drop the transferred packets to ensure the privacy, confidentiality, access control, and integrity of the data as it traverses public data network.

This sub layer is introduced by implementing kernel-mode network driver. This driver is used as an intermediate network driver layered between protocol drivers such as TCP/IP and the miniport drivers which talking to the real NIC.

ARCHITECTURE OF THE IMPLEMENTED INTERMEDIATE DRIVER

The Windows Driver Model (WDM) is a specification for writing device drivers. A driver that conforms to the WDM specification is responsible for handling a set of I/O Requests Packets (IRPs) delivered to it by the system.

Intermediate drivers rely upon the device drivers below them in the windows driver hierarchy for access to a physical device. The implemented kernel-mode driver is a network intermediate driver (IM) which is layered between network driver interface specification (NDIS) protocol drivers, such as TCP/IP, and NDIS miniport drivers, talking to real NICs.

The implemented intermediate driver exposes both a virtual adapter that binds to protocol drivers in the system and a protocol edge which binds to miniport drivers in the system.
For each proposed NDIS Intermediate Driver (IM) binding there are actually two logical "adapter" interfaces:

- Lower adapter - A logical interface to the "real" adapter that the NDIS IM driver binds to at it's lower "protocol" edge

- Virtual adapter - The logical interface associated with the virtual adapter that the NDIS IM driver presents to other protocols at its upper "miniport" edge.

Fig.1 presents the architecture of the implemented intermediate driver. All protocols that were bound to physical adapters are now bound to the virtual adapters exposed by the added driver. The implemented network driver interface specification (NDIS) intermediate driver expose one virtual adapter for each physical adapter (NIC) to which it binds on the lower edge.

The system automatically inserts the intermediate driver between all protocols and the physical adapters. The kernel-mode driver attaches itself above the NDIS miniport drivers, talking to real NICs directly. If any other filter drivers is installed (e.g. Antivirus, file monitor) the implemented driver attach itself below the newly introduced driver, this ensures that nothing can monitor or intercept the secure filter operation.

ARCHITECTURE OF THE TRUSTED VPN

The commercial VPNs offer secure communications between network applications using a public or unsecured medium such as the internet through the use of various commercial technologies offering user authentication, data integrity and access control [5]. But there is a need to add the proprietary security layer.

The main goals of this layer are improving privacy, confidentiality and data integrity of the commercial VPN. These goals are achieved by applying a proprietary encryption algorithm and filtering mechanism on the network traverses data packets between computers working on secure applications. Users of these computers need to have the choice to isolate their computers from the commercial virtual private network when talking in a secure application.

This could be applied by installing the implemented kernel-mode driver on these computers. These computers communicating over a trusted virtual private network implemented after installing the driver. These secured sessions are fully encrypted and isolated from the other devices on the VPN. After ending the secured session, computers could communicate with the others normally by uninstalling the driver.

This isolation enhances the privacy and access control of the network and the encryption of the data packets enhances the security and confidentiality. Fig.2. shows the trusted VPN layers and the embedded layer. The installation of the implemented intermediate driver on commercial VPN improves the privacy and security of the communication between users.

This can be achieved by using the tunneling capabilities of the commercial VPN. Tunneling offers the encapsulation of the data packets and traverses these packets after applying the proprietary encryption mechanisms achieved by the implemented kernel-mode driver. Secure tunnels are used to construct VPN over the internet [6].

VPNs were devised to provide remote users with cost-effective access to the private network by eliminating the costly telephone calls and modem banks [1]. For these reasons, security concepts and cost effective benefits, the added driver is located in the VPN to enhance the security of the Intranet using virtual private networks.

The virtual trusted network which implemented by the added driver means that the network is dynamic and the logical structure of the network is formed only of the network devices regardless of the physical structure of the underlying network [3]. Dynamic VPN is a VPN with a high degree of change in terms of implying the necessity for fast reconfiguration and provisioning [7].

INTERACTION BETWEEN THE IMPLEMENTED DRIVER AND WINDOWS NT NETWORK DRIVER COMPONENTS

A device driver is the software that enables a computer to work with a particular device. The software driver tells the computer how to drive or work with the device so that the device performs the job it is assigned in the way it supposed to [8].
All drivers that are written under Windows NT must have a Driver Entry function that acts as an entry point to the driver. The other functions in the driver are known to the application only through the Driver Entry routine.

