Al-Khwarizmi Engineering Journal

Al-Khwarizmi Engineering Journal, Vol. 5, No. 4, PP 69-98 (2009)

Windows-Based Special Applications Router

Omar A. Athab* **Ahmed S. Hadi*** **Sufyan T. Faraj****

* *Information and Communication Engineering Department/ Al_KhawarizmiEngineering College/ University of Baghdad*

** *Computer College/ Al-AnbarUniversity*

(Received 16 March 2008: accepted 17 August 2009)

Abstract

The design and implementation of an active router architecture that enables flexible network programmability based on so-called "user components" will be presents. This active router is designed to provide maximum flexibility for the development of future network functionality and services. The designed router concentrated mainly on the use of Windows Operating System, enhancing the Active Network Encapsulating Protocol (ANEP). Enhancing ANEP gains a service composition scheme which enables flexible programmability through integration of user components into the router's data path. Also an extended program that creates and then injects data packets into the network stack of the testing machine will be proposed, we will call this program the packet generator/injector (PGI). Finally, the success of the node architecture and its prototype implementation is evaluated by means of a few practical applications.

Keywords: *Active Network, Active Router, User Components, Windows Operating System, Active Network Encapsulation (ANEP), Packet Generator /Injector (PGI).*

1. Introduction

"Active networks allow individual user, or groups of users, to inject customized programs into the nodes of the network."Active" architectures enable a massive increase in the complexity and customization of the computation that is performed within the network, e.g., that is interposed between the communicating points."

In traditional networks an intermediate node executes ordinary computations on packets. All packets are treated in the same way: nodes only forward packets towards the right destination. Instead, an active node makes difference between packets. They contain programs that have to be executed on them. The difference between a traditional network (store-and-forwarding model) and an active one (store-compute-and-forwarding model) is that, in traditional network the delivery process is static, relatively passive, because all packets are treated individually. While in active network delivery process is dynamic, where the packets become smarts, i.e., they contain their own handling instruction that enable these packets to arrive, execute, and move to their destination [1]. A key characteristic of this technology is the ability to rapidly create, deploy and manage new network services in response to user demands.

2. Literature Survey

Since the focus of the work presented in this paper concerns the development of active node architecture, this section focuses primarily on completed works in active systems design.

D. J. Wetherall and D. L. Tennenhouse have first pursued the idea of placing program fragments into internet protocol (IP) packets as part of the ActiveIP project [6, 7]. Initially, they studied the potential of placing small programs within the option fields of IP packets. These so-called active options, encoded in Tcl [5] language in their prototype implementation, were executed by modified network nodes as the packets traversed the network.

Active Node Transfer System (ANTS) [8, 2], provides a capsule programming model. Capsules are packets that encapsulate data with a customized forwarding code. Applications use the

network by sending and receiving capsules via active nodes. When a capsule arrives at an active node, the type field is used for de-multiplexing to the corresponding forwarding routine, before the corresponding routine is executed to forward the capsule. ANTS provide active node application programming interface (API) calls to query the node environment. The demand-pull mechanism is used to obtain code from the previous node that the capsule visited. The ANTS prototype is implemented in Java under UNIX operating system.

The Smart Packets project [9], emphasizes particularly on applying active networking technology to network management. It aims at addressing scaling problems that are inherent in typical polled managed devices rather than aiming for general transport mechanisms such as ANTS. Smart packets are encapsulated within ANEP packets and ANEP packets are encapsulated within an IP packet using a specific option (router alert). The Smart Packets architecture expects all programs to fit within one Ethernet maximum transmission unit (MTU). There is no existing language that had a compact enough representation for Smart Packets environment. As a result, Sprocket [10] and Spanner [11] languages are developed as part of the Smart Packets project.

Switch Ware Active Network Architecture [12], consists of three layers: active packets, active extensions and a secure active router infrastructure. Active packets carry programs consisting of code and data to replace both the header and payload of traditional packets. As a consequence, a new programming language for Active Networks, known as PLAN [13, 14], is designed and implemented. Active extensions [15], which are not mobile, form the middle layer of SwitchWare architecture. They communicate with other routers via active packets. It is programmed in Caml. A secure active router infrastructure forms the lowest layer of SwitchWare architecture. It provides a secure foundation on which the other two layers are built. Secure Active Network Environment (SANE) [16] is designed to embody secure active router infrastructure. The role of SANE is to ensure that the presumptions of the other system elements are true.

MIT's Click [17] is software architecture for building flexible and configurable routers. A Click router is "configured" from packet processing modules called elements. Individual elements support simple router functions such as packet routing, queuing, or scheduling. A complete router configuration is defined by a directed graph whose nodes are the elements. A Click router configuration is determined at compile time. The elements are internally represented by C++ objects that are inter-linked with each other through object references under Linux OS. Packet passing between functional elements is thus simply a matter of passing memory pointers between objects.

Router Plugins [18], aims to build a flexible network subsystem that offers the ability to select implementations (or even instances of the same implementation) of router components, called plugins, on a "per-flow" basis. Plugins are binary code modules that can be dynamically loaded and unloaded into the router kernel at run-time. NetBSD, which is used as the base platform for Router Plugins, provides appropriate kernel support to load modules into the kernel. The Plugin Control Unit (PCU) provides the "glue" to bind individual plugins to the network subsystem.

The application level active network (ALAN) system [19], introduces value added network services by means of an overlay active network infrastructure. The system is assembled from standard IP applications and servers connected to the Internet. Active processing "inside" the network takes place in so-called dynamic proxy servers (DPS). The active programs, called proxylets [20], act as communication proxies for data streams dispatched through the dynamic proxy servers. ALAN requires the data streams to be explicitly addressed to the proxy servers. The current implementation, known as FunnelWeb [21], is based on RMI of Java 2.

The Lancaster Active Router Architecture version 2 (LARA++) [22, 3] proposes an active router architecture. The key building blocks of LARA++ are: the active NodeOS, the policy domains, the processing environments, and the active and passive components. The Active NodeOS provides low-level system service routines and policing support to enable controlled access to node-local resources and system services. Policy Domains (PDs) form the management units for resource access and security policies which are enforced on every active program executed within the PD. Processing Environments (PEs) provide the protected environments for the safe execution of active code. Active Components (ACs) are the units of active code processed within the PEs. To assure platform independence, there are two prototype LARA++ implementations for both MS Windows and Linux being developed.

3. Work Objectives

This work primarily aims to get the following:

1. Proposing architecture for active network system, regarding the available tools.
2. Implementing a prototype for active router.
3. Investigation of new techniques in the implementation. Rather than standing in the old layering model in networking, a component-based approach was introduced. Also, entering the challenge of in-kernel programming technique instead of the common user-mode approach. In addition to use a closed-source-code OS "windows", which is rarely used in in-kernel network implementations.

4. Paper Layout

Section 5 presents the design of the active router (AR) architecture. This central part of the paper describes in detail how AR operates and how the component-based active node architecture enables network programmability through flexible integration and extensibility of network functionality. In addition to the basic node design, special focus is placed on the service composition framework.

Section 6 then describes the ongoing implementation efforts of developing prototype nodes of the AR architecture. Due to the considerable extent of the AR architecture, this section focuses primarily on validating the key aspects of the design through a 'proof-of-concept' implementation.

Section 7 continues with a qualitative and quantitative evaluation of AR and its prototype implementation. It evaluates how the AR architecture satisfies the objectives and requirements. Finally, section 8 concludes the paper by drawing together the main arguments of this work. It also describes further work that could be carried out based on this line of research.

5. AR Architecture Overview

The overall architecture of the proposed active network has been divided into three functional parts: the Component Distributor (CD), the Packet Manipulator (PM) and the" Proof-of-Concept" part.

The first part; the component distributor (CD),concern the transferring of user components from a Privileged End-System (PES) or network administrator (ADMN) system to the active router (AR). The management of the transferred component is also the responsibility of the CD.

In other side, the PM functional part extends the OS networking stack such that it can intercept the in-bound packets that enter the AR and discriminate among the various types of packets. After distinguishing the type, the PM forwards the packet to the proper component to be serviced.

The third functional part is dedicated for the Proof-of-Concept of the idea in this paper. It also contains the design and implementation of Packet Generator/Injector (PGI) system. Its main purpose is the construction and sending of two types of packets: Active and Traditional.

The proposed architecture is designed to extend exiting routers by layering active network-specific functionality on top of the router operating system. A generic high-level active network layer enables cross-platform programming and processing of active programs. Low-level functionality of the AR architecture as provided by the active node operating system (NodeOS) is directly integrated with the router OS in order to maintain good performance for the active processing. The following sections explain in details the first two parts of the proposed AR, which there positions are also shown in fig. 1. The third part will be explained in section 7.

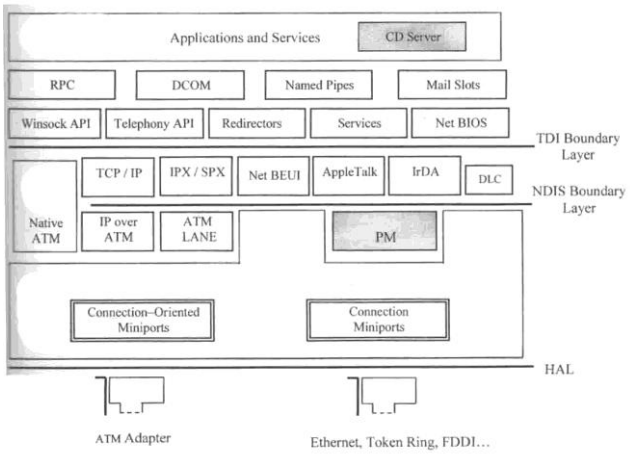

Fig.1. Positions of the CD and PM Units with Respect to the Windows Layers of the Proposed AR

5.1. Component Distributor (CD)

The proposed Component Distributor allows the user to load new components in the AR, where it is responsible for the fetching of User Component (UC).

The UC is a program performing either protocol processing or value-added function to the packet. The code of the UC is written by a Privileged-End User (PEU) or purchased from software third party, then sent from any PES to the AR using the CD unit. The UCs are distributed in the form of pre-compiled machine code. Thus by using UC the AR is programmed individually through out-of-hand instantiation of active program. The UC is in the form of Dynamic Link Library (DLL) file. The CD unit provides the capability of reading, packing, and then transferring the UC from a PES to the AR.

The second function of the CD is to upload the UC to the AR using a certain Protocol; the file transfer protocol (FTP) [27]. The CD unit operates in a client/server fashion. The AR represents the server, whereas the PES represents the client. Hence, jobs of the CD can be summarized as: packing, uploading, and controlling the UC.

5.2. The Packet Manipulator (PM)

The packet manipulation begins with intercepting the packet and ends with forwarding it to the appropriate user component to be processed there. Hence, in addition to catching a packet, the PM performs a light firewalling, lifting the packet from the kernel to the user mode, recognizing its type, and finally (if required) dispatching the packet to the user component that it whishes for processing.

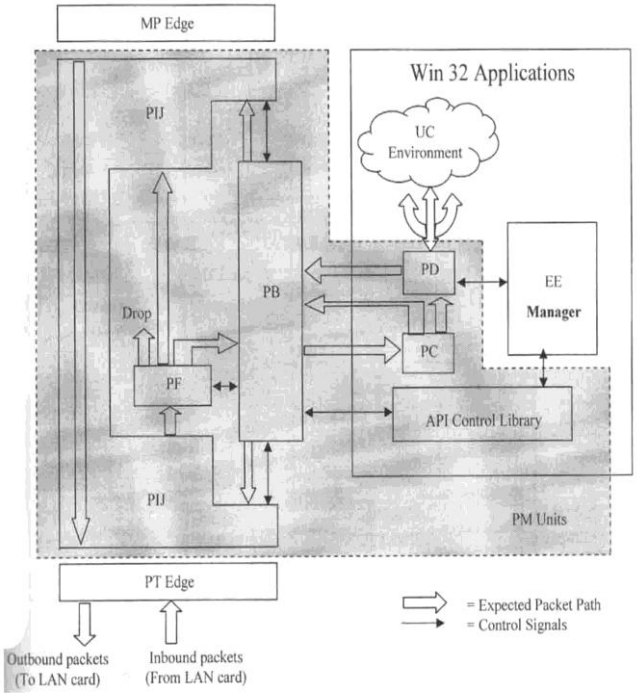

Fig.2. Block Diagram of PM Architecture.

Consequently, the PM architecture was further divided into the following functional units: The Packet Interceptor/inJector (PIJ), Packet Filter (PF), Packet Bridge (PB), Packet Classifier (PC) and Packet Dispatcher (PD). It is noteworthy to state that the packet classifier and packet dispatcher are residing in the user-space of the AR. The packet bridging unit behaves as a channel between the user space units and the rest kernel space units (interceptor and filter). A simple block diagram of the designed PM is shown in fig. 2. The figure also depicts paths that may be taken by packets that passing the AR. The following sections illustrate the PM units briefly.

5.2.1 Packet Interceptor/Injector (PIJ)

The Packet Interceptor/injector provides the interface between the active network environment and the data path on the node. The Packet Interceptor (PI) is responsible for intercepting the network traffic traversing the node and passing it to the active network environment for processing. The Packet Injector (PJ), by contrast, re-injects the network data back into the default forwarding path on the node or sends it directly through one of the outgoing interfaces.

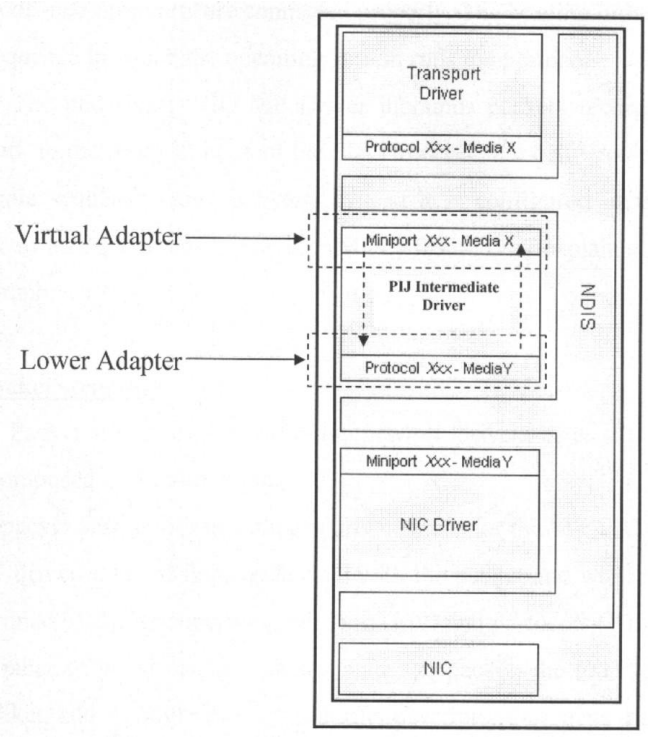

Fig.3. Simple Diagram of the Proposed Skeleton IM Driver (PIJ unit).

