Australian Journal of Basic and Applied Sciences, 4(7): 1632-1640, 2010
ISSN 1991-8178

# A Protocol Analyzer Using a Sockets Based Network Sniffer

Mumtaz M.Ali AL-Mukhtar

Information Engineering College AL-Nahrain University Baghdad-Iraq

**Abstract:** A network sniffer based on a socket type of application programming interface is presented in this paper. The proposed system captures and analyzes the network traffic data packets passively without interfering with the network information flow. It extracts and views different valuable detailed information in a human-readable format concerning the monitored network in real-time. The proposed system is conducted through four main modules. Real Time Network Sniffer Module performs main system functions by passively capturing and extracting information from all packets transmitted through the network. Packets Meter Module is responsible for real-time representation of all the transmitted packets in a dynamical graphic meter. Network Statistic Module views and logs detailed information and statistics concerning main network protocols: TCP, UDP, ICMP and IP. Nodes Viewer Module is responsible for scanning and viewing the network nodes in a representative graphical mode. The proposed system has been developed employing Windows Sockets techniques that facilitates processing and dealing with network traffic and packets through its Application Programming Interface (IPC) using visual basic programming environment.

**Key words:** Computer Networks Sniffing, Network Analysis, Windows Sockets, TCP/IP Protocol.

## INTRODUCTION

Many of today's applications and communication systems rely on data networks. Networks have been growing in size rapidly and have come to support more complex operations. As result, monitoring and maintaining networks has become cumbersome and has created the need for new specified network protocol analyzers, better known as "Network Sniffers" (Pande Brajesh, 2005).

Several tools exist that monitor network traffic (Jipping Michael, 2003; Corley Michael, 2004; Misun Yu, 2007; Inoue, D., 2008; Kaushik, A.K., 2010). However, most standalone network traffic monitoring tools are quite expensive and those freely available on general purpose platforms (e.g. Linux or Windows) are quite cryptic.

This paper presents the design and implementation of a network sniffer, based on a socket type of application programming interface. It allows a host to capture any packets in an Ethernet network by putting the network interface card (NIC) into a promiscuous mode. This mode allows the NIC to blindly receive all network packets (Berry Paul, 2007; Trabelsi Zouheir and Rhamani Hamza, 2004), i.e., packets that are not supposed to arrive to the designated host are no more blocked by the NIC.

The proposed network sniffer snoops packets transferred through the network, and collects various measurements and statistics. It enables network managers to evaluate and examine the data running on their networks. Besides its usage in the technical environment, the proposed network sniffer can gain access into confidential documents and cause intrusion into anyone's privacy.

### *2. Sockets:*

Sockets are communication channels that enable unrelated processes to exchange data locally and across networks. Sockets can be created in pairs, given names, or used to rendezvous with other sockets in a communication domain, accepting connections from these sockets or exchanging messages with them (Quin Bob and Shute Dave, 1996).

Sockets provide a sufficiently general interface to allow network-based applications to be constructed independently of the underlying communication facilities. They also support the construction of distributed programs built on top of communication primitives.

**Corresponding Author:** Mumtaz M.Ali AL-Mukhtar, Information Engineering College AL-Nahrain University Baghdad-Iraq
E-mail: Mumtaz_almukhtar@yahoo.com

Socket subroutines and network library subroutines provide the building blocks for Inter Process Communication (IPC). An application program must perform the following basic functions to conduct IPC through the socket layer:

- Create and name sockets.
- Accept and make socket connections.
- Translate network addresses.
- Shut down socket operations.

### *2.1 Microsoft Windows Sockets (Winsock):*

Short for Windows Socket, Winsock is an Application Programming Interface (API) for developing Windows programs that can communicate with other machines via the TCP/IP protocol. Microsoft Windows comes with Dynamic Link Library (DLL) called winsock.dll that implements the API and acts as the glue between Windows programs and TCP/IP connections (Comer Douglas E., 2000).

Two major versions of WinSock exist for Windows, which are Winsock 1.1 and Winsock 2.2. Winsock 2.2 is the latest Microsoft windows Socket version which has more capabilities than previous versions for network applications. One of the primary goals of Winsock 2.2 has been to provide a protocol-independent interface fully capable of supporting the emerging networking capabilities.

Winsock is an interface, and so it does not in any way affect the bits on the wire and that gives the applied system the capability of processing all the network data passively and without effecting or interfering with data.