An application calls functions such as CreateFile, ReadFile, etc that in turn call the NT I/O manager that generates an IRP (input/output Request Packet) corresponding to every function call. Under Windows NT, almost all I/O is packet-driven. Each I/O is described by a work order that tells the driver what to do and tracks the progress of the request through the I/O subsystem.

These work orders take the form of a data structure called an I/O Request Packet (IRP). This IRP in turn invokes the entry point present in the driver for performing the particular operation. Because of the location of the proposed driver between the protocol driver and the miniport driver, it operates directly with the data buffers in the NDIS packets level so the proposed driver is application and protocol independent driver.

Fig.3 illustrates the location of the implemented intermediate driver related to the Windows NT network driver components. The driver manages the Network interface card (NIC). Intermediate driver interfaces directly to the hardware (NIC) at its lower edge and at their upper edge it provides an interface that helps the upper level drivers to:

- Send and receive packets.
- Reset NIC
- Halt the NIC.
- Query NIC.
- Set operational characteristics of NIC.

Implemented kernel-mode network driver lies between the protocol driver and miniport drivers. To the upper level transport driver an intermediate driver acts as a miniport driver, in the same time it acts as the protocol driver to the miniport driver i.e. NIC.

An upper level protocol driver implements a Transport Driver Interface (TDI) or another application-specific interface to provide services to its users. Such a driver allocates packets, copies data into the packets and sends the packets to the lower level driver by calling the Network Driver Interface specifications (NDIS). It also provides a protocol interface at its lower level to receive packets from the next lower level driver.

CLASS STRUCTURE OF THE IMPLEMENTED KERNEL-MODE NETWORK DRIVER

The block diagrams represented in fig.4and fig.5, presents the driver's class structure of the send and receive processes respectively. The structure of the driver begins with initialization phase that includes the calling of a new object of the class driver adapter for giving a chance to accept or reject the binding.

The driver calls the initialize method when a new virtual NIC represented by the driver adapter. At this time the binding to the underlying real NIC is completed already. This method could return NDIS status success to grant the creation of the driver. Or, it might inspect the underlying medium type and/or set-up some private configuration parameters from the Registry Configuration before returning.

Intercepting upper-layer Object Identifier OID requests by Intercepting protocol's OID queries/replies to the NIC. This method is called after the underlying miniport returned its information.

Intercepting upper-layer by handling the send process where the packet filtering in the outgoing direction takes place. The driver is given a chance to examine the original packet submitted by a protocol and to filter the broadcasting packets from the stream of transmitted packets then apply proprietary encryption algorithms on the payload of the data packet after disassemble the packet header from it and save it.

The driver supplies a fresh packet descriptor good for passing one packet down the binding. Then the driver notifies on the completion of submitted transmitted packets. Last call on the object either virtual NIC or underlying binding is being destroyed by NDIS then, resetting the NIC. Notifying on return of the submitted received packet indications.

Intercepting lower-layer receive where the driver in the incoming direction takes place. The driver is given a chance to examine the original packet indicated by the real NIC (Original) and isolate the broadcasting and data packets from the stream of received packets then apply proprietary decryption algorithms on the payload of the data packet after disassemble the packet header from it.

The driver supplies a fresh packet descriptor good for passing one packet up to the protocols. A filter that needs more packets (e.g. it implements some sort of repacketization) has to maintain an internal packet pool and implement the OnReturnPacket() handler.

Intercepting partial receive indication, the framework calls this version of Receive packet handlers when the underlying real NIC miniport uses partial receive packet indications. Non-bus mastering NIC miniport may use this form of packet indication. Intercepting lower-layer Status indications by intercepting the real NIC status changes. The driver is given a chance to examine the associated status buffer (if any) and make changes if necessary to both the buffer and the status code indicated to the protocols. The content of the buffer is status-value and medium dependent. Finally, intercepting NIC power change then destroys the virtual adapter and destructs the driver adapter class.

CRYPTOGRAPHIC ENGINE

The Cryptographic engine is responsible for the encryption and decryption processes. It communicates with the trusted VPN driver through an encryption / decryption request packet (EDRP).

Fig.6. represents the block diagram for the encryption and decryption processes. These processes applied by the implemented driver on the captured packets.

PERFORMANCE ANALYSIS OF THE ADDED DRIVER

In this section the performance analysis is suited as a function of processing time. Since the performance, depends on the data file properties, three data files with different sizes (10K.Byte, 100K.Byte, 1000K.Byte) are considered. Moreover, such files are formed with various types of Internet applications (SMTP, HTTP and FTP).