A network driver interface specification intermediate (NDIS IM) network driver was proposed to realize the PIJ under windows 2000 OS. We choose

this driver in this paper because it's well documented and it's in excellent location in the stack, which will gives a lot of control over network packets without affecting other network protocols in the stack. A simple diagram of the proposed skeleton IM driver (PIJ unit) is shown in fig. 3.

The proposed PIJ can be loaded (or unloaded) dynamically at run-time (without interrupting the entire system). This allows the AR to dynamically activate (and de-activate) the active network functionality on a node. Note that removing the packet interceptor completely disables any AR specific processing on the data path.

5.2.2 Packet Filter (PF)

The proposed project provides a programmable PF on the read handle. The PF introduces a list of network source/destination address ranges and an "action" to be performed on packets that match the firewall criteria. The actions are:

Block: Have the PF drop the matching packet from the normal packet flow.

Pass: Have the PF allow the matching packet to pass up to the PT driver as in the normal flow.

Read: have the PF pass a copy of the received packet to the packet bridge which is the next unit in the AR.

5.2.3 Packet Bridge (PB)

The paper establishes so-called Packet Bridge to provide the means to transfer network data to and from the UC. The PB unit targets to facilitate the interface between the kernel and user sides of the PM. Hence, PB consists of modules in both the kernel and the Win32 application sides. It is noteworthy to recall that the PIJ and PF units are placed within the kernel space of the proposed AR. However, to make the system more flexible it is suggested to load the UCs in the user space of the OS.

5.2.4 Packet Classifier (PC)

At this point a copy of complete packet has been captured and reached the PC. The object of the PC is the discrimination among the various types of packets that may pass through the PM.

5.2.5 Packet Dispatcher (PD)

The PD defines the "route" through the UC space for the active data packets (ADPs) passing a node. The PD plays a central role in the service composition process. It determines based on the set of component identifier (CID) in service composition field of the ANEP header which UC(s) are involved and in which order they should process the ADP.

After the UC(s) finished it's processing on the ADPs, the PD returns the packet back to the windows network stack through the PB. The PB, in turn, either injects the packets into its previous default forwarding path (virtual adapter) on the node or sends it (lower adapter) directly through one of the outgoing interfaces. If the ADP is re-injected to the virtual adapter, the routing operation is applied. It is performed strictly like the conventional routers.

5.3. Service Composition

A composite service [28] is constructed from a set of components by means of a composition method. The composition method determines the set of software components needed to compose a service and the bindings to join these components. A PD is designed to associate Active Data Packets (ADPs) passing an active node with the appropriate active extension (or UC). For this reason, ANEP [4] has been proposed as a means to assign packets passing a network node to active computations.

5.3.1. Enhancing ANEP

The basic ANEP header [4] specifies a 16-bits field for the type identifier (ID) of a single executing environment (EE) that must process the ADPs in the AR. But, in this paper, the concept of component-based services is envisaged to achieve a good flexibility in introducing functionalities for the ADPs. This means that we have either multi-Execution Environment (where each one represents a single component) or single EE (contains all installed UCs).

In the two assumptions there is a limitation in determining which components(s) and in what order are appropriate to process the ADPs. According to the basic ANEP header, the type ID field in the entering ADP can assign only one component to process the packet. This is a big restriction. This will restrain the AR to be really active and flexible. Furthermore it may weak or omit the principal of component-based services in the AR; services that require more than one component can not be achieved in such fashion. Therefore, two proposals are presented below to enhance the original ANEP to overcome this limitation.

5.3.1.1. ANEP Proposal 1

Using the same ANEP structure but with giving a unique type ID not only for each single components but also for each combination of 2 or more components with various sequences. For example, if there are two components (namely A and B) installed in the AR, the coding of the type ID field, to assign one of them or both, appears as in table 1. The statement (A then B) that appears in table 1 means that the active packet like to enter component "A" processing first and then component "B". It is a simple service composition method. However, the same context can be followed for three or more components.

Table 1,
Type ID coding of active component for proposal 1.

Component Name	Type ID
A	1
B	2
A then B	3
B then A	4

5.3.1.2. ANEP proposal 2

In which the type ID field is not used to indicate exactly the UC, instead it is exploited for indicating the count of UCs that may be composed to introduce the service to the active packet. The Component ID (CID) of the component itself is described in a new proposed variable-length field which we call "service composition" field. It is placed after the basic header and before the options field. This new field consists of the CIDs of the components that must be composed to create the required active service.

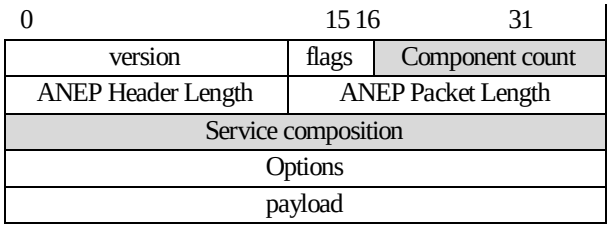

Fig.4. Format of ANEP Proposal 2.

The order at which the CIDs appear in the service composition field is considered as the sequence of the components that will be composed in the active network node. The proposed ANEP format is shown in fig. 4. Using this format, the lack of service composition capability in the original ANEP can be avoided.

- **Service Composition field format**

The proposed format of the service composition field is shown in fig. 5. It is divided into subfields, each one contains a CID of the component to be processed and its length is 16 bits. Conceptually, the count of components that can be assigned in the service composition field is bounded by the count of component (CC) field value, which is not exceeds 2^{16} as maximum edge.

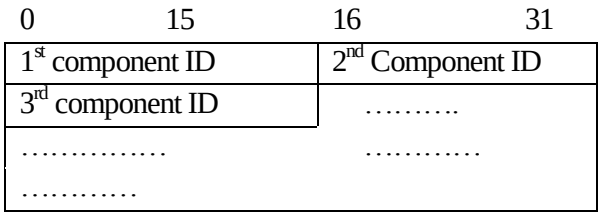

Fig.5. Format of "Service Composition" Field.

At a first glance, one may think that the 16-bits long component ID field of each component will cause a large overhead on the system, especially if there are a large number of components required to be composed to provide the suitable processing for the packet. However, this is not true, because normally the components that need to be composed may not exceed about 10 components (i.e. about 20 bytes overhead only out of 1500 bytes long for IP protocol packet).

5.3.2. Discussion

- **Proposal 1**

Advantages

1- Functionality better than the original ANEP format since it provides a single and multi-component service composition with controlled order.
2- Relatively, No additional processing time needed over the original ANEP since the same fields are used.

Disadvantages

1- Type ID range available to identify components is decreased because of the range that is wasted in covering the multi-component service composition IDs with various ordering.
2- The active components and the composed services are offered by the active node designer or administrator and not the active network user who construct the active packet. The active node designer, for any reason, may not offer all the probabilities of services that may composed by different components or he may impose a charge for specific services. This might represent a contradiction with

the idea of active networking that depend on, as long as possible, moving the programmability from the active node administrator to the privileged users for more flexibility.

- **Proposal 2**

Advantages

1- Like in ANEP proposal 1, it has functionality better than the original ANEP format since it provides a single and multi-component service composition with controlled order.

2- It keeps the wide range of type ID field by providing 16-bit field for each single component in the service composition field. This property supports the component-based principal of the network node by keeping the capability of building different useful services by small program components.

3- The PEU (or the packet producer or installer) himself would be the composer of the services and not the active node designer or administrator. This evidenced that the end-user is the first beneficial of the active network idea.

Disadvantages

1- The overhead is more than both the original ANEP and proposal 1 formats because of the additional service composition field.

2- It requires more processing time than both the original ANEP and proposal 1 formats because of the new appended service composition field and the CC field.

A brief comparison among the original ANEP format and the two proposals for enhancing ANEP is shown in table2.

Table 2,
Brief Comparison Among ANEP, ANEP Proposal 1 and 2.

	Function	Original ANEP	ANEP Proposal 1	ANEP Proposal 2
1	Services Component	Single Component Services	Single and Multiple Component Services	Single and Multiple Component Services
2	Functionality	Fair Functionality	Better Functionality	Better Functionality
3	Processing Time	Required the Original ANEP Processing Time	No Additional Processing Time	Require More Processing Time than the Original
4	Type ID Range	Wide Type ID Range	Type ID Range Less than Original	Wide Type ID Range
5	Flexibility Of Services	Services are Restricted Because it Depends on Single Component	Services are Flexible But Offered by the Active Node Administrator	Services are Flexible Because it is Fully Controled by the End User
6	Bit Overhead	Original Bit Overhead	Similar to the Original Bit Overhead	Overhead is Large thab the Original

5.3.3. Conclusion

According to the above discussion, it is evident that proposal 2 for enhancing the basic ANEP header is preferred. Therefore, proposal 2 will be adopted throughout the work presented in this paper. One of the contributions, of the proposed AR is the novel service composition scheme for active services.

6. AR Implementation

This section describes the ongoing efforts to engineer a prototypical realization of the proposed active router (AR) architecture.

6.1. Component Distributor Implementation

6.1.1. CD Server

The CD server is implemented using the FTP

server provided by windows OS. The FTP service is available already but not activated. To activate this facility, information services (IIS) in the Add/Remove Windows Components dialogue must be chosen. In the internet IIS dialogue; the FTP service should be selected. The default folder used to cache the UCs in the AR is located in C:\Inetpub\FTProot. Any FTP client can, till now, only visit this folder, but it can not read or write files on it. To enable the read and write on UCs in the FTP root folder, the Read and Write boxes must be checked in the home directory of FTP server properties. The FTP server properties can be reached from IIS service in the administrative tools within the control panel.

6.1.2. CD Client

The CD client in the PES side is realized by exploiting the FTP client capability which is available in with the internet explorer under Windows OS.

PEU can display the two side screens (FTP client and server) in his workstation. The FTP client in the PES side is realized with the internet explorer under windows OS. While the FTP server is realized by inserting the IP address of the target AR in the address bar preceded by FTP acronym to connect to the FTP server. UCs can be uploaded easily by drag and drop from the PES screen to the AR screen. PEU, also, can control (delete partially or completely and rename) the installed UCs in the AR. But this control capability obeys the previous configuration of the FTP server.

6.1.3. Structure of UC

The proposed UC is implemented as two files: code and configuration files. Code file contains the source code required to process the ADP actively to enforce a certain protocol or achieve a value-added function. The code should be written in high-level language support Microsoft Visual Studio application programming interfaces (APIs). In this paper, C++ language is used to program the source file.

Concerning the code file of UC, PEU can either upload it as it is (source code file) or convert it to a dynamic link library (DLL) file. In this implementation, only the latter case is adopted. DLL is a module contains function(s) and data. A DLL is loaded at run time by its calling module. When a DLL is loaded, it is mapped into the address space of the calling process. DLLs can define two kinds of functions: exported and internal. The exported functions can be called by other modules. Internal functions can only be called from within the DLL where they are defined. Although DLLs can export data, its data is usually only used by its functions.

The second part of the expected UC is the configuration file. Actually, it is an initialization (ini) file that contains configuration data (i.e. idiom terms) for Microsoft Windows based applications. It allows a program to store initialization data, which can then be easily parsed and changed. "ini" files are used to store configuration information for applications programs.

To provide more flexibility for PEU, in this project the "ini" file is created using simple C language routine. The file consists of only a single C-language structure (Servlnfo structure). This structure, in turn, contains the name of code file of UC (the .dll file name) in a string format and a control buffer with its length. The control buffer is involved for configuration purposes (if required).

6.2. Packet Manipulator Implementation

The PM is implemented using IM driver (in the kernel space), the PC and PD units are realized in the user-space as an extension of the driver.

6.2.1. IM driver

The foremost and significant step in realizing the PM architecture is the implementation of a simple "pass through" IM driver. A "pass through" (or passthru) driver means a do-nothing pass-through IM NDIS driver. It abstractly performs the basic principles of initializing and setting up an IM driver. This driver exposes a virtual adapter for each binding to a real physical adapter. In turn, the overlying protocol drivers can bind to these virtual adapters as if they are real adapters.

However, the passthru driver has been implemented as six software modules: Main, Adapter, NDISreq, Recv, Send and Status. In addition, there are other auxiliary modules that are related to power management of the driver such as the reset and shutdown states and the plug and play (PnP) capabilities.

The following subsections briefly discuss the developed passthru driver modules. The routines of these modules may require prior knowledge of NDIS drivers' development:

A) <u>Main Module:</u> This module consists of the routines that load the passthru driver and initialize its driver-wide data structures and resources. In

addition, the main module register and de-register the input output control (IOCTL) interface and the device object that are used by the user-mode applications to communicate with the IM driver.

B) Adapter Module: The adapter module functions are called by the NDIS wrapper, for example to allocate and de-allocated a passthru virtual adapter instance, to halt or shut down the driver upper edge, to query information about the capabilities of the passthru and to request the driver to bind (or unbind) to an underlying MP NIC driver. Also there are routines capable to allocate and de-allocate a context pool to be used for exchange of data between the adapter and the applications.

C) NDISReq Module: Mainly the functions of this module related to NDIS requests to the passthru driver. NDIS may query information about the status of the IM or request changes in the state information that the passthru maintains. Each NDIS driver contains its own management information block (MIB) [23]. It is an information block in which the driver stores dynamic configuration information and statistical information that higher-level drivers or applications can query or set.

D) Status Module: Status notifications initiated by the underlying NIC MP driver are indicated using functions of this module.

E) Send Module: NDIS calls the functions of this module to transmit a single (or multi) packets to the underlying NIC driver. Then, NDIS also indicates the completion of the send operation to the IM driver.

F) Recv Module: After the underlying NIC driver indicates a received packet to the wrapped NDIS, the NDIS in turn indicates this packet up to the passthru driver by calling the functions of the Recv module.

Two standard functions were written to receive the indicated packet (as recommended by driver development kits (DDK) documentation). These are the PtReceivePacket and PtReceiveIndication functions. The two functions must be exposed by the passthru driver to the underlying MP driver. Nevertheless, it may not be easy to guess which one the NDIS will decide to use. In either case, the passthru performs the minimum manipulation here to just forward the packet up to the overlying PT driver. Exposing the two receive functions is mandatory. It is important to be noticed that the routines mentioned above are only the essential routines that must be included according to Microsoft DDK documents [23].