Winsock 2.2 is the version that has been used in the proposed system. This is due to its capabilities that enabled the proposed system to achieve its development goals in real-time processing and in offering wide flexibility in dealing with networks traffic (Stevens, W., 2004).

### *2.2 Sockets Types:*

The type of sockets describes the communication protocol it uses to send data. There are three sockets types (Bohrouz A. Forouzan, 2007):

#### *a. Stream Sockets:*

Processes that communicate through sockets of this type must connect explicitly to a socket, and the data transfer from the client to the server (or vice versa) takes place through a reliable byte stream.

#### *b. Datagram Sockets:*

They offer "connection-less" semantics with datagrams. Connections are implicit rather than explicit as with streams. Either party sends datagrams as needed and waits for the other to respond. Messages can be lost in transmission or received out of order.

#### *c. Raw Sockets:*

Raw sockets allow network packets to be defined over the IP layer. The raw socket does not format the data; whatever value entered in the packet byte array will be forwarded "as-is" to the remote host.

The proposed system uses this type of sockets which have the advantage of custom level protocol development. Rather than going through the normal layers of encapsulation/decapsulation that the TCP/IP stack demands, packets are passed to the SBNA system which is now responsible for stripping the headers and payload analysis.

## *3. System Architecture:*

The proposed Sockets Based Network Sniffer (SBNS) system consists of four main modules:

1. Real-Time Network Sniffer (RTNS).
2. Nodes Viewer (NV).
3. Network Statistics (NS).
4. Packets Meter (PM).

Two additional support modules are also integrated with the main modules:

- Initialization Module that initiates all what SBNS system needs to start network data processing and analysis.
- System Results Output module that implements several synchronization processes in order to output the results in a real-time manner.

Figure 1 illustrates the SBNS system architecture and interaction among its modules. These modules are responsible for collecting raw network data regarding all the nodes in the network (promiscuous mode). The data is processed and analyzed to create useful information. At the top of the system, the statistics, data display and user interface functions are implemented. The obtained information are formatted and displayed to the user through the GUI as textual/or graphical information and recorded in log files. The proposed system main modules are discussed in the following subsections.

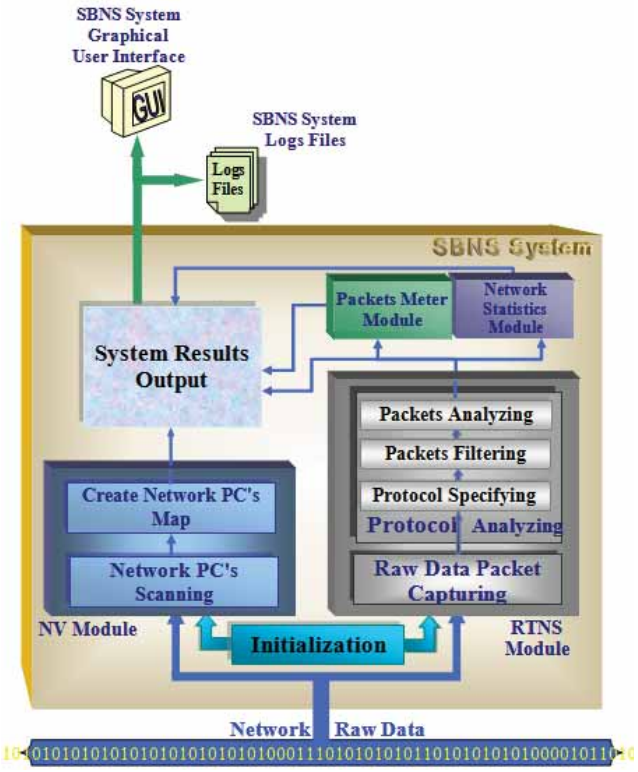


**Fig. 1:** SBNS System Architecture

### *3.1 Real-Time Network Sniffer Module (RTNS):*

RTNS Module is the most important module in the SBNS system. It comprises two sub-modules as shown in figure 1.

1. Raw Data Packet Capturing.

It is responsible for capturing all the raw data packets from the monitored network according to the previous socket setting.

2. Protocol Analyzing.

It comprises three functions:

- Protocol Specifying

This function conducts determining the captured data protocol by checking specific raw data values that are related to a protocol structure.