The performance analysis of the implemented driver copes the analysis of the transfer time taken before applying the driver, the time taken after applying the driver and the time overhead for applying the driver which is the difference between the time before and after applying the driver.

The performance of applying the driver on the transferred data packets is greatly affected by the overhead that may be added to the original data-packet. The overhead, as such, may be one or more of the following [9]:

1- The frame format overhead, added to each data packet such as: (source and destination MAC-addresses, source and destination IP-addresses, and port numbers).
2- The control packet exchange between the client and server to establish the connection. They are changed from service to another. The FTP service has the largest overhead of this type, since it uses two ports to transfer data. Port 20 is using or transfer data files and port 21 for controlling the

session. The added driver filters these control packets to bypass them from the encryption process and save time.

3- The commands between server and client during the session represent another overhead added to the acquired file.
4- The embedded tags in the html text format. Such overhead is associated with HTTP service, and increasing with increasing the data size.

Figures (7), (8), (9) show the time overhead (time cost) for transferring SMTP,HTTP, and FTP files respectively with different file sizes (10KB, 100KB, 1000KB) after applying the added driver.

Time overhead versus file size

The pure SMTP file has main overhead due to mail server commands, another overhead is considered, which is the frame format overhead. The SMTP service is affected mainly by the basic frame format overhead. The time consumed for transferring SMTP large size files is high because of the large number of mail server commands and the data frames size.

The time overhead after applying the driver is small in transferring 10KB, 100KB files size of the FTP application file and increases with the large file size because the driver mainly filter the control packets and encrypt the payload of the other data packets within the transferred frames.

The FTP application files have its main time overhead due to the control packets overhead. In addition the increase of the frame format overhead, needed to transfer the file data (control packets on port 21 and data transfer packets on port 20). The time consumed for transferring FTP application files seems high before and after applying the driver due to the large number of exchanged control packets, and the frame format overhead.

The time overhead for applying the driver is high because the driver filters some of the exchanged control packets which is the main overhead in transferring FTP application files and don't encrypt them, and encrypt the payload of the data transfer packets.

The main overhead of the HTTP files based on the data part due to the embedded tags. The significant consumed time by HTTP according to the HTTP embedded tags overhead, which is associated with the large file size, but in small files the time consumed is smaller than the large files. The time overhead is small with all HTTP file sizes because there are no server commands or exchange control packets.

CONCLUSIONS

Through this paper we have concluded that the description for design and implementation of trusted VPN using kernel-mode network driver is illustrated. This driver works at the kernel level of the operating system. The driver captures all the I/O transmitted network packets.

At the sender side the driver, at first, saves the header of the captured packet in a temporarily memory area. Second, the driver encrypts the content of the transmitted packet's payload. Finally, it assembles the packet and transmits it to the next layer.

At the receiver side, the driver captures all the received packets. First, the driver disassembles the packet's header and saves it. Second, the driver decrypts the received packet's payload. Finally, it assembles the packet and its header and sends it to the upper layer. The encryption engine will be associated with the kernel-mode driver for the encryption services provisioning.

Also we concluded that, each unit installs the trusted VPN driver will not be reached on the network by the other units which are not install the driver. This because the driver encrypts all the transmitted data packets between the connected units, so that, the broadcasting network packets which identify the unit on the network is encrypted. Units which install the implemented driver will be communicated in a trusted VPN connection, and will be isolated from the other units on the network.

The performance of the added layer is calculated through extensive experiments, in which the time overhead of data transfer between users is computed after adding the layer. These experiments are applied by using the most famous Internet application protocols (SMTP, HTTP, and FTP).

These experiments are tried under conditions close to be realized. The selected data files sizes are 10KB, 100KB, and 1000KB. The total overhead for each application protocol is demonstrated. The growth of the time overhead for each file size is presented. It was found that the time overhead of the HTTP application files is the smallest one, the SMTP application files has a highest time overhead at the large file size. The time overhead of the FTP file is large for the small files only but it decreases with the increasing in the file size.