6.2.2. PIJ Unit

The implemented passthru IM driver simply passes the packet data along without modification. Also, this driver stops short of actually illustrating any observable function. To be of any actual use, the developer must take the next step and add functionality of his own to the implemented skeleton driver.

Also, the passthru driver requires an accurate tracking to get a copy of a complete received packet. In addition, when gaining a copy of a packet it may be needed (after processing, if any) to re-inject the packet again into the IM driver to complete its passage up in the networking stack. As stated in Recv module, PtReceivePacket function or PtReceiveIndication function may be called by NDIS to indicate a received packet. Internal considerations in the operating system determine which one is used. Other crucial considerations such as avoiding interrupting with Windows normal activities and take care in allocating and de-allocating of memory storage areas must also be taken into account in the PIJ implementation.

However, two approaches are available in PIJ realization (or extending IM driver, in general). The developer either enforces the above step-by-step (manual) implementation or using a cloned packet approach. The cloned-packet approach has been realized by adding a smart module called UTIL module, which support Recv module. UTIL module consists of routines deal with allocating and de-allocating packets, memory, and buffer descriptors, and tracing the packet path in the send and receive routes to aggregate its hashed portions. Also, UTIL module support the capability of operating with the lower-edge (lower adapter) and the upper-edge (virtual adapter) and making read and write with these two bindings of the IM driver in kernel level. In addition, it provides a complete packet in a clear buffer in the kernel-mode. Flowcharts of the implemented receive functions (PtReceivePacket and PtRecveiveIndication) of PIJ unit are shown in figs. 6.a - 6.m. functions preceded by UTIL acronym are belong to PCAUSA [25] code. After realizing the PIJ, the next step is to implement the Packet Filter.

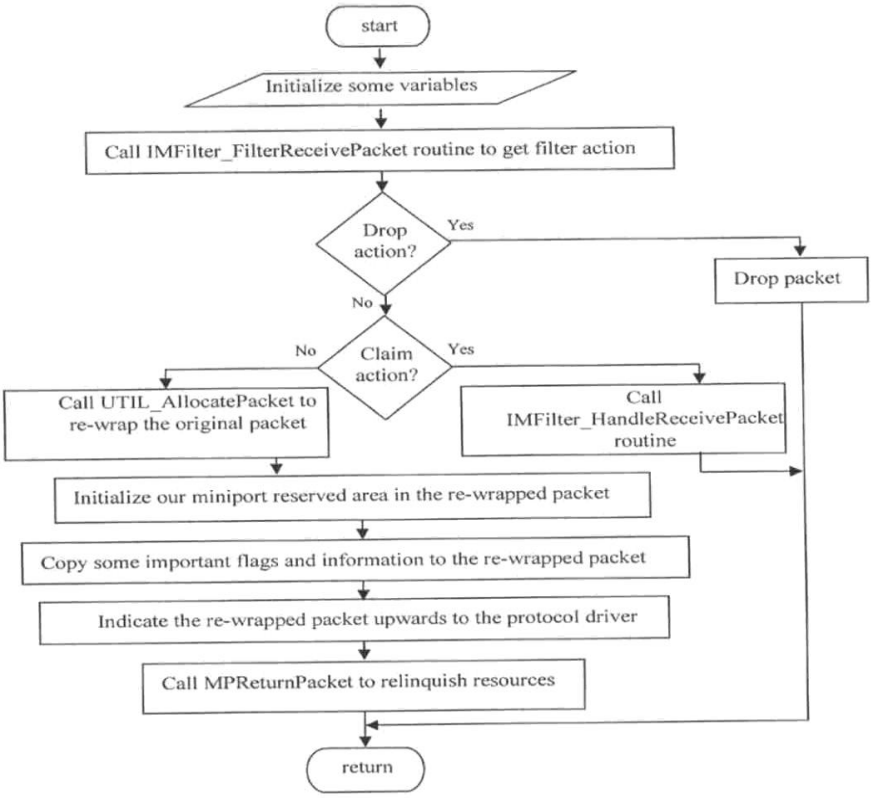

Fig.6.a. Flowchart of Pt Receivepacket Function.

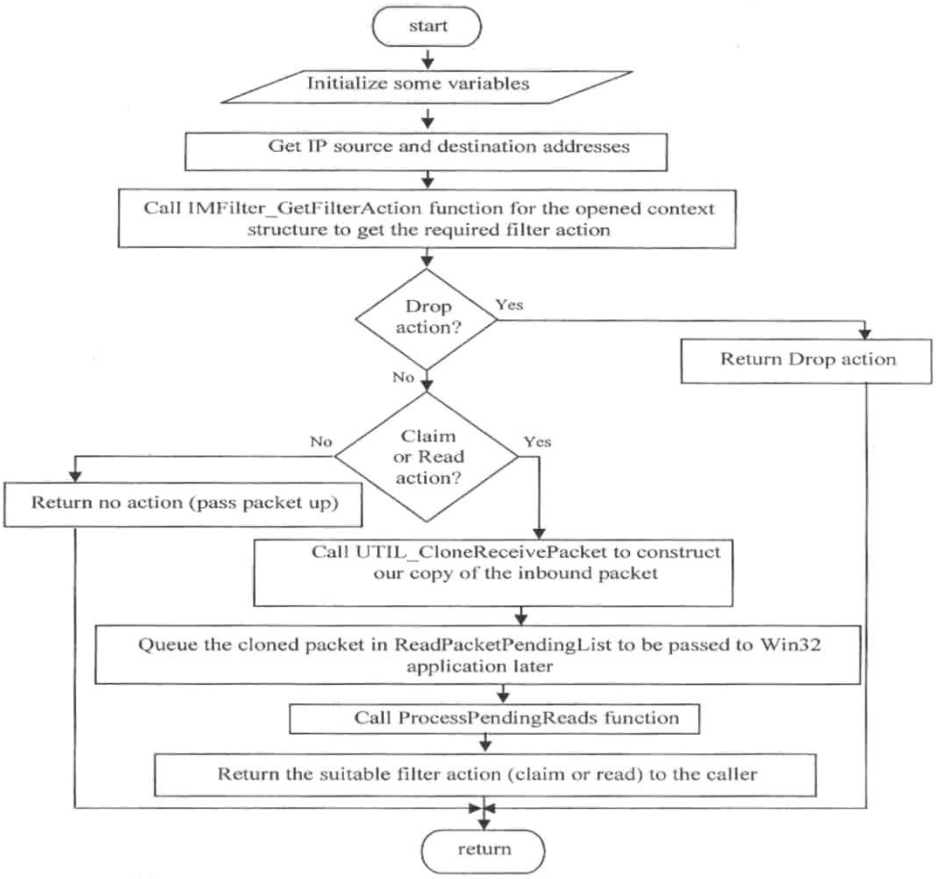

Fig.6.b. Flowchart of IM Filter_Filter Receivepacket Function.

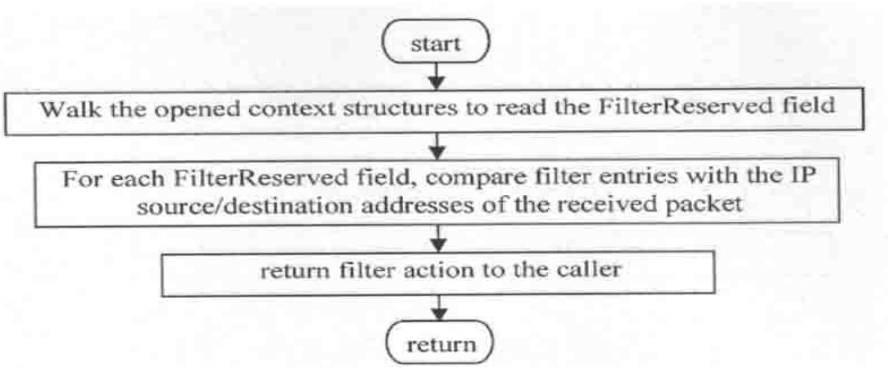

Fig.6.c. Flowchart of IM Filter_Getfilter Action Function.

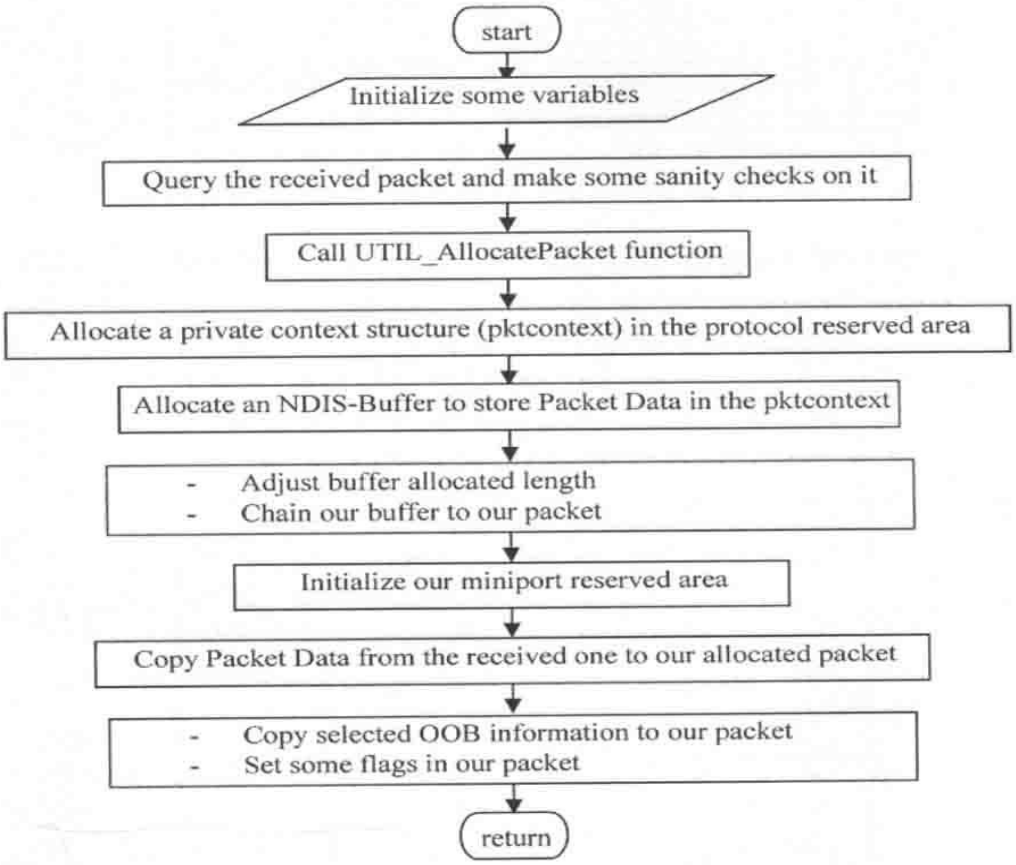

Fig.6.d. Flowchart of UTIL_Clone Receivepacket Function.

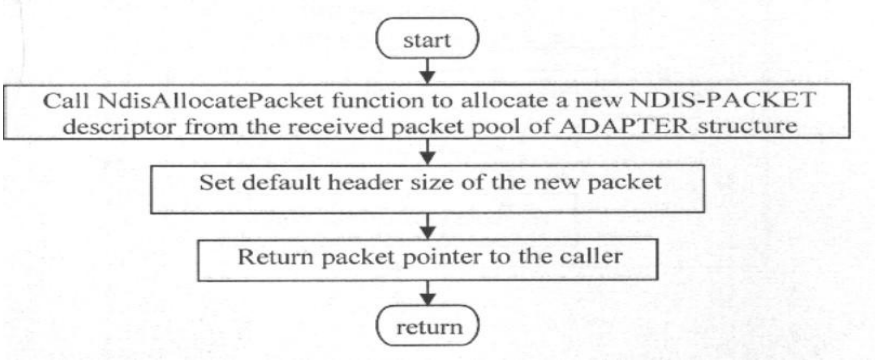

Fig.6.e. Flowchart of UTIL_Allocatepacket Function.

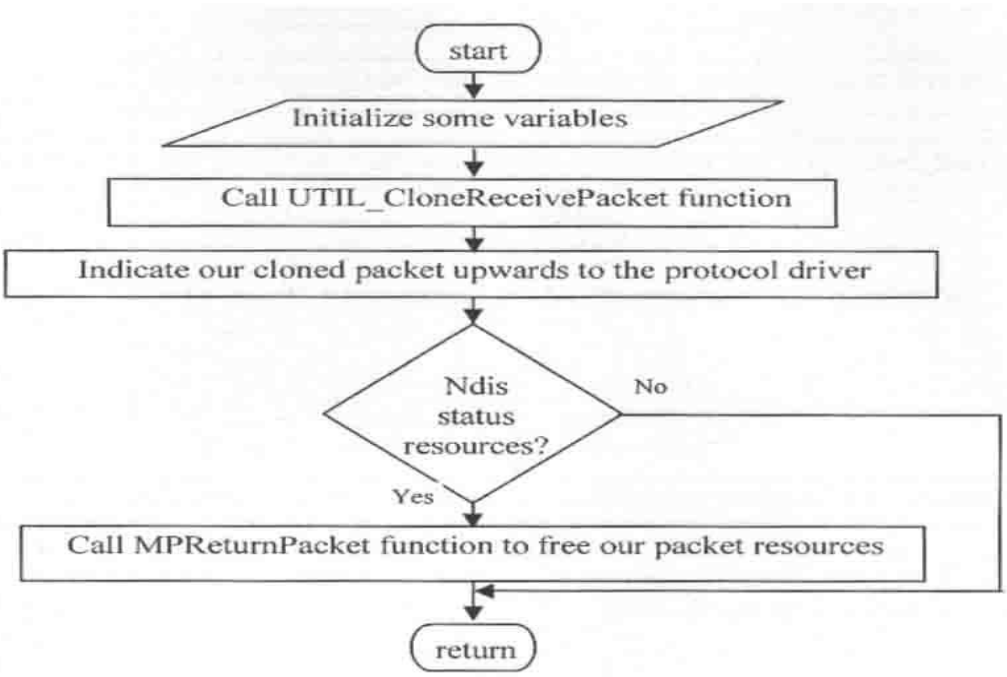

Fig.6.f. Flowchart of IM Filter_ Handle Receivepacket Function.