- Packet Filtering

Each packet traveling on the network is matched against a set of rules in order to determine whether it is interesting for a particular application. Since the amount of traffic traveling on a network segment can be huge, filtering out irrelevant traffic is an essential step to reduce the demand in terms of storage and processing on the sniffing and analysis.

- Packet Analyzing

This function performs activities related to observing and analyzing networks protocols in detail. It categorizes data types, reconstruct transferred information, and maps protocol usage to a pattern of application usage.

Figure 2 gives the SBNS flowchart which illustrates the detailed steps involved in the module functioning.

***1. Initializing Windows Sockets Functions:***

This step initiates the functions of Ws2_32.dll which represents the dynamic link library of windows socket version 2.2. This is accomplished by calling windows socket function WSAstartup(&H202, data).

***2. Obtaining Host IP:***

The host IP must be obtained in order to be the address for the socket that will be created in the next step. This is done by execution three windows socket functions in sequence: gethostname(name, namelen), gethostbyname (name), and ntoa(hostnamepointer).

***3. Socket Creating:***

In order to deal with network data by using Winsock API functions, a socket must be created by calling the windows socket function socket (af, type, protocol).

***4. Socket Creation Errors Checking:***

This is done by checking the socket descriptor value. A value <1 means an error has occurred. In this case the windows socket function *WsACleanup* will be called to terminate the use of the Ws2_32.dll. In addition *closesocket* function is called to terminate and close all the open and created sockets.

***5. Socket Binding:***

When a socket is created with a socket function, it will exist in a name space (address family), but it has no name assigned. The windows socket function ***bind ( s , add , namlen )*** will establishes the local association of the socket by assigning a local name to an unnamed socket.

***6. Checking Socket Binding Errors:***

If the binding function returns a none zero, this indicates an error has occurred. Therefore the error handling routine will be called.

***7. Set The Socket Mode To Promiscuous Mode:***

In order to control the mode of the socket and set it to promiscuous mode, the windows socket Input/Output Control function *WSAIoctl* is called.

***8. Checking Socket Promiscuous Mode Setting Errors:***

If the *WSAIoctl* function returns none zero value which means an error has occurred, the error handling routine will be called.

***9. Set The Socket Mode To Real Time IP Packets Capturing:***

To achieve a real-time network IP Packets capturing, the windows socket Synchronization function *WSAAsyncSelect* is called.

***10. Checking Socket Real Time Setting Errors:***

If the *WSAAsyncSelect* function returns a none zero indicating an error, the error handling routine will be called.

***11. Listening and Waiting For Network Events Occurrence:***

The application now listens and waits for network events occurrence to receive all the captured packets from the bounded socket.

***12. Receiving Network Data from Bounded Socket:***

When there are network's events occurring, the application enters a loop involving the following processes in sequence:

a. Calling the windows socket function ***recv (S, Buffer, Len, Flags)***. *This function receives data from a bound socket, and returns the number of received bytes.*
b. Transfer all the information in the first field of the IP datagram which represents the IP header from the *recv* function buffer to a custom data type structure which acts as a template that includes the captured raw data.