REFERENCES

[1] Paul Izzo, "Giga bit Networks", John Willey & Sons, Inc., USA, 2000.
[2] William Yurcik and David Doss, "A Planning Frame-work for Implementing Virtual Private Networks", IEEE , May/June2001 (Vol. 3, No. 3), pp. 41-44.
[3] Kosiur, Dave, "Building and Managing Virtual Private Networks", New York, John Willy & Sons, Inc., 1998.
[4] Charles P. Pfleeger, "Security in Computing", Prentice-Hall International, Inc., USA, 1989.
[5] Christopher M. King, Curtis E. Dalton & T. Ertem Osanoglu, "Security Architecture Design Deployment & Operation", McGraw-Hill Companies, USA, 2001.
[6] P.C. Cheng, "An architecture of Internet key exchange protocol", IBM systems journal, Vol. 40, No. 3, 2001.
[7] R. State, O.Fester, E. Nataf "Managing highly dynamic services using extended temporal network information model", IM 2001,IFIP/ IEEE International Symposium on Integrated Network Management, No. 1; May2001, pp. 705-718.
[8] Microsoft Press, MCSE Training Kit Networking Essentials Plus, Washington, 2000.
[9] Gamal Aly Osman, "Remote monitoring for Computer Networks", Ph.D. Dissertation, Electronics and communication Engineering Department, Cairo University, September 2002.