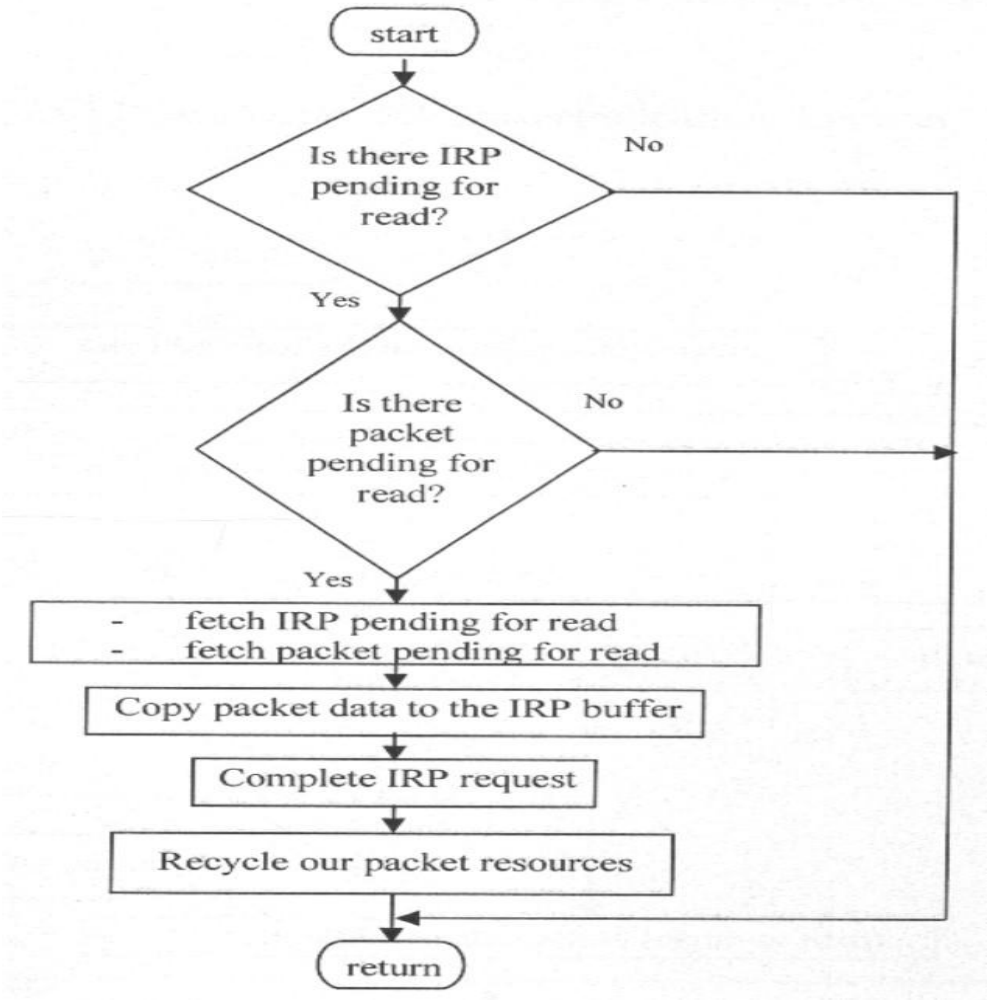

Fig.6.g. Flowchart of Process Pending Reads Function.

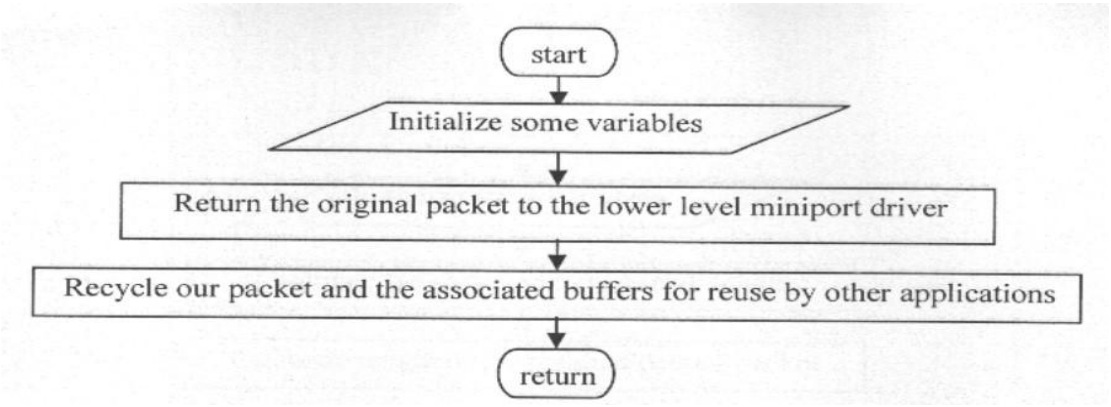

Fig.6.h. Flowchart of MP Return Packet Function.

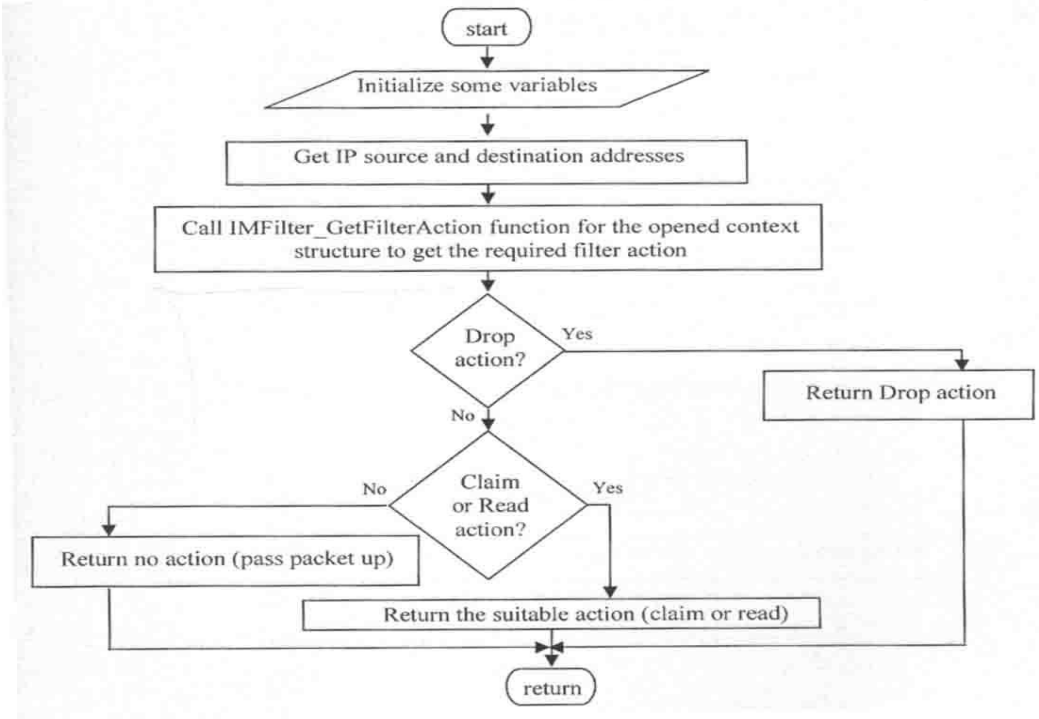

Fig.6.i. Flowchart of Pt Receive Indication Function.

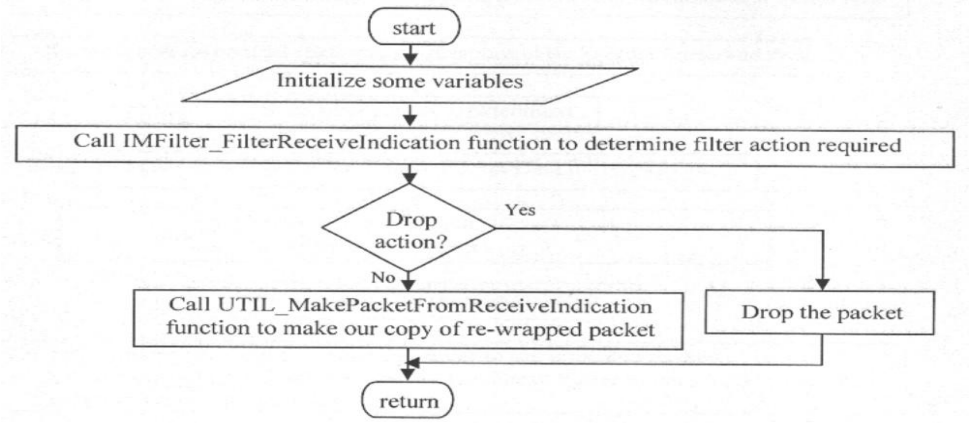

Fig.6.j. Flowchart of IM Filter_ filter Receive Indication Function.

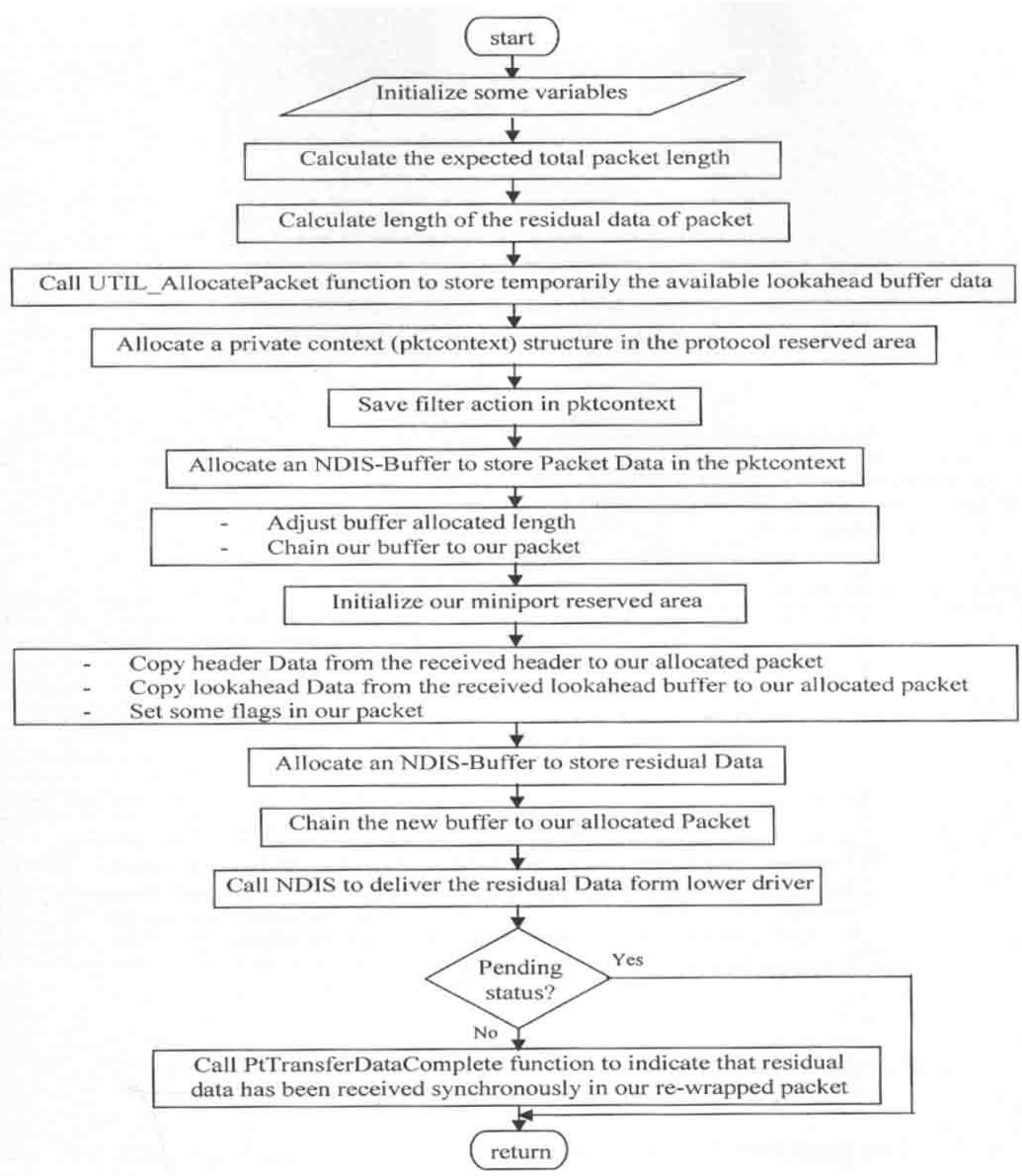

Fig.6.k. Flowchart of UTIL_Make Packet From Receive Indication Functions.

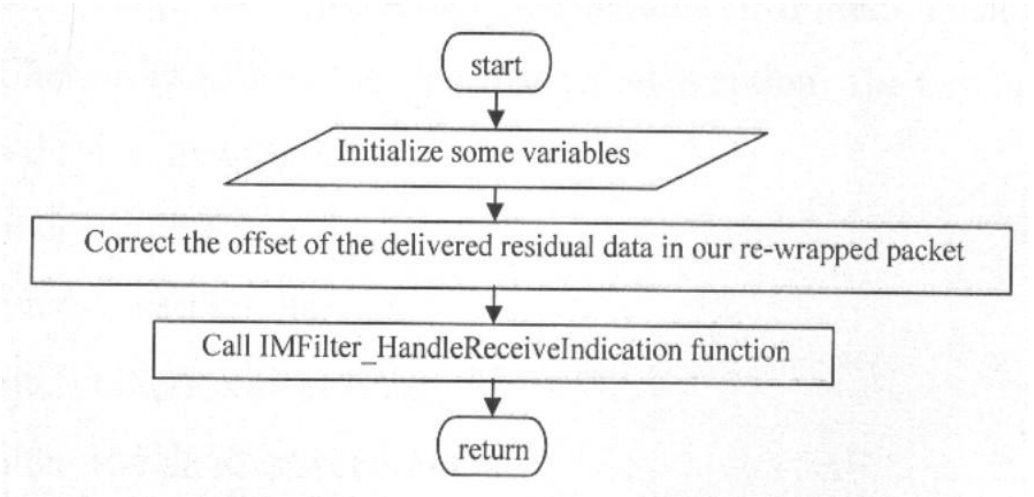

Fig.6.l. Flowchart of Pt Transfer Data Complete Function.