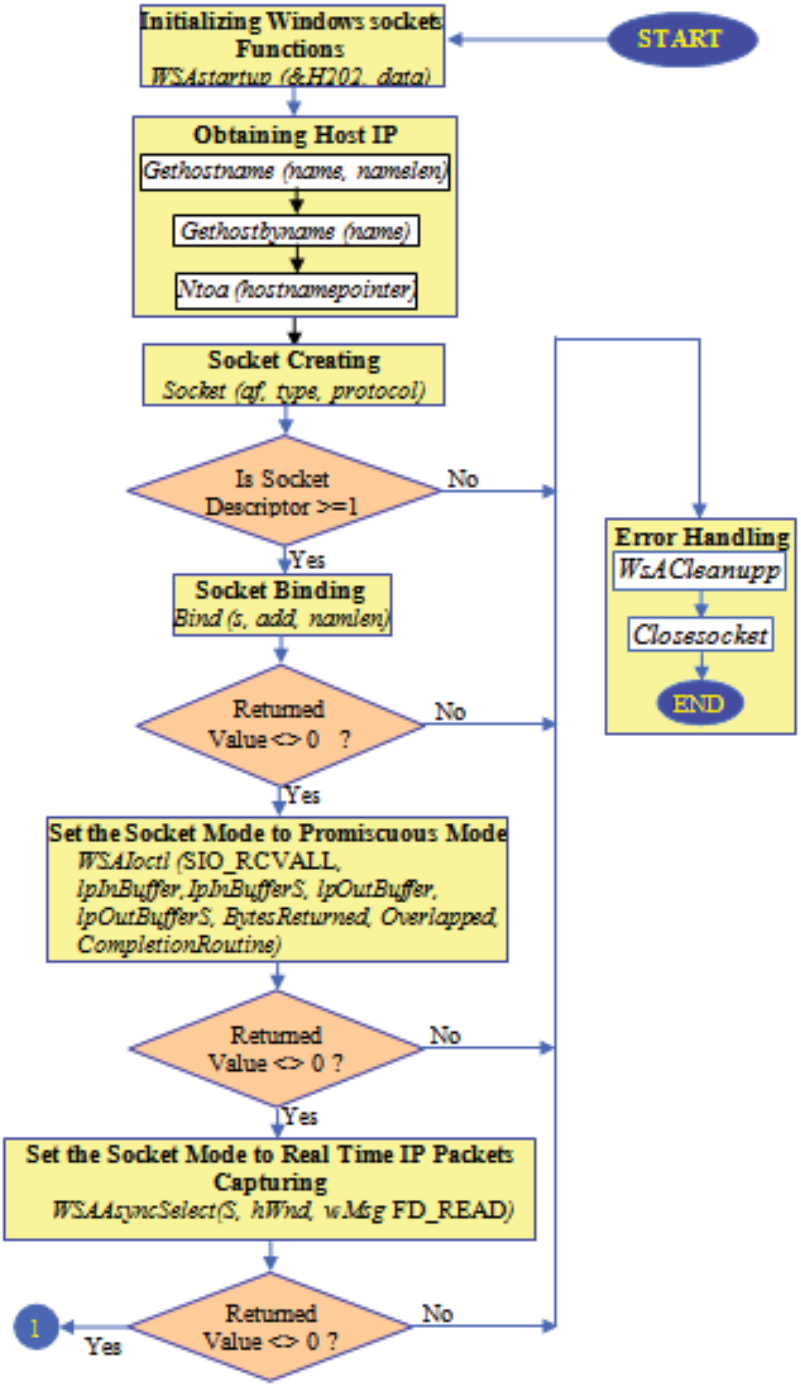


**Fig. 2:** SBNS System Flowchart

c. The application will detect the protocol type according to the received raw data and execute the following actions according to the protocol type:

- Sending the processed IP Datagram to the other modules, i.e., *Packets Meter and Network Statistics Modules.*