FIGURES

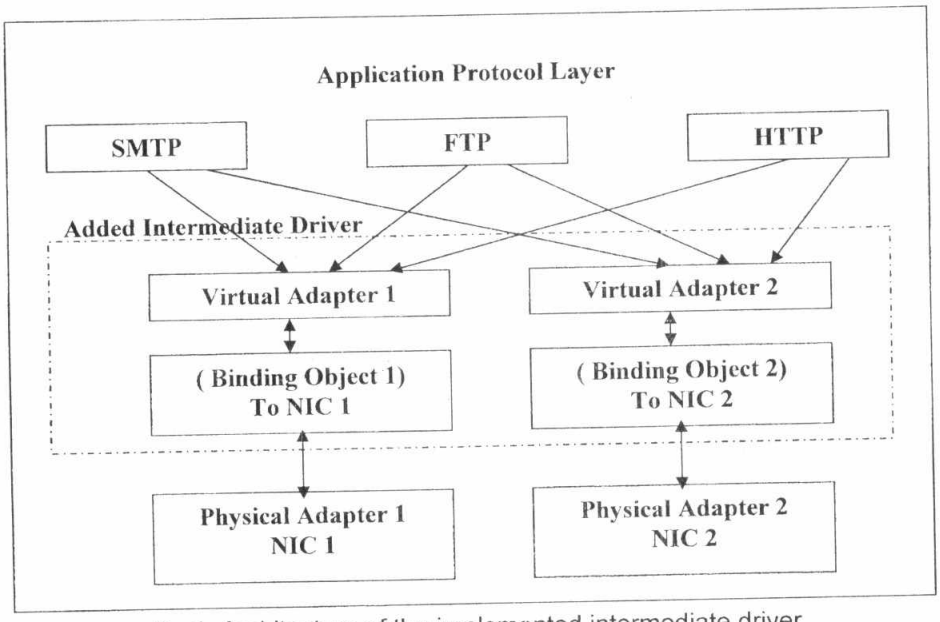

Fig.1. Architecture of the implemented intermediate driver

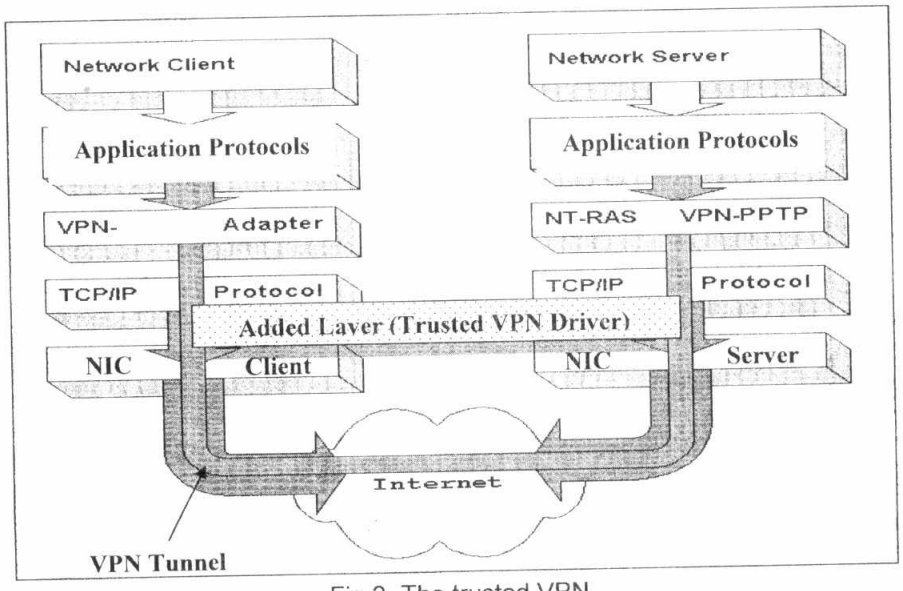

Fig.2. The trusted VPN

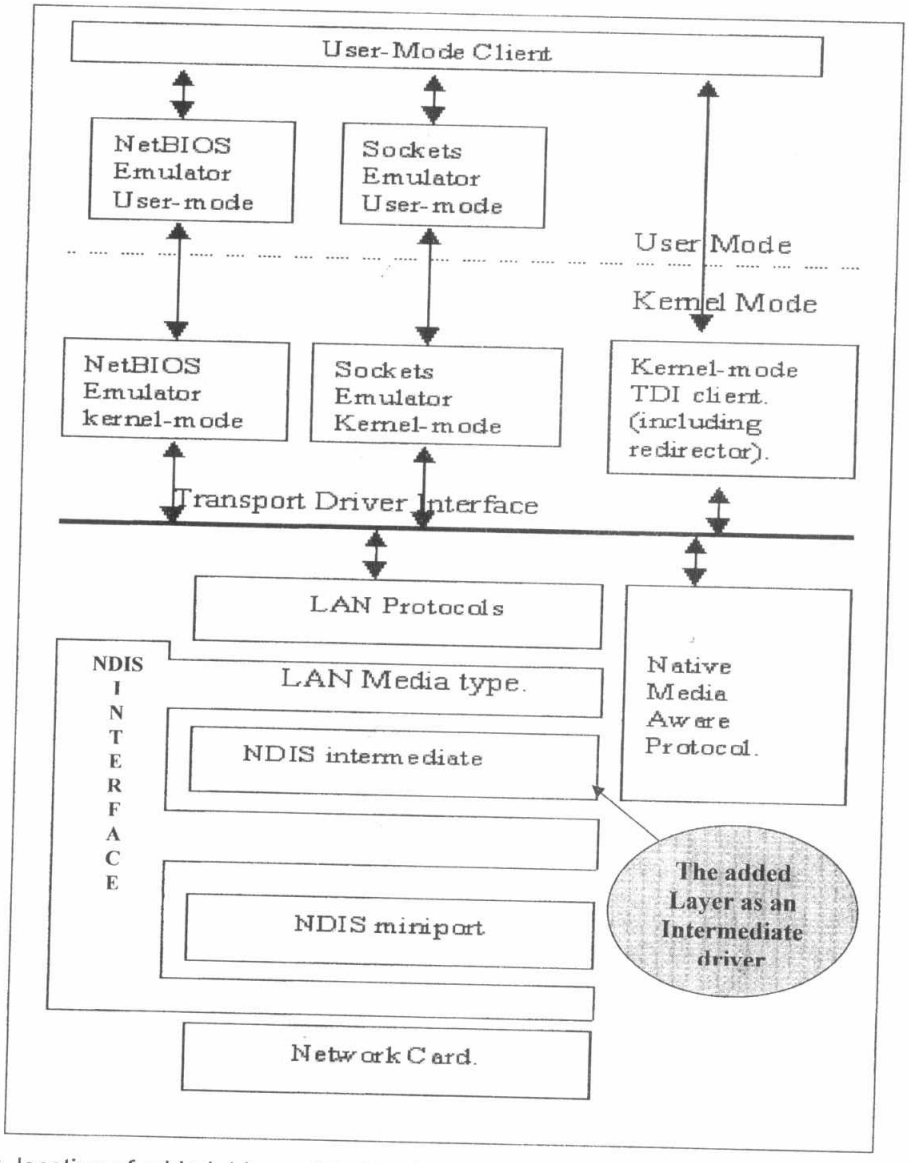

Fig.3. location of added driver related to the Windows NT network driver components