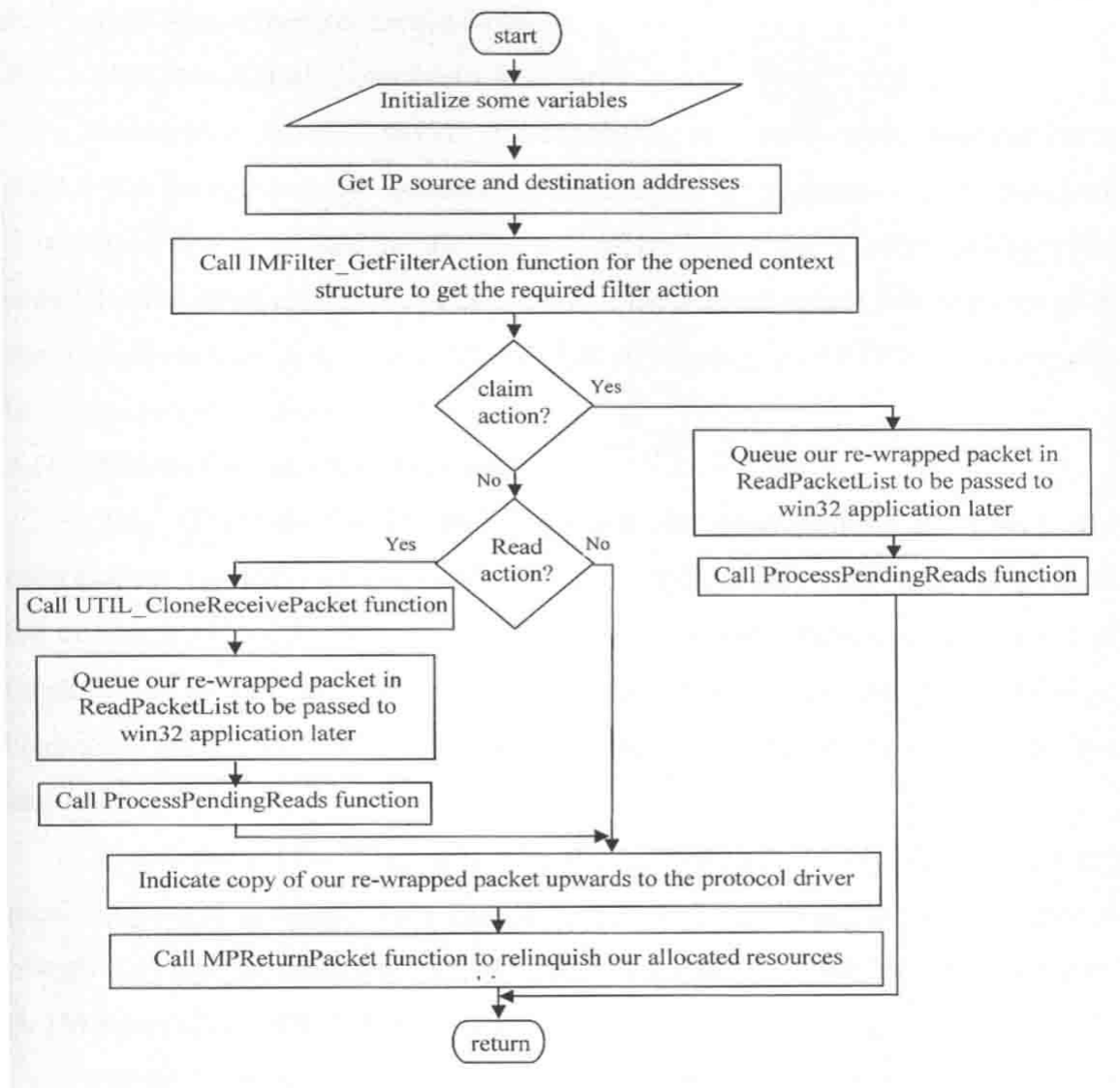

Fig.6.m. Flowchart of IM Filter_Handle Receive Indication Function.

6.2.3. PF unit

This section illustrates how to extend the proposed PIJ so that it has the capability to block the received packets according to a pre-defined list of network IP addresses to be blocked. The basic requirements for IP address blocking driver are:

- **Kernel-firewall:** develop a kernel-space software based on the PIJ driver that blocks packets that are received using an IP addresses list.
- **Application-control:** Include a companion Win32 application that controls and read the list of IP addresses to be blocked. To make the matter more explicit, dropping a packet in the IM driver is accomplished by not forwarding it to the upper layer. There is no NDIS API to drop a packet. This notion may clarify some of the constraints imposed due to in–kernel programming.

Generally, the PF provide the capability of a user-mode application to read a list containing the adapter name and the IP addresses to be blocked. The sorted list is passed to the Kernel-mode using the PB unit. The list of IP addresses is duplicate in the Kernel space and then saved in the FilterReserved area of the ADAPTER structure (ADAPTER structure will be explained in section 6.2.4).

The starting point of this PF is the PIJ unit. The base code was reorganized by adding a new module (IMFilter.C) that isolates the actual filtering code from the basic packet interception. The key functions provided in the new module are:

A. IMFilter-SetPktFilter function: The IMFilter-

SetPktFilter function is responsible to move the information of filter setting from the user application to the context pool of the currently opened adapter. The filter setting information consists of list of Destination/Source IP address ranges, the action of the filter (Block, Duplicate (or read), and claim). The adapter must be priory opened by the application before setting the filter parameters. Then, the given filter structure that is required to be imposed on the received packet is drawn from the input/output request packet (IRP) pointer. Completing the IRP request is essential to free the resources before returning from the IMFilter-SetPktFilter.

B. IMFilter-ResetPktFilter function: In the other side, the IMFilter-ResetPktFilter function resets what the IMFilter-SetPktFilter has been set. It frees the allocated memory of the recently stored filter structure and nulls the corresponding parameters in the specified open context. Finally the IMFilter-ResetPktFilter will complete the IRP request and feedback the suitable status of the processing.

C. IMFilter-Filter Receive Packet, IMFilter-Handle Receive Packet, IMFilter-Filter Receive Indication, IMFilter-Handle Receive Indication: Jobs of theses four are illustrated previously if figs. 6.b, f, i and m, respectively. Generally, IMFilter-Filter Receive Packet and IMFilter- Filter Receive Indication functions are responsible to find the required filter action for the received packet. Whereas, IMFilter-Handle Receive Packet and IMFilter-Handle Receive Indication functions handle the received packet according to the found filter action.

6.2.4. PB Unit

The DEVMJFCN.C and WDMSUP.C are modules in the kernel side that encapsulates the routines related to the IOCTL interface. In user mode side, the modules are realized in C language and then converted to "dll" file such that they export a ready to use APIs for the Win32 applications of the AR. These APIs involve the following: Open Lower Adapter API, Open Virtual Adapter API, Open Lower Adapter By Link Address API, Open Virtual Adapter By Link Address API, Read On Adapter API, Write On Adapter API, Set PKT Filter API, Reset PKT Filter API. In addition to auxiliary APIs that are offered to facilitate Opening an adapter and querying information, these are: Enumerate Bindings API, Make Ndis Request API, Make Private Request API.

In this paper, the IRP-based interface is used to implement the user/driver programming interface. In a consequence, applications can use the basic Win32 functions; Creat File, Devicelo Control, Read File, Write File; and Close Handle in the user-mode side of the interface.

In the Main module (section 6.2.1), the driver creates a device object and a Win32-visible symbolic link name that can be opened in user-mode using CreatFile. Also, at Main Module initialization, the driver was registered IRP-based functions that are (eventually) called to implement the kernel-mode end of the interface. The foundation of the device I/O control is accomplished by calling the Ndis Register Device API from the WDM Initialize function in the Main .C module.

Using Creat File API, the user mode is able to open and close a general device handles on the IM driver file space name. A handle of the IM file object is an ordinary handle to the device that is not associated with a specific adapter binding. This device handle "or control channel" is used to access global information such as drivers binding list but not beyond that. Nevertheless, this handle is not sufficient to make interface with the driver. Knowing that, any IM driver may be connected to more than one MP NIC driver and also more than one overlying PT driver.

The connection between the IM driver and each one of the underlying MPs is represented by a certain Adapter structure; we were called (ADAPTER). Each ADAPTER structure establishes a driver-specific context area between the IM and one of the specified MP drivers. Most information and resources used by the PM units such as packet pool handles, buffer pool handles, and locks are kept in this structure. Several members of an ADAPTER structure is shown in fig. 7.

```
// Per Adapter control block
typedef struct _ADAPTER
{
  LIST_ENTRY          Linkage;
  ULONG               RefCount;
  ULONG               nAdapterStructSize;
NDIS_HANDLE           LowerMPHandle;        // To the lower miniport
          .
          .
          .
NDIS_STRING VirtualAdapterName;
NDIS_STRING LowerAdapterName;
          .
          .
   NDIS_HANDLE        SendPacketPoolHandle;
          .
  NDIS_HANDLE         RecvPacketPoolHandle;
          .
  NDIS_HANDLE         BufferPoolHandle;
          .
          .
NDIS_MEDIUM           VirtualAdapterMediaType;
NDIS_MEDIUM           LowerAdapterMediaType;
          .
          .
UCHAR     AdapterAddress[ETH_LENGTH_OF_ADDRESS];
          .
          .
// Packet Size Information From Lower-Level Driver
  ULONG   HeaderSize;
  ULONG   FrameSize;  // doesn't include the header
  ULONG   TotalSize;
IM_ADAPTER_STATS  Stats;
// List Of Open Win32 Handles On The Adapter
  LIST_ENTRY    W32NOpenList;
          .
          .
  // Fields For Handling IM-Initiated (Local) Requests
  // ------------------------------------------------
  // These fields are used to wrap a request that is initiated locally
  // from the NDIS IM driver itself. This method requires the IM driver
  // to serialize its private NDIS requests.
        NDIS_REQUEST                IMRequest;
        PULONG                      IMBytesNeeded;
        PULONG                      IMBytesReadOrWritten;
          .
          .
NDIS_SPIN_LOCK          Lock;
          .
          .
}ADAPTER,*PADAPTER;
```

Fig.7. Several ADAPTER Struct Members.

6.2.5. PC Unit

At this point, a copy of a complete, received, filtered packet has been gained and queued in the Pending Read Packet List in the OPEN-CONTEXT structure.

However, because we use the standard IP header in implementing the AR architecture, we must discriminate the IP which carry an ANEP header from that which carries non-ANEP content (i.e. discriminate active from non active packets). We were proposed to use a special value in the protocol field of the IP header to refer to the ADP (which contains ANEP header). In such case, all the required to do by the PC module is to inspect the value of the protocol field in the IP header of the received packet. If it refers to ADP, the packet will be passed to the PD unit.The non-ADP value of protocol field represents either a traditional data packet (TDP) or a component packet (CP) types, hence the PC re-inject (write) these packets to the virtual adapter of the PIJ to be lifted to the upper layers for processing.

The PC is realized in a discrete function and we call it a CLSF function. CLSF function takes a pointer to the received packet and its length as input parameters and returns nonzero value if the packet is ADP. However, CLSF function follows the following steps:

- Examine the type of Network layer protocol.
- According to the protocol type, CLSF function can determine which field in the protocol header should be tested to know the packet is an ADP or not.
- According to the suitable header field, if the packet is ADP, return non zero value, else return zero value.

6.2.6. PD Unit

After completing the filtering and classification operations, the PC will deliver only the ADPs to the PD. The PD is a user-mode unit. It is responsible for directing the ADPs to their appropriate UCs. According to the ANEP header of the ADP, the PD can recognize which UC is required to process the new received packet. As has been stated in section 5.3, the ANEP header contains a Component Count (CC) field and a service composition field. These fields tells the PD how many and which UC must be called to process the ADP. The order of these UCs is also determined by the ANEP header.

At first, PD delivers an ADP from the PC and checks the integrity of the enhanced ANEP header. Then, the PD pushes the ADP in a temporary polling cycle to be processed by the appropriate UC. If it is found that one of CIDs does not recognized, the PD will cancel the packet. Jobs of PD unit are implemented as three C++ functions: Serv No, Serv ID, and Get Serv Info. Below is a brief description of each one of them:

A) Serv No Function: The Serv No function is dedicated to determine how many UCs must be called and executed on the received ADP. ServNo, simply, initialize a pointer to the CC field in the enhanced ANEP header and return a copy of CC field value.

B) Serv ID Function: In other side, Serv ID function targets to withdraw the CIDs of the required UCs in a correct order. Using the packet array and taking into account the order of the required UC and the type of protocol headers, a pointer to the correct CID value in the service composition field is initialized. This value (CID) will return to the Execution Environment Manager (EEM) module (see the next section). Actually, the CID is used by EEM to search the cached configuration files which, in turn, used to call the associated UC.

C) Get Serv Info Function: The last support function is the Get Serv Info. It is intended for getting a copy of Serv lnfo structure from the configuration file associated with the UC. CID integer will be inserted as input and Serv lnfo structure is achieved as output from Get Serv Info function.

Below is a brief description of the necessary steps followed by Get Serv Info function:

- Define some useful variables.
- Convert the CID integer into an ASCII string.
- Search and then open the ".ini" file indicated by CID string.
- Read and copy the file contents into a pre-allocated structure of type Serv lnfo. This structure will be exported as output of Get Serv Info function.
- If the above steps work OK, return a non-zero value, otherwise return a zero.

6.3. Active Network Operation

UCs are implemented as dynamic link libraries under windows 2000. Since the AR prototype implementations exploit standard link library technique and execute active code within user-space processes, components can be developed and tested based on standard development tools (for example, the visual studio IDE compiler and debugger). As DCs can be loaded and executed in the form of binary code, the implementation of the components is conceptually independent from any particular programming language. Note that this designed AR doesn't build safety and security upon a specific programming language or certain language constraints (such as strong typing, range checking .. .etc). However, since the APIs must be linked to the component at execution time of DLL, only programming languages for which the system API is available can be used. At the present time, the system APIs of the current AR prototype implementation are only available for C and C++. The software linker between the CD server and the PM in the AR is the EEM. The EEM is a module designed to integrate the two running software units, namely, the IM driver-based and the FTP-based programs. Briefly the EEM performs the following main jobs:

1. Declare two instances of lower and virtual adapters to be used for network access and low-level information.
2. Define and initialize necessary variables.
3. Define two packet arrays (TempPack and OutPack), one for temporary processing and the other for returning the processed packet.
4. -Declare an instance of ServInfo structure. It is used to read and store temporarily the contents of the configuration (.ini) file.
5. Open the lower and virtual adapters, using APIs exported by PB unit (see section 6.2.4).
6. Read the received packet(s) and its length from the lower adapter and store it in TempPack array. If there is no received packet yet, EEM still wait until one is reached to the PendingReadPacketList buffer.
7. Call CLSF function (see section 6.2.5) to check if the received packet is an ADP or not. CLSF function take TempPack array and its length as input parameters. If it is non-active packet, EEM will write the packet directly in the virtual adapter to be processed traditionally by Windows stack. In this case, this packet is either a TDP or a Component Carrying Packet. In the case of TDP, the burden of packet-routing and forwarding is

moved to the shoulder of windows protocol layer. Whereas, if it is a component carrying packet, windows stack will raise it to the CD server to be dealt with there. In other side, if the packet is an ADP then, perform the following steps.