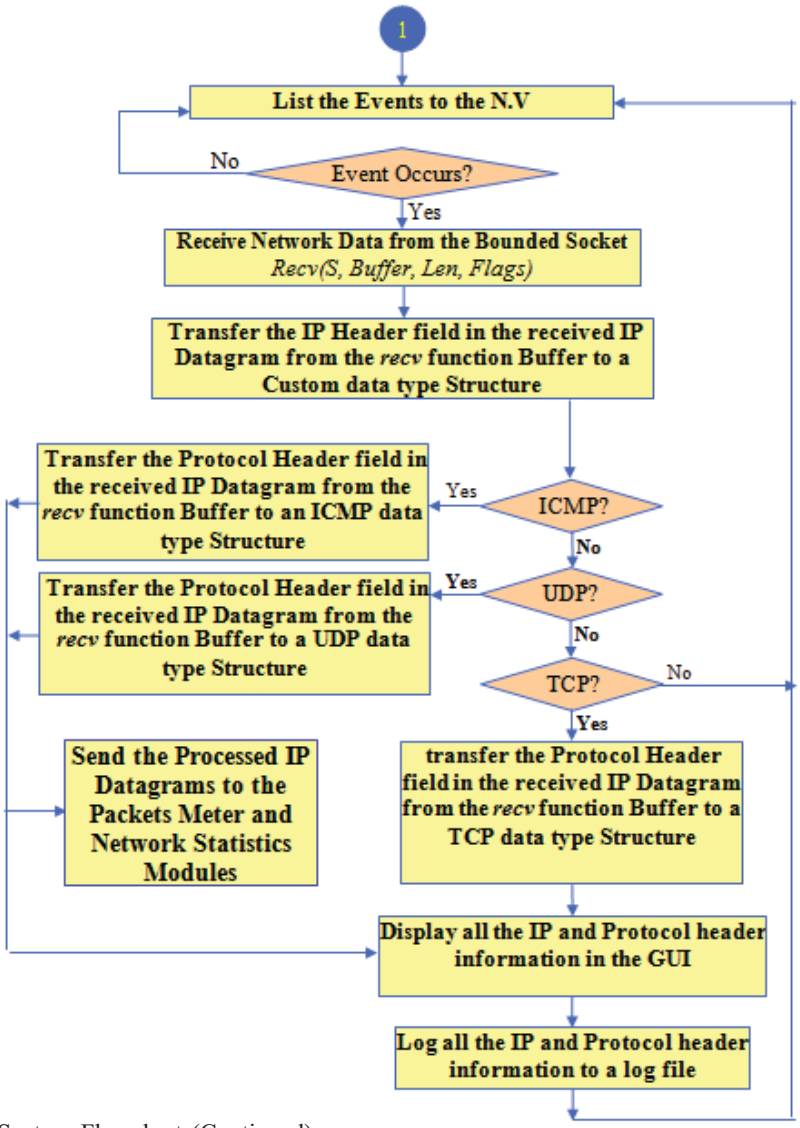


**Fig. 2:** SBNS System Flowchart (Continued)

- Displaying each captured packet IP Header and protocol header information.
- Logging each captured packet IP and the protocol headers information.

### *3.2 Nodes Viewer Module (NV):*

Nodes Viewer Module (NV) is responsible for:

- Graphically demonstrating the network map and viewing the map nodes IPs.
- Displaying the connection statistics of the network nodes in a real- time.
- Logging the network map into logs files.

To achieve the above mentioned abilities the Nodes Viewer Module (NV) has four log text files:

1. Network Map (NM) Log File: This log file records the network nodes IPs in two modes. The old NM that stores the previous scanning results and the new NM that records the new results for the same nodes.
2. Ping Log File: This log file records the obtained reports from the pinging process using the Packet Internet Groper command which is a DOS command installed as a part of Windows to test nodes connection over a TCP/IP network.

3. Active Nodes Log File: This log file records the active nodes that are functional and currently connected to the network.
4. Dormant Nodes Log File: This log file records the nodes that were connected in the previous lunch but disconnected currently.