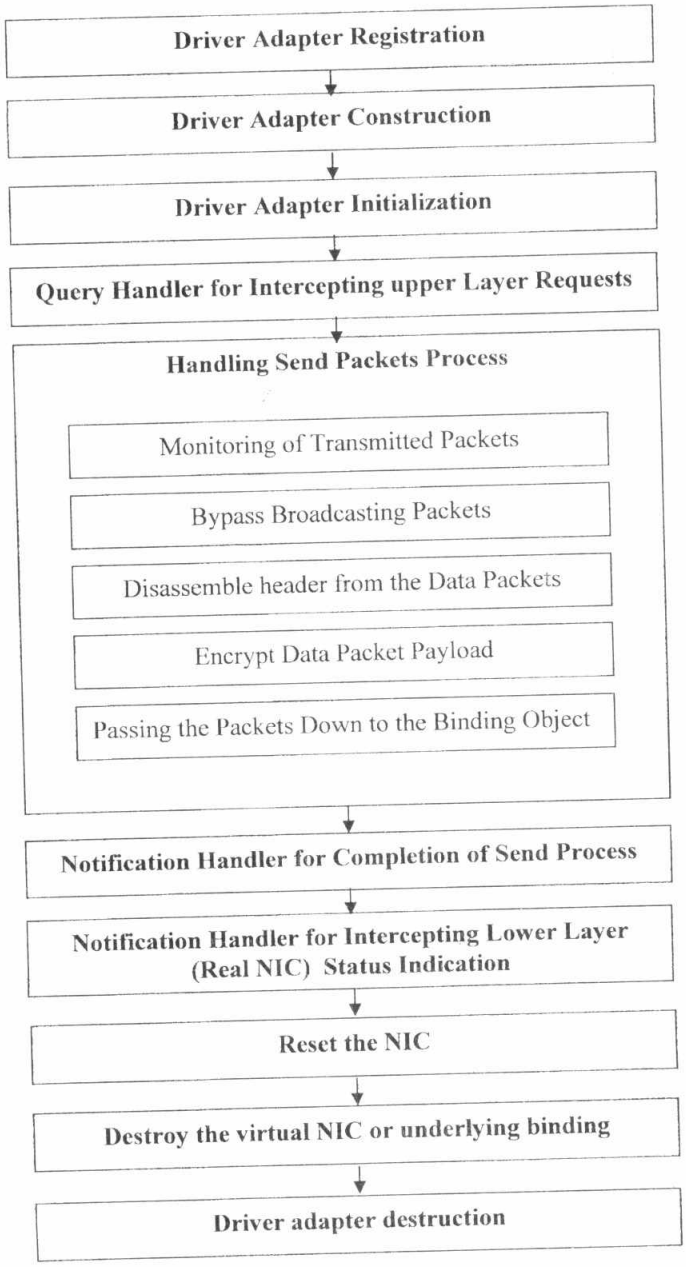

Fig.4. The added driver's class structure of the send process

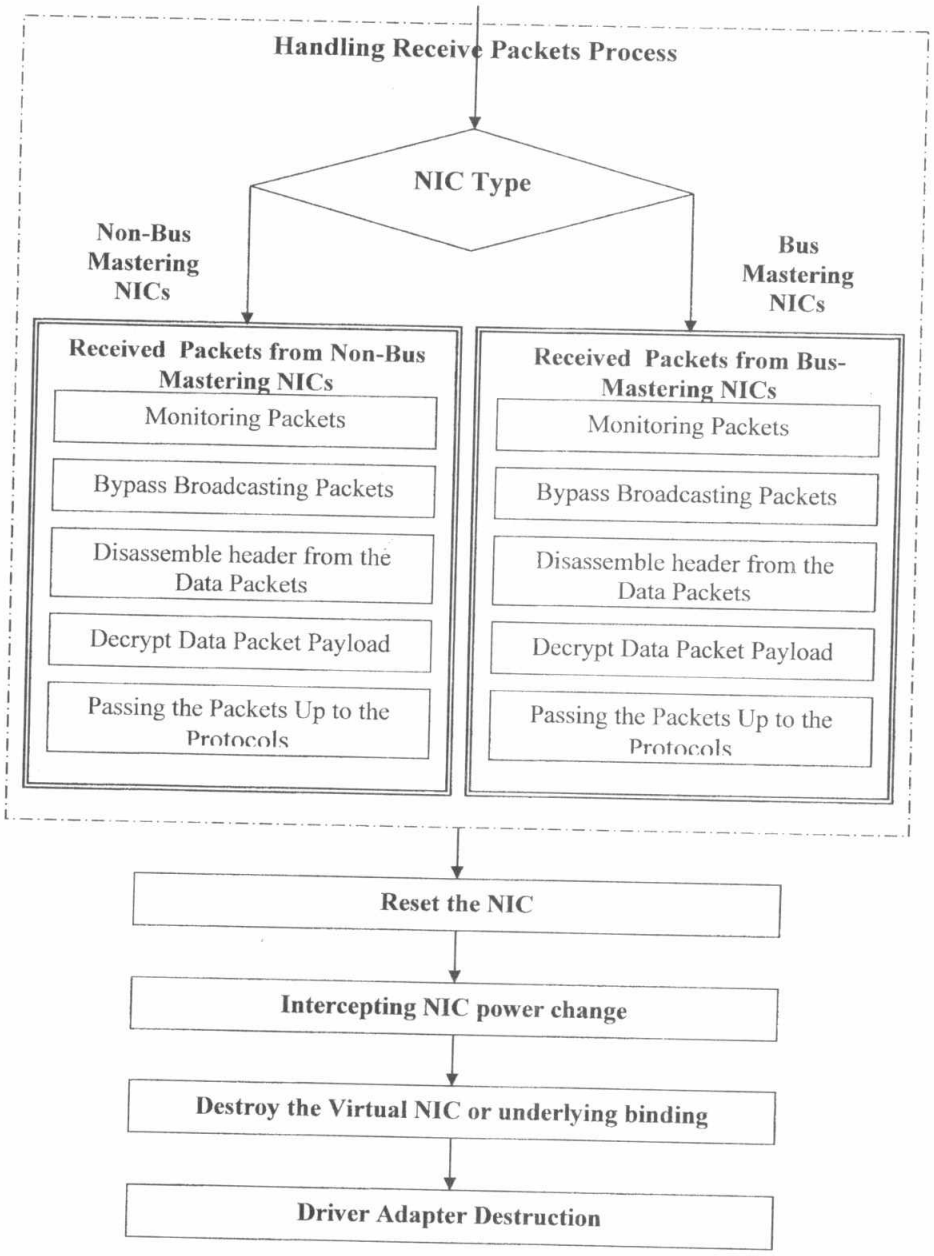

Fig.5. The added driver's class structure of the receive process

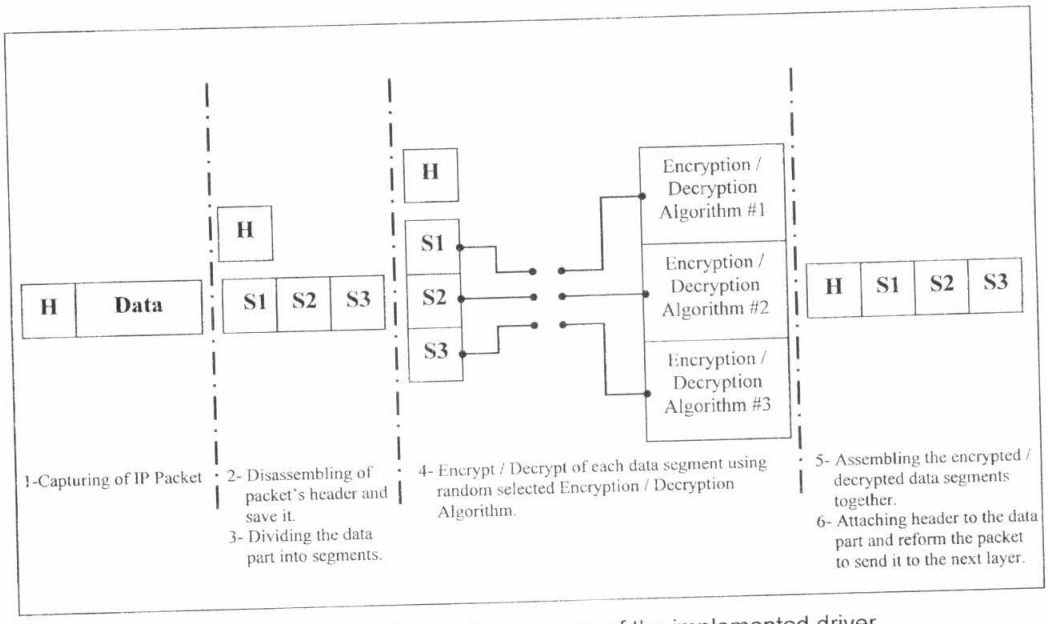

Fig.6. Encryption / Decryption process of the implemented driver

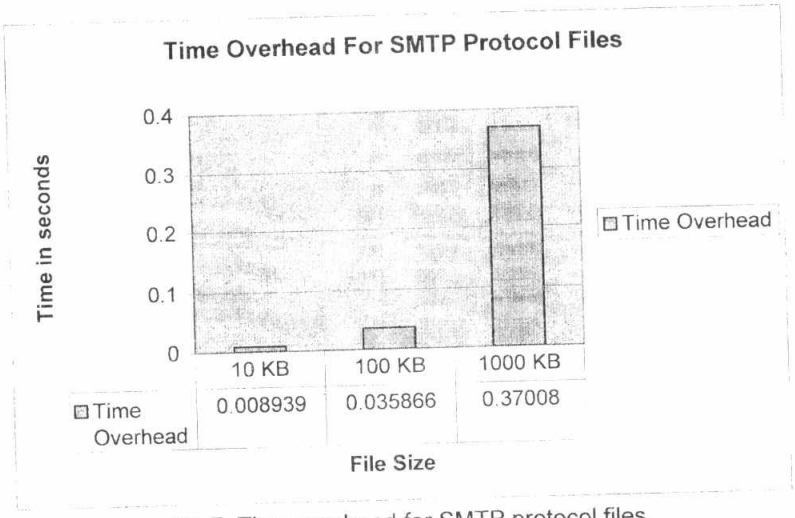

Fig.7. Time overhead for SMTP protocol files