8. Declare a general handle to a dll file which is expected to refer to the required UC.
9. Call the PD unit. PD will perform the following steps:
   a) Call ServNo function to get how many UCs are required to process the ADP. According to the count of the UCs, steps 9.b up to 12 must be repeated.
   b) Call ServID function to get the suitable CID from the composition service field in the Enhanced ANEP header.
   c) Using the CID, the cached configuration files (associated with UCs) are searched to get the correct Servlnfo structure. GetServInfo function is called to accommodate this job. Servlnfo structure contains the name of the UC's dll file whose its CID is given as input to GetServInfo function.
10. Call the Component Loader (CL) function. The CL of the EE is responsible for loading UCs into memory, and to initialize and start them. The CL is initiated by the EEM. However, steps followed by the designed CL are:
    a) Load the ProcPacket function from the DLL library into memory to be executed. Calling LoadLibrary API will attempt to locate the DLL and then perform such loading. The code file name (i.e. DLL name) of the UC is given as input parameter to LoadLibrary API. LoadLibrary Windows API maps the specified executable module (dll) into the address space of the calling process. The returned value will be a specific handle to the required DLL library that is loaded in the memory. Remembering that a general handle to DLL file has been declared in step 8. This handle is assigned now to the handle dedicated to the useful UC returned from LoadLibrary.
    b) Given the dedicated DLL handle and the name of the **exported** function (ProcPacket) as inputs to the GetProcAddress API, a pointer to the specified exported DLL function (i.e. ProcPacket function) is returned. ProcPacket is the Template name of the UC function which is required to process the received packet. The function name given to GetProcAddress API must be identical to that in the EXPORTS statement in the DLL's definition (.def) file.
11. To execute ProcPacket function, TempPack array, its length, (a control buffer and its length if required) is passed as input parameters, and then enforce the ProcPacket function. The actively processed packet is returned in the OutPacket array.
12. Decrement the count of UCs that has been read in step 9.a. If there are more components must be executed, repeat steps from 9.b up to this step. If component count reaches zero move to step 13.
13. Using the developed WriteOnAdapter API, push the processed packet again to the windows stack through writing on the opened virtual adapter to be routed and forwarded conventionally there.
14. As long as there is no manual EEM exit instruction, the program continued in step 6. If there is an exit EEM instruction, close the lower and virtual adapter instances and exit program. The two support APIs (LoadLibrary and GetProcAddress) are provided by Microsoft Visual C++6.

7. Evaluation and Results

This section evaluates the designed AR qualitatively and quantitatively.

7.1. Packet Generator/Injector

In this paper, an external program is proposed and implemented which creates and then injects data packets into the network stack of the testing machine. We will call this program a Packet Generator/Injector (PGI). The generated packets involve ADPs and TDPs. In each testing course the testing machine which runs the PGI software is connected to the AR (to be tested) or even the ES(s).

Several ready-to-use Packet Generators/Injectors are available as executable software in the internet [29] [30]. All the offered programs use Windows Socket (WinSock) [26] capability to inject user defined packets. The use of WinSock involves encapsulating the injected headers (Ethernet, IP ... etc) within the Transport layer (TCP, for example). This method obliges users to insert virtual headers within the real headers. Furthermore, such technique decreases the MTU of the original protocols due to the virtual headers overhead.

Due to the above reasons, a novel PGI is designed and implemented. It uses only the original protocol headers with full control on filling their fields. Also, the designed PGI pushes

the generated packets directly to the IM driver in the network stack. PGI exploits the implemented PM units. It uses the PIJ and PB units to inject our created packets directly into the lower adapter. Fig. 8 shows a simple block diagram for the designed PGI.

A new function in user-space is developed to facilitate creating and injecting data packets by AN users in the ES. However, it is intended mainly for AN developers and not AN customers. The function is implemented as stand alone module called PGI.C.

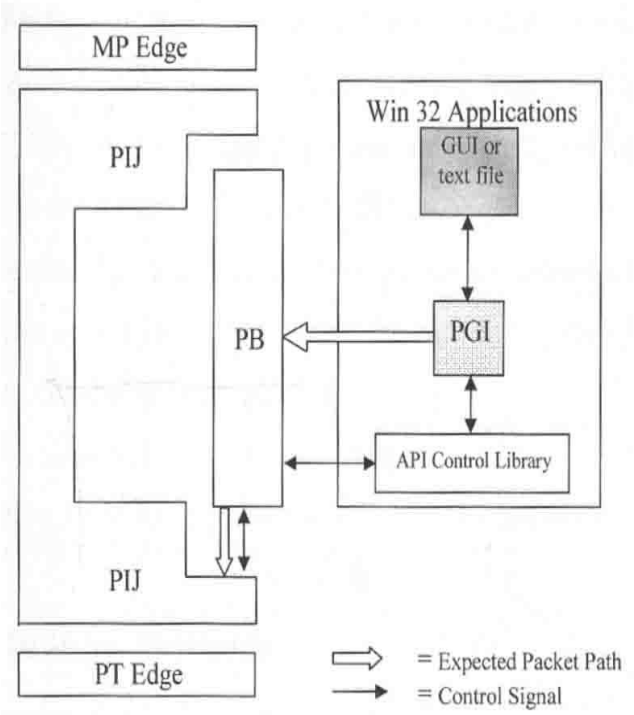

Fig.8. Illustration of PGI Units

7.2. Evaluation Methods

In general there are two methods used for the evaluation of system such as that presented within this paper, namely qualitative and quantitative evaluation [2]. Since the main contribution of this paper is the architecture for active routers, a qualitative evaluation of the concepts and design of the AR architecture has been regarded as more meaningful. Since it has been feasible to implement only a subset of the overall AR architecture, a quantitative evaluation of the entire system (with applications) cannot be provided at this stage. Nevertheless, section 7.4 provides a quantitative evaluation of the key units.

7.3. Qualitative Evaluation

Issues regarding the design and concepts behind the AR programmability are considered here. In order to evaluate the AR architecture, an example case study is introduced.

7.3.1. Case Study

A) <u>System Setup:</u> As illustrated in fig. 9, set up assume existence of an establishment (Company, Bank, Library, Campus ... etc) divided into headquarter and other branches distributed around wide area, such that they exchange information through the internet. Headquarter may contain multiple networks (LANs or/and WANs) such that there is a network for each department or section.

It is envisioned that ARs will form the core elements (access and gateway) of this network. Administrator (ADMN) of the network was decided to operate under TCP/IP suite and running the ARs under Windows 2000 server.

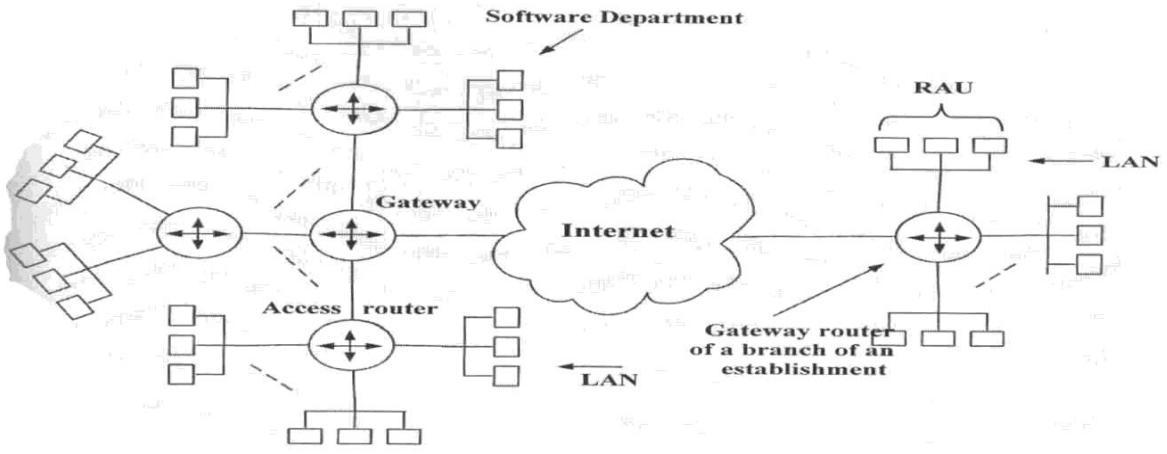

Fig.9. Hypothetical Diagram of Computer Network of an Establishment.

B) The Challenge: The key challenge of the setting is to "open up" the establishment network and to provide public services (through internet) to remote authorized users (RAUs). RAUs may belong to the establishment staff or external customers. It is required to provide sufficient confidentiality for the transferred information from RAUs to the establishment (headquarter or one of the branches) services. It is decided to provide security by the developed encryption algorithm called "RC5" [31].

C) The Solution (Network-Level Encryption): The focus here lies not on the solution itself, as the concepts behind the solution are not specific to ANs and therefore can be realized otherwise, but on the fact that the AR provides a generic platform that is sufficiently flexible to resolve this problem.

To enable remote RAU to send information to a specific section (headquarter or branch) in the establishment, ADP are encrypted in the ES of RAU side. The ANEP pay load is encrypted using RC5 algorithm [31]. Since the transferred information targets one of the multiple sections of the underlying establishment, all gateway routers (which are active) must be prepared to decrypt the arrived ADPs.

The security system can be realized as follows: The software department in the establishment installs RC5 decryption component in the gateway ARs which are belong to the establishment. The installed UC (decryption component) has its own CID. RAU who wish to benefit from the services introduced by any deterministic section in the establishment must gain the CID and the secret key (user's-specified key). The CID can be gained from publicly shared specifications of the running ARs which are distributed by AN community. The secret key is distributed securely (manually or through network) by the software department of the establishment to their authorized customers.

D) Prototype Realization: For prototype implementation, the basic setting of case study is minimized into only three computers, namely "A", "B" and "C", as shown in fig. 10. Computer "B" realizes the gateway active router whereas computer "A" realizes two entities. Before any information exchange, computer "A" will represent the PES; actually it is the software department of the establishment which will create and install the UC (the RC5 decryption component). After installing the UC, the function of this computer will be replaced to represent one of the RAUs who will send encrypted ADPs to the establishment (computer "C"), through the gateway AR (computer "B"). The three computers have their own IP addresses and also their own NIC cards. A unique MAC address is assigned to each one. NIC cards used in this prototypical setting are of type SURECOM 10/100M PCI adapter and uses Fast Ethernet Protocol. Fig. 11 shows a picture for the configured computers of the case study in the laboratory.

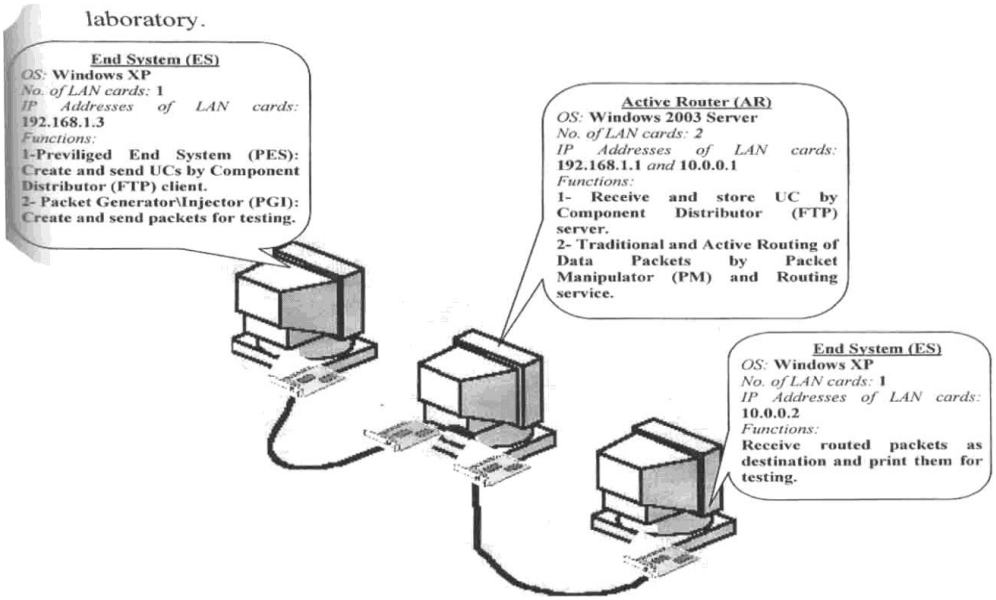

Fig.10. Setup of Prototype Realization of Case Study.

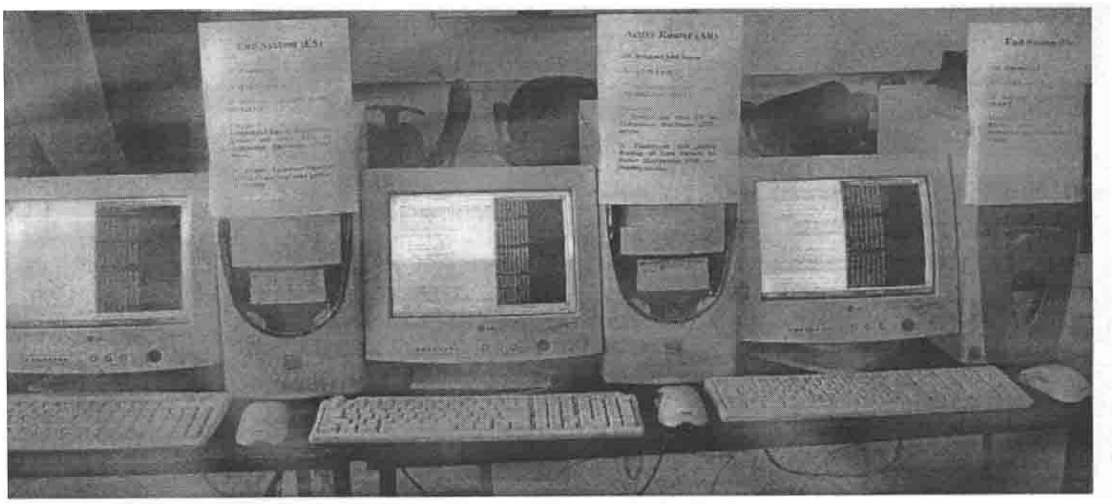

Fig.11. Configuration of Case Study in the Laboratory.