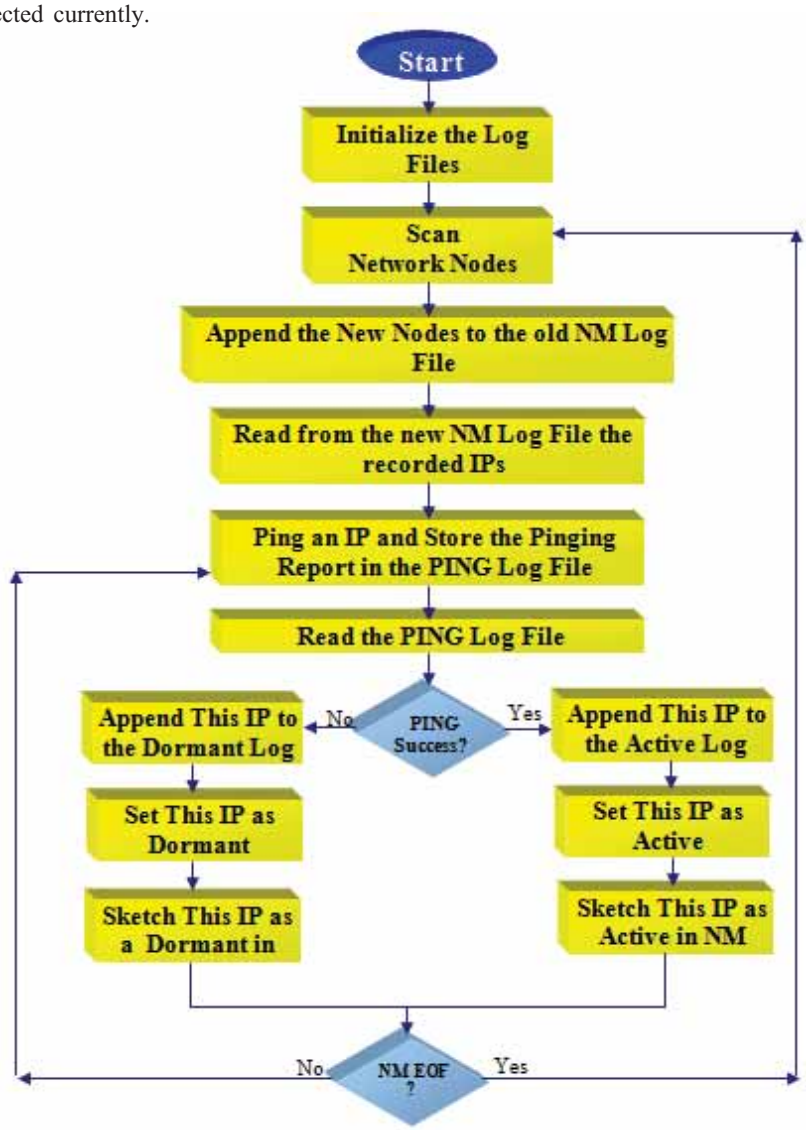


**Fig. 3:** Nodes Viewer (NV) Module Flowchart

To achieve the real-time concept the scanning process is implemented by the NV module each 0.1 second. Figure 3 demonstrate the processes carried out by the NV module.

### *3.3 Network Statistics Module:*

The windows socket application programming interface *Getting Statistics* function is used in the Network Statistics Module to get a detailed statistics information about the TCP/IP Protocols. For achieving the real time performance manner, all the implementation steps except the initialization will be refreshed every 0.1 second. The processes involved in this module are illustrated in figure 4.

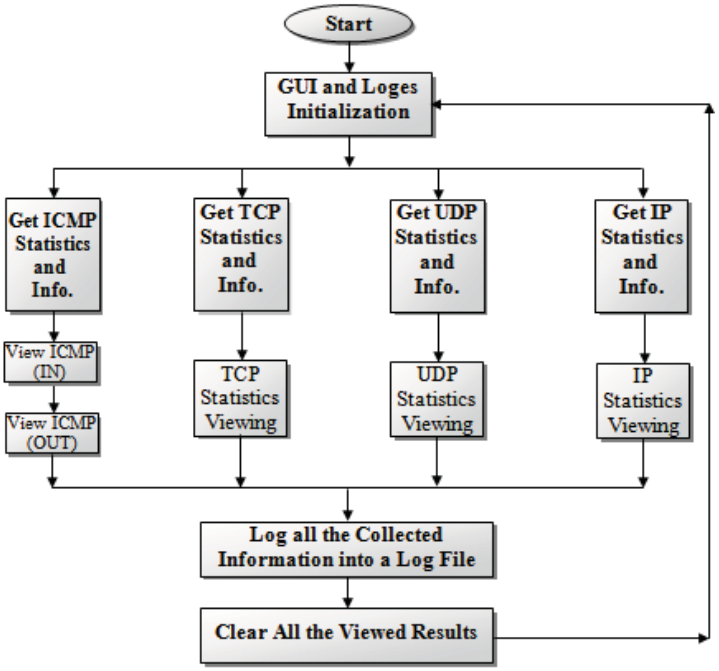


**Fig. 4:** Network Statistics Flowchart

### *3.4 Packets Meter Module*

It is a graphical dynamic meter that graphically represents all the transmitted packets through the monitored network along with their sizes. Figure 5 shows the Packets Meter flowchart. The Packets Meter module starts with the initialization step necessary to build the graphical meter. As it gets the data concerning the captured packets from the RTNS module, it determines the size of each captured packet, i.e. the number of bytes of the entire IP packet, including the data and header.

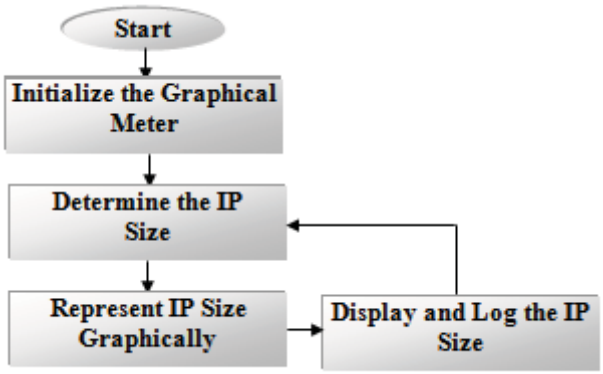


**Fig. 5:** Packets Meter Flowchart

Thereafter, the module will display the IP packet size graphical representation in the SBNS System GUI in addition to the logging process of the Packet size and its transmitted time. The next recurrent step is to determine the size of the next captured packet size if any is transferred throughout the sniffed network.