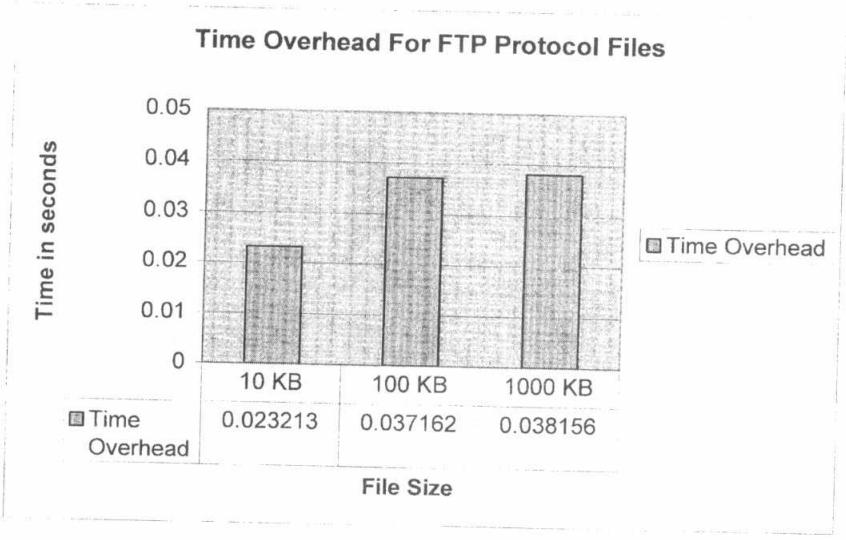

Fig.8. Time overhead for FTP protocol files

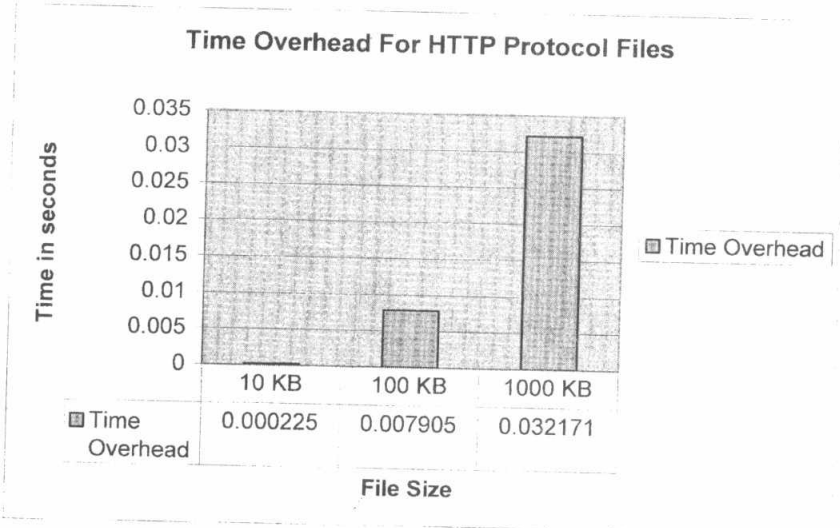

Fig.9. Time overhead for HTTP protocol files