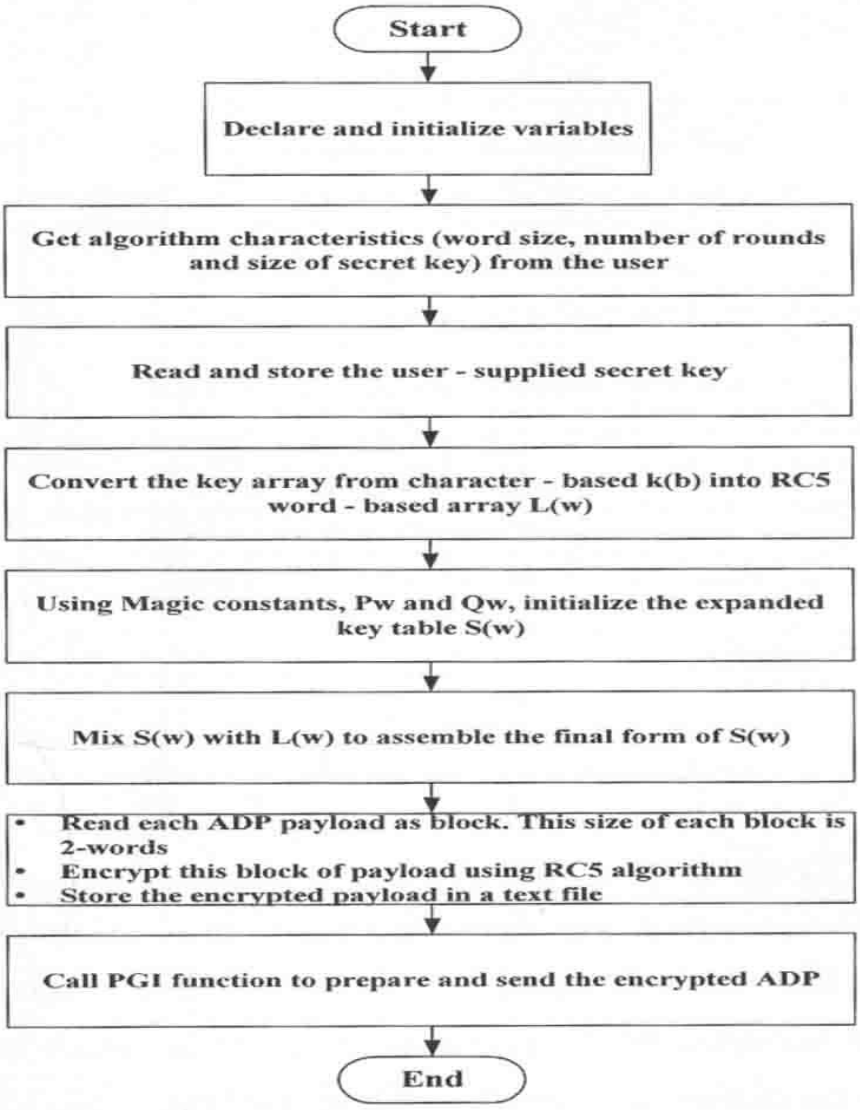

Fig.12. Flowchart of RC5 Encrypting and Sending of ADPs.

In this proof-of-concept case study, RAU realizes the RC5 encryption algorithm as a single module (RC5.C) using VC++6 environment. A general flowchart of RC5 module realization is shown in fig. 12. As shown in the flowchart, after performing RC5 encryption of ADP payload, the RAU exploits the previously explained PGI unit to create and then send the packet. The flowchart addresses encrypting and sending of only one ADP; however, this routine can be repeated as many as required to proof the concept.

Software department enforce accurately steps mentioned in section 6.1.3 to implement the two files of the UC of the RC5 decryption algorithm. C++ language is used in programming the code and configuration files. Concerning the control buffer in ServInfo structure, it is exploited to store the users' supplied secret key. The flowcharts shown in figs. 13, 14, and 15 are demonstrate the steps of realizing the RC5 decryption component in the PES.

In the AR computer (gateway), the pre-installed CD server receive the UC (RC5 decryption function) sent by software department of the establishment. CD server installs the two files (code and configuration) in a pre-defined folder in the AR. However, EEM continue running the Packet Manipulator for searching whether there is a new received packet or not. When the RAU begins send the RC5 encrypted ADPs, EEM will perform the required processing (decryption). The word size and number of words are fixed to (32 and 16) respectively, in the implemented decryption component.

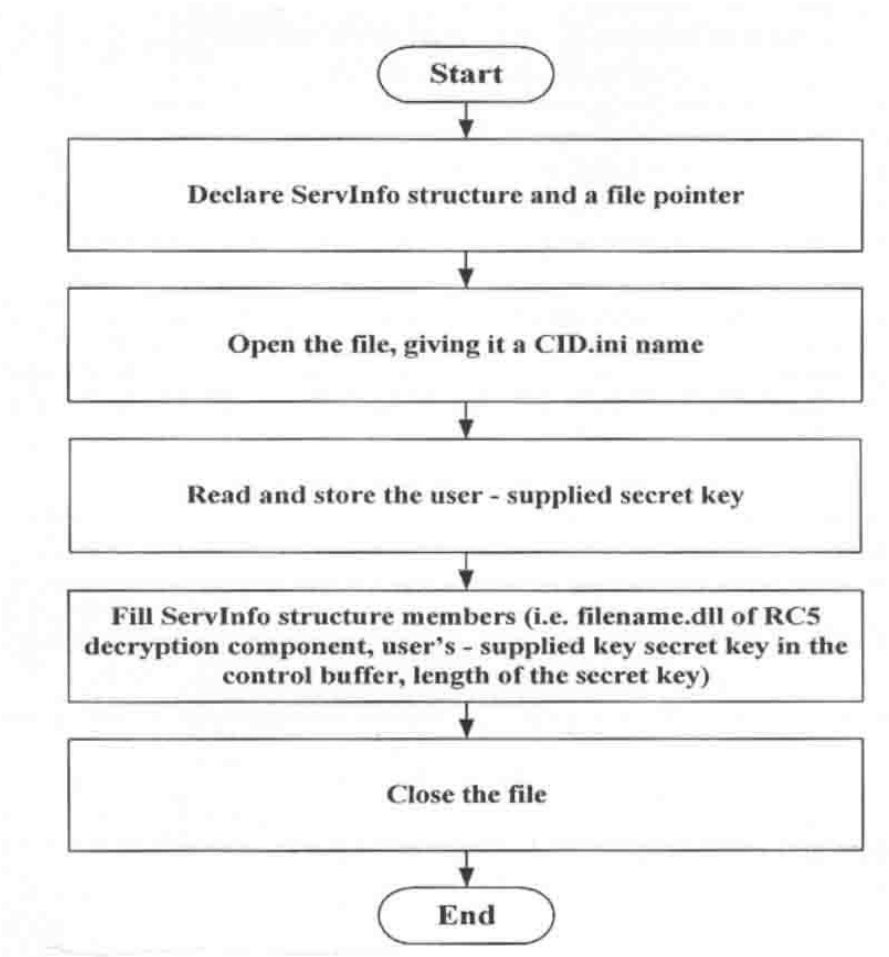

Fig.13. Creating the Configuration File.

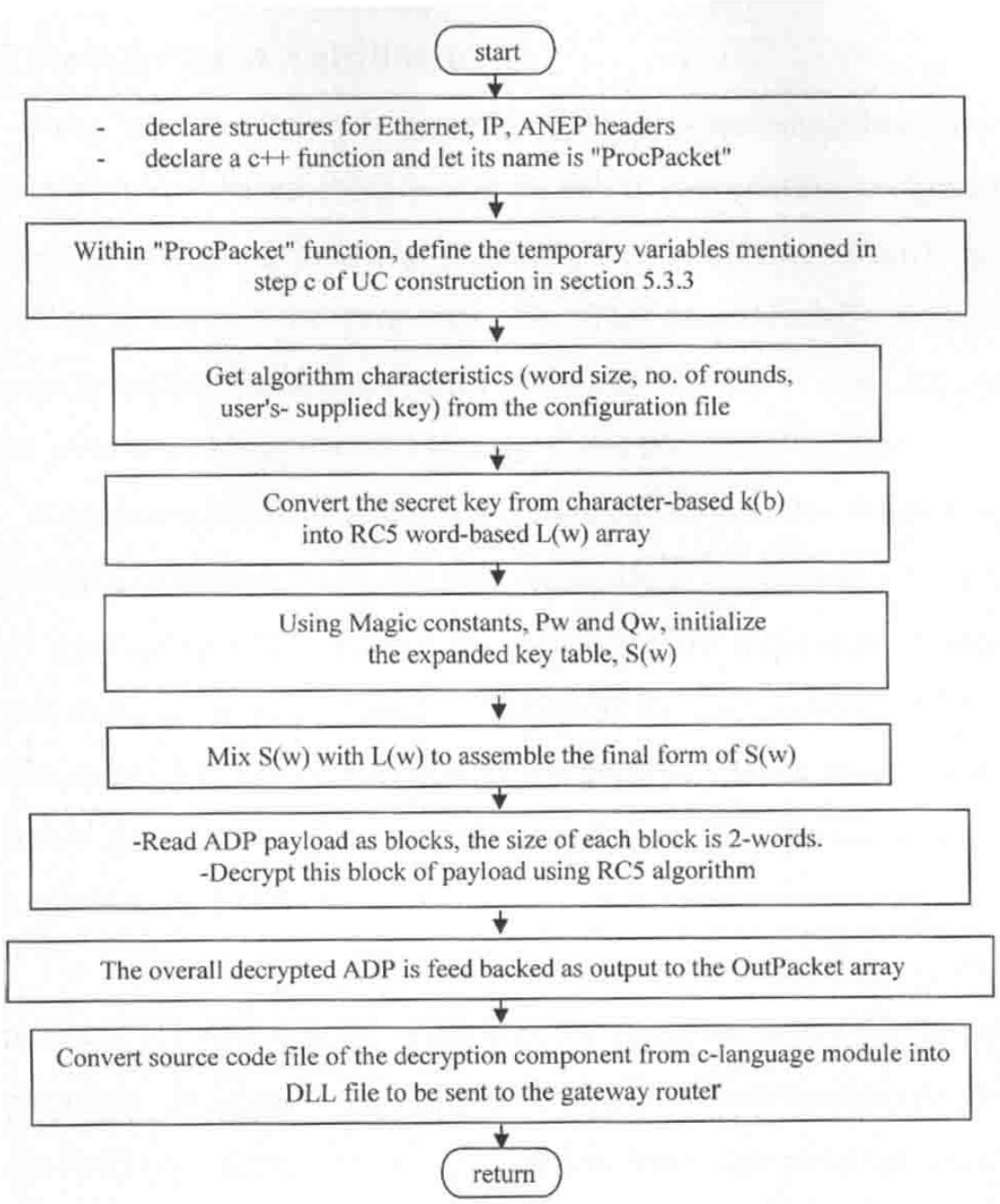

Fig.14. Steps of Construction of RC5 Decryption Component.

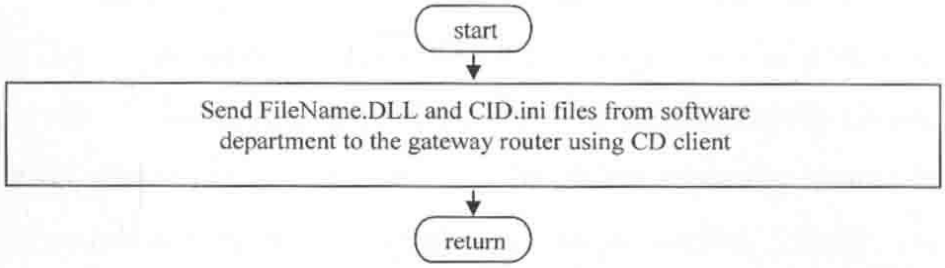

Fig.15. Final Step in Programming the AR.

7.3.2. Requirement Fulfillment

Requirement 1, namely programmability, is accomplished through the addition of ANEP-based composition model. In this context, programmability is merely a matter of inserting (or removing) UCs into (from) the packet processing chain on the proposed AR. This is essentially done through loading (or unloading) component's code into the EE (virtual memory) and inserting (or removing) it's CID into (or from) the ANEP header.

Requirement 2, demands flexible router functionality. While support for data plane programmability through transparent integration of UCs in the packet processing chain makes the AR a highly extensible platform, the dynamic composition framework proposed by the architecture provides a very flexible means for that. In addition, it is a highly dynamic process that can be efficiently carried out at run time without the need for restarting or even reconfiguring the system. The

Requirement 3, safety for active code execution within the designed AR is achieved through software fault isolation.

Requirement 4, secure programmability. Simple firewall is achieved by PF unit in PM. In addition, user authentication is accomplished using basic authentication of IIS server [24] provided by Windows. Nevertheless, this is only simple encoded user-name and password authentication. AN demands more strong encrypted authentication. More secure system can be considered as a future work for this paper.

Requirement 5, is to achieve router performance close to the line speed of typical edge networks. Later in this section (section 7.4), a performance estimate of our prototype implementation, shows that the AR has not fulfill this requirement completely. From the architectural point of view, there is certainly tradeoff between modularity and performance. Since the AR tries to maximize flexibility through the concept of (de-) composition of active services, it trades off performance. For example, ADPs traversing the AR must be passed to all the components that indicated the interest in the packet. Furthermore, the AR architecture has been carefully designed to maximize the performance where possible. Because of this, the idea of application-level active-networking has been rejected, and a true network-level approach has been employed instead. For the same reason, AR enables the safe execution of efficient binary code rather than having to rely on code interpretation.

Requirement 6, which demands easy usability, must be considered from the point of view of the end-users and developers. In the case of our particular implementation of the AR, the development task is greatly facilitated by the design decision to execute active code within the user-space EE, which allows convenient development of user-space code.

Requirement 7, scalable manageability is not addressed in the work throughout this paper.

Table 3 lists the relevant requirements and indicates to what extent have been satisfied in the project.

Table 3,
Designed AR Compliance with Relevant AN Requirements

Requirement	Description	Satisfied?
1	Programmability	yes
2	Flexibility	yes
3	Safety	yes
4	Security	partially
5	Adequate Performance	partially
6	Easy Usability	partially
7	Scalable Manageability	No

7.4. Quantitative Evaluation

To evaluate the designed AR architecture and hence the performance of the envisaged AN, three types of tests were done; Control Test, AN Test, and Backward Compatibility Test. Three personal computers (we call them “A”, “B”, and “C”) are used to accommodate the experimental setup.

One of them (computer “B”) is operate as traditional router (in control test) and as an AR (in the other two tests). The other two computers (“A” and “C”) are used as two different LANs. These LAN computers are connected directly to the router (computer “B”) using a UTP cable. Actually, LAN computer represents an ES in a LAN. The NIC cards used is of type SURECOM 10/100MB Fast Ethernet PCI adapter.

The hardware architecture of the prototype AR is based on a standard personal computer consisting of single Intel Pentium 4

microprocessor running at a clock speed of 2GHz, 256 MB of RAM and two 100Mbps Fast Ethernet interface. The two interfaces are connected to the two ESs. Each NIC has its own IP and MAC addresses such that it belongs to one of the connected LANs. The types of the router's interfaces are similar to that of the ESs.

In the three tests, a data of 360 MB size has been transferred from computer "A" to "C", through "B", and all measurements are accomplished at computer "B".

7.4.1. Control Test

The first test carried out was a control test, which just tested the speed of the network, operating system overhead, and network stack overhead without installing the developed AR software.

In control test, computer "B" running Windows 2000 server, and it is configured appropriately to operate as a traditional router. Packets of TDP type are used in this measurement. The resulting average throughput and CPU loading at computer "B" are shown in table 4.

Table 4,
Results of quantitative evaluation tests.

Test Sequence	Test Type		Throughput (Packet/Sec)	% CPU Usage
1	Control Test		5679.8	30.2
2	PM Test		2902.6	78.8
	AN Test	5 UCs Test	914.9	84.1
		10 UCs Test	453.1	86.7
		15 UCs Test	335.8	89.8
3	Backward Compatibility Test		4041.3	75.2

The control test is selected to be a reference gauge, because it represents the actual normal operation of traditional data, router, and protocol. The next two tests, which involve the active enhancement software, will be compared with control test to measure how much degradation and processing cost is paid for active programmability.

7.4.2. Active Network Test

In this test, all the implemented software, which has been explained in section 6, is installed in computer B which still running Windows 2000 server. The object of this test is to gauge throughput and CPU usage required to achieve the active programmability of the router. AN test involves two parts, the first one evaluates the cost of PM and the second measures how many UCs can be processed with keeping reasonable throughput and consuming acceptable CPU time. The developed PGI software is used in computer "A" to construct and send the required ADPs in each part of AN tests.

A) PM Test: In the first part of AN test, the measurements try to quantify the, processing speed and load needed in PM units (namely, PIJ, PF, PB, PC and PD). The Component Loader (CL) is suppressed temporarily from the EEM to accomplish this target. The resultant throughput and %CPU usage is tabulated in table 4.

B) UC Test: This test aims to check how many UCs can be used in each service such that the AR is not saturated. However, the evaluation in this paper is directed to evaluate the architecture only and not tied to a specific service or application. Hence, applying a specific UCs to implement a certain application or protocol will not fulfils the target of this evaluation, as it is intended to specialized application.

Five, ten, and fifteen user components are used in three measurement steps. The resultant averages are also shown in table 4.

7.4.3. Backward Compatibility Test

In this test, the developed AR software is also installed in computer "B" under Windows 2000 server. A stream of TDPs (IP packets) is sent from computer "A" toward computer C through the AR. The purpose of this test is to ensure the backward compatibility to the current traditional routers and check the quantity of degradation in throughput and the cost in processing time when AR is used to route TDPs rather than traditional router. The results are listed in table 4, too.

7.4.4. Discussion

The relative high throughput and low % CPU usage in the first experiment, control test, represent the capability of the three computers (namely "A", "B", and "C") during normal

networking operation without effects of the driver. It reflects the behavior of in-kernel processing of traditional routing of packets.

The first part of AN test, namely PM test, gives expected result when it takes approximately double processing time and half throughput with respect to the control test. We can argue that this change is the price that must be paid to transport to the user-level processing of the PM stage (i.e. PB, PC and PD). Because of the PIJ and PF units are operate within the kernel of the router, their effects are negligible. It is agreeable [24] to state that kernel mode processing is faster than the user mode one.

Concerning the second part of AN test (i.e. UC test), the values of measurements refer to the capability of dealing with about 5 user components, acceptably, for each ADP. But tests of 10 and 15 UCs show high throughput degradation accompanied by consuming of most processing power. Practically, it is also possible to conclude that the system in its current state (experimental) is not extensible when more than 15 UCs are added to the path of ADPs. This doesn't really affect the validity of the system. It can be solved, for example, by pulling the PC and PD units down to the kernel space which will lead to two times increase in throughput. Using multi-processor router (as in current traditional routers) or upgrading router's hardware by field programmable gate array (FPGA) technology may also contribute in solving this shortcoming.

The last test "backward compatibility test" is succeeded in proving the compatibility with the current available traditional routers in two aspects. From one side, the AR routes and forwards the TDPs (IP packets) correctly without any error or side effects. From the other side, it is not severely affects the standard throughput obtained in control test. The relative little degradation in throughput is due to the overhead of PM units (except PD unit) with paying double processing time.

8. Conclusions and Future Works

8.1. Conclusions

Several conclusions about the development of AR architecture can be drawn from this work:

- Component-based active router architecture enables network programmability through extensibility of router functionality and services.
- Active network programmability demands sever safety mechanisms to protect the nodes from malicious or erroneous active code.
- Reuse of the standard 'process technology' of today's operating systems as safe execution environments for active code has proven to be very practical.
- A split implementation across both kernel and user space of the underlying system appears to be a good choice. This approach takes advantage of the high flexibly programming environment in user-mode and sophisticated protection and safety mechanisms of today's OSs.
- Standard user-space implementations for active networks typically suffer largely from the performance hit resulting from the copy operations required to pass the network traffic "up" into user-space and back "down" again. As far as possible, packet processing must be in kernel space.

8.2. Future Works

Future work presented in this section focuses on the ongoing development efforts to complete the AR prototype implementation and on using and extending it in order to build and experiment with novel AN services:

a) The use of poor security scheme is considered one of the shortcomings of this paper. To address this disadvantage, authentication within the AR may be built on public key encryption mechanisms such as RSA or DSA. The user installing a component (or code producer developing a component) encrypts its identity (i.e. user name or company name) with its private key. Public key encryption ensures that the encrypted message can only be decrypted with the public key assigned to the user.

b) Demand-push is the base at which the developed AR depends to install a new UC. To lift the burden from the user's shoulder, it is more pragmatic to distribute a number of component cache servers throughout the AN. All that required by the AN users is to instruct the suitable UC to be downloaded from the server to the target router. For example, when ADP arrives to the AR, the CID will excite a certain program in the AR to download the required UC from the nearest server. This scheme is called a demand-pull mechanism in transferring UCs.

c) Further decrease in throughput and increase in CPU usage is envisaged with each excess in the number of networks or nodes attached to the designed AR. Hence, improvement in AN performance still crucial. To decrease the effects of this issue, it is possible to pull the PC and PD down to the kernel level.

d) The designed AR in this paper demands processing only one active packet at a time. To get more efficient use of processing resources and Windows OS capabilities, it is useful to implement multi-packet processing in the EEM. Multi-Packet processing demands allocating a dedicated EE for each packet or uses one EE with some form of preemptive scheduling mechanism that allows the low-level system to interrupt active programs that exceed their scheduling quantum.

9. References

[1] DARPA Agency, "Active Network project", 1998. Available at: http://www.darpa.mil/ito/research/anets/

[2] D. J. Wetherall, "Service Introduction in an Active Network", PhD thesis, Department of Electrical Engineering and Computer Science,Massachusetts Institute of Technology, USA, 1999.

[3] S. Schmid, "LARA++ Design Specification", Work in progress report on the next generation active router architecture of Lancaster University, Computing Department, Lancaster University, UK, 2000.

[4] D.S. Alexander et al, "Active Network Encapsulation Protocol (ANEP)", Internet draft, IETF, July 1997.

[5] A. Fuggetta, G.P Picco, and G. Vigna, "Understanding Code Mobility", IEEE Trans. on Software Engineering, 24(5):342-361, May 1998. Available at: http://www.polito.it/~picco/listpub.html.

[6] D. J. Wetherall and D. L. Tennenhouse, "The ACTIVE IP option", In 7th ACM SIGOPS European Workshop, Ireland, September 1996. www.tns.lcs.mit.edu/publications/sigops/ws.html.

[7] K. Psounis, "Active networks: Applications, Security, Safety, and Architectures", IEEE Communications Surveys, First Quarter, 1999. Available at: www.comsoc.org/pubs/surveys.

[8] D. J. Wetherall, J. Guttag and D. L. Tennenhouse, "ANTS: A toolkit for building and dynamically deploying network protocols," IEEE OPENARCH, pp. 117-129, April 1998.

[9] B. Schwartz, W. Zhou, A. W. Jackson, W. T. Strayer and D. Rockwell, "Smart Packets for Active Networks.", In 2nd Conf. on Open Architectures and Network Programming, OPENARCH'99, NY, Mar.1999. Available at: www.ir.bbn.com/~bschwart.

[10] B. Schwartz, "Sprocket language description for the Smart Packets project," Technical paper, September 1999. Available at: www.ir.bbn.com/documents/techmemos/TM l 221 .ps.

[11] B. Schwartz, "Introduction to Spanner: assembly language for the Smart Packets project," Technical paper, September 1999. Available at: www.ir. bbn.com\ documents\ techmemos\ TM1220. ps.

[12] D. S. Alexander, W. A. Arbaugh, M. Hicks, P. Kakkar, and J. M. Smith, "The Switch Ware active network architecture," IEEE Network, vol. 12, pp. 29-36, May/June 1998.

[13] M. Hicks, P. Kakkar, J. T. Moore,.C.l A. Gunter, and S. Nettles, "Network Programming Using PLAN", Project supported by DARPA, and University of Pennsylvania, 1997. Available at: www.dsl.cis.upenn. edu.

[14] M. O. Stehr, C. L. Talcott, "Plan in Maude: Specifying an Active Network Programming Language", Electronic notes in theoretical computer science 71(2002), Published by Elsevier Science B. V., Germany, 2002. Available at: www.elsevier.nl/locate/entcs/volume71. html.

[15] D. S. Alexander, M. Shaw, S. M. Nettles, and J. M. Smith, "Active bridging," SIGCOMM conference, pp. 101-111, 1997.

[16] D. S. Alexander, W. A. Arbaugh, A. D. Keromytis, and J. M. Smith, "A secure active network environment architecture: realization in Switch Ware," IEEE network, vol. 12, pp. 37-45, May/June 1998.

[17] R. Morris, E. Kohler, J. Jannotti, and M.F. Kaashoek, "The Click modular router", In Proceedings of the 17th ACM Symposium on Operating Systems Principles (SOSP), pages 217-231, December 1999.

[18] D. Decasper, Z. Dittia, G. Parulkar, B. Plattner, "Router Plugins: A Modular and Extensible Software Framework for Modern High Performance Integrated Services Routers". Project at: Computer Engineering and Networks Laboratory, ETH Zurich, Switzerland and Applied Research Laboratory, Washington University, St. Louis, USA, 1999. Available at: www.tik.ee.ethz.ch & www.arl.wustl.edu.

[19] M. Fry and A. Ghosh, "Application level active networking,", Computer Networks, 31 (7) (1999) pp. 655-667. Available at: http://dmir.socs.uts. edu.au/ projects/ alan/ papers/cnis.ps.

[20] K. T. Krishnakumar and M. Sloman, "Constraint-Based Configuration of Proxylets for Programmable Networks", Proc. 8th (IDMS'2001), Lancaster, UK, 4-7 Sep 2001.

[21] A. Ghosh, "FunnelWeb v2.0.1", Online

reference, 2000. Available at: http://dmir.socs.uts.edu.au/projects/ alan.htm.

[22] R. Cardoe, J. Finney, A.C. Scott, and W.D. Shepherd, "LARA: A prototype system for supporting high performance active networking", In Proceedings of the First International Working Conference on Active Networks (IWAN), volume LNCS 1653, pages 117-131, Berlin, Germany, 1999.

[23] Microsoft Corporation, Microsoft Windows 2000 Driver Development Kit, "Network Drivers", 2000.

[24] D. A. Solomon and M. E. Russinovich, "Inside Microsoft Windows 2000", Third Edition, Microsoft Press, 2000.

[25] T. F. Divine, "NDIS IM driver samples for windows NT and higher", Online article, USA, 2006. Available at: www.pcausa.com.

[26] Microsoft Corporation, Microsoft Development Network, Platform SDK, "Networking and distributed services: Winsock version 2", 2000.

[27] J. Postel and J. Reyonlds, "File Transfer Protocol", RFC959, IETF,1985.

[28] T. Wolf, et. al., "Tags for High Performance Active Networks", Applied Research Lab., Washington University, USA, 2001. Available at: www.arl. wustl.edu.

[29] Jeff Nathan, "Nemesis packet injection utility", 2003. Available at: www. packetfactory.net /projects /nemesis/ windows.

[30] Subversive Technologies and Counter-measures Corp., "Network packet generator", Online software, 2006. Avilable at: www.wikistc.org/w/ images /3/3c /Npgl .3.0.zip.

[31] R. L. Rivest, "The RC5 encryption algorithm", MIT Lab. for computer science, USA, 1995.

موجه شبكات لتطبيقات خاصة بالاعتماد على نظام وندوز

عمر علي عذاب* احمد ستار هادي* سفيان تايه فرج**

** قسم هندسة المعلومات والاتصالات/ الهندسة الخوارزمي/ جامعة بغداد*

*** كلية الحاسبات/ جامعة الانبار*

الخلاصة

هذا البحث بدأ بوصف الاليات الاساسية للحصول على الشبكة الفعالة. تم عمل تمحيص للنظم والتقنيات المرافقة للشبكة الفعالة الموجودة حاليا. قلب هذا البحث يقدم تصميم وتنفيذ لمعمارية موجه (router) فعال جديد والذي يمكن برمجة الشبكة بمرونة بالاعتماد على "وحدات بناء المستخدم". هذا الموجه الفعال صمم ليوفر اقصى مرونة لتنمية مهام وخدمات الشبكة المستقبلية. في هذا الموجه تم استخدام نظام التشغيل ويندوز windows وتعزيز بروتوكول ANEP. ان تعزيز ANEP جعل طريقة بناء الخدمة في هذا الموجه يسمح بالبرمجة المرنة من خلال التجميع الشفاف لوحدات بناء المستخدم في طريق البيانات الداخلة للموجة. كذلك سنقدم ونطبق برنامج لتشكيل وحقن باكيتات المعلومات في طبقات الشبكة لماكنة الفحص، سوف ندعو هذا البرنامج بمولد وحاقن الباكيتات. اخيرا، نجاح معمارية العقدة وتنفيذها الابتدائي تم تقويمه باستخدام بعض التطبيقات العملية.