## *4. Conclusions:*

Through out the SBNS System development, many conclusions were obtained; they are as follows:

1. The proposed system adopted Windows Socket (WinSocket) technique as being an object through which the program can send and receive data across the network. Sockets simplify the program development because the programmer need only to worry about manipulating the sockets and can rely on the operating system to conduct operations across the network correctly, and that gives a wide flexibility in dealing with the network traffic data packets.
2. Using the socket type "Raw-Socket" allows having an absolute control over the data that's being sent or received through all the network nodes and provides an overwhelming power for the SBNS System to achieve its goals in real time
3. The SBNS System managed the most used TCP/IP protocols. This is achieved through out setting its Socket Address Family (AF) to AF_INET that corresponds to the standard Internet Address Family and enable the sockets to deal with the TCP/IP protocols Suite.
4. SBNS system managed a real-time manner sniffing and analyzing the traffic through the entire network. This has been realized passively without effecting these transmitted packets.
5. SBNS system could be used as an educational resource for learning about protocols by providing detailed information and statistics concerning the monitored protocols.
6. Capturing and analyzing the network data flow in real-time can be used by the network administrators for evaluating and diagnosing network related problems, network intrusion detection and network traffic logging.
7. The proposed system can be used with law enforcement agencies for national security and helping crime prevention and detection.

## REFERENCES

Bohrouz A. Forouzan, 2007. "*Introduction to data Communication and Networking*", Fourth Edition, McGraw-Hill Inc.

Berry Paul, 2007. "*An Ajax-Enhanced Web-Based Ethernet Analyzer*", Linux Journal, 157: 20-31.

Corley Michael, Weir Michael, Nelson Kenric and Karam Andrew, 2004. "*Simplified Protocol Capture (SIMPCAP)*", Proceedings of the IEEE Workshop on Information Assurance, United States Military Academy, pp: 176-182.

Comer Douglas E., L. Stevens David, 2000. "*Internetworking with TCP/IP, Vol. III: Client-Server Programming and Applications, Linux/Posix Sockets Version*", Prentice Hall.

Inoue, D., M. Eto, K. Yoshioka, S. Baba, K. Suzuki, J. Nakazato, K. Ohtaka and K. Nakao, 2008. “nicter: An Incident Analysis System Toward Binding Network Monitoring with Malware Analysis”, Workshop on Information Security Threats Data Collection and Sharing, pp: 58-66.

Jipping Michael, Bugaj Agata, Mihalkova Li Liyana and Porter Donald, 2003. "*Using Java to Teach Networking Concepts with a Programmable Network Sniffer*", SIGCSE Conference, Nevada, USA, pp: 120-124.

Kaushik, A.K., E.S. Pilli and R.C. Joshi, 2010. “Network Forensic System for Port Scanning Attack”, Second International Conference on Advanced Computing (IACC), pp: 310-315.

Misun Yu, Haeyong Kim and Pyeongsoo Mah, 2007. “*NanoMon: An Adaptable Sensor Network Monitoring Software*”, IEEE International Symposium on Consumer Electronics, ISCE, pp: 1-6.

Pande Brajesh, Gupta Deepak, Sanghi Dheeraj and Jain Sanjay, 2005. "*The Network Monitoring Tool-PickPacket*", Proceedings of the Third International Conference on Information Technology and Applications (ICITA'05), 2: 191-196.

Quin Bob and Shute Dave, 1996. "*Windows Sockets Networks programming*", Addison-Wesely.

Stevens, W., Richard, Fenner Bill and M. Rudoff Andrew, 2004. "*Unix Network Programming, Volume 1: The Sockets Networking API*", 3/e, Addison-Wesley Professional.

Trabelsi Zouheir and Rhamani Hamza, 2004. "*Promiscuous Mode Detection Platform*", The Second International Workshop on security in Information Systems (WOSIS-2004), Portugal, 13-